A Tool for Web-based Management of Call-for-Papers

Said Nuh

Kongens Lyngby 2007
IMM-BSC-2007-100

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-BSC: ISSN 2007-100

Summary

In this document we are going to identify and outline (and communicate) the purpose, requirements and context for a paper management service that is to be implemented in order to ease the task of managing and relaying Call For Paper documents between the participating entities in a conference.

The goal is to develop a tool that can be deployed as an intra- and inter-institutional application. The overall proposed system consists of a schema-centric language grammar that governs XML documents in an arbitrarily chosen dialect, a transformation vocabulary that is used to convert these documents into other formats: SQL, (X)HTML.

Other modules include a toolset that can be deployed to validate/generate documents in that grammar and a scalable document database. Some interfaces are devised to enable interaction:

- Mail: fetching and managing mail-repository at a remote server
- Web: submitting document by either uploading document or entering data into a web-form.
- Client: A preliminary local client has been developed earlier. Later remarks made by Chris suggested that further development of this client may not be worthwhile to pursue.

Addendum:

The relevant documents mentioned above are found in the annex to this document. Source code, among other relevant files is found on the attached CD-ROM.

Preface

This thesis was prepared at Informatics Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the the B.Sc. degree in software technology.

The project was supervised by Christian Probst, Assistant professor at the Informatics and Mathematical Modelling institute, DTU.

I thank Chris for his encouragement and enduring patience. Due to his affiliation with the targeted audience, Chris' insightful comments has been an extremely valuable asset during the requirement extraction process. Thanks, Chris.

Lyngby, October 2007

Said Nuh

To my late father. For the love and inspiration.

Contents

0.1 Introduction

Since Sir Tim Berners-Lee proposed the idea of hypertext and the distributed hypertext system, both the in-depth understanding of it's uses/users and the specifications that describe the connectivity of it's components, have taken many sharp turns.

Ahead of the ongoing transition of the World Wide Web from a set of isolated web-sites to more of a computing platform[1] , an application was said to be residing in the end-user's machine, necessitating a 'download'. Software previously distributed as local applications are eclipsed by applications available as online *services*.

These web-services with an increasingly large spectrum of functionalities have gained momentum, and the web browser has became a universal multi-platform, multi-purpose entity which has become increasingly significant.

The World Wide Web was conceived as a information retrieval tool built to facilitate sharing and updating information among physicists at he CERN laboratory, and has since become a telecommunications revolution. In this project we will device a web-application that puts a small part of 'academic' back into the World Wide Web.

The main objective of this project was to develop:

- A service-oriented, schema-aware, web-native, document management computing system according to the *Software as a service* application delivery model,
- A local client that can generate documents that are associated with that application.

Though neither required nor necessitated by an urgent user-need, the local client was developed early in the process and was later rendered redundant by other interfaces in the system, still some domain problems can only be solved through this local client.

[1] A phenomena recently coined as ´Web 2.0', the concepts of which can be traced back to the early '90s

0.1.1 Academic Conferences

By a `paper`, we understand an academic work that is aimed for publishing. A 'Call for Paper' (CFP) is a document that contains principal information about an event - mostly academic - and is sent to participating parties and prospective presenters to collect conference presentations and articles.

The main purpose of academic conferences is exchange of information among researchers with a common interest. Papers submitted to the conference are subject to editorial refereeing to qualify texts for publication. As conferences are organized around a particular (albeit broad) topic, a Call For Paper is usually identified by it's acronym, which is usually a memorable short name composed of the initial letters or syllables of the topic.

Some disciplines require presenters to submit a paper of about 12-15 pages, which is peer reviewed by members of the program committee or referees chosen by them. Providing a rudimentary tool that enables data-exchange between users using (possibly) dissimilar systems, is the main purpose.
Instead of deploying a freely constructable, open-format document, some semantic constraints are applied to enhance accessibility and consistency.

Although a CFP can describe other types of events (workshops, journals, etc) this project mainly focuses on data that is intrinsically bound to, or are relevant to, conferences.

0.1.2 Related Work

Some sites that share some characteristics with or are related to this project, and whose aim is to bridge this document-exchange gap, have been brought to my attention:

wikicfp.com :
Chris sent me link to this site. The initial impetus of this site, as the name *wiki* suggests, might have been to establish a network of users who can customize and share lists containing conferences on topics of interest. The registration is open and simple; neither user registration data nor CFP data submitted are verified or checked, merely date and title-fields are checked for emptiness. Site contains a relatively large amount of entries, roughly 2000, a considerable growth given that the site has been active in less than a year. Site can accommodate CFPs categorized into different subjects. Multi-user site with a simple, plain interface.

papersinvited.com :

This site is a multidisciplinary service that contains an exhaustive list of Calls for Papers that are submitted by scientists, professors and students alike. I have acquired a temporary single-login access to the site and have seen most the of inner utilities.

While the site claims to have (and probably has) "*world's largest listing of Calls for Papers*", the submission pages seem to have no dynamic content behind it. The layout of the listings are nice and simple, most of which are pre-rendered static html pages. The site supports many types of events across a multitude of categories, all of which users can *subscribe* to. I have not tested their notification method, but mails are dispatched to the subscribing users, presumably. Site only grants access to institutional subscribers, while individuals can request a single-time login access for evaluation purposes. This service focuses heavily on it's ability to alert subscribers: users can receive by email all calls for papers in chosen areas of specialization.

As for both pages, there is lack of transparency and external interfaces, through which users can interact with the sites. Both require users to register and submit data pertaining to CFPs through a web-form. There is no contractual binding between the submitted documents and set of agreed-on rules.

0.1.3 Test Environment

This application has been deployed and tested rigorously on a remote server. DTU UNIX databar servers (student.dtu.dk) do not provide all the necessary facilities needed. The following addresses are valid:

- Application server: cfp.smallmeans.com
- Database server: db.smallmeans.com
- Mail server: mail.google.com.

Interfaces and APIs accessed are:

- Google Maps^{TM}- mapping service application provided by Google
- SIMILE Timeline

- Mail-account used as a repository :cfp.imm@gmail.com

It is not necessary to have an account to access some parts of the system. Guest accounts can be used (limited functionality)

0.2 Terminology and Definitions

Here, we define terms and abbreviations that are deemed important and are used throughout this document. Most of these definitions are coined or derived from various ISO and W3C specifications.

W3C : The World Wide Web Consortium is an international industry consortium that functions as governing body for the development of web standards and specifications (http://www.w3.org)

W3C Recommendation : W3C working groups has developed and deployed many widely applied technologies. A technology or a standard is said to be a "W3C Recommendation" when it's incubation period has ended and the development has reached the final stage of the ratification process [W3P].

Below are subset of the XML-Based W3C Recommendations that will appear repeatedly throughout the document.

XML : eXtensible Markup Language XML is a human-readable, machine-understandable syntax for describing hierarchical data.

XPath XML Path Language. Used to extract subsets of the data stored within an XML document. XPath is an indispensable subset of XSL.

XSL : e**X**tensible **S**tyle **L**anguage consists of two W3C recommendations:

- XSL Transformations (XSLT): used to transform one XML document into another (format)
- XSL Formatting Objects (XSL-FO): used to specify the presentation of an XML document. Currently, few applications and fewer or no Web browsers can display a document written with XSL formatting objects. XSL-FO is to XSL what CSS is to (X)HTML documents.

The transformation and formatting subsets can be used independently of each other. XSL-FO, due to it's limited support, is not used in this project.

XML Schema [HSTM04] The XML Schema allows us to create vocabularies with XML by adding further restrictions to the core XML rules

XHTML : A restrictive subset of SGML that has more or less replaced HTML 4.0.

Web Application (Webapp) By a web application, we understand applications, usually a three-tier architecture, that are accessed over a network such as internet/intranet via web-clients.

User Interface (UI) :
UI refers to the graphical or textual elements of a software, through which a user can interact with the program. A menu, button, toolbar or command line interfaces are some examples thereto.

Shared Environment :
An operating server environment where PHP runs as an Apache module and as such has read access to all files accessible by the web-server regardless of the owner.

Chapter 1

Domain analysis

In following two chapters, we will use methods derived from interdisciplinary fields such as Software and Requirement Engineering (SE) to help us conceive a product that meets the user(s) implied and stated needs.

1.1 Introduction

This section is loosely based on IEEE's blueprint [IEE98] for Software Requirement Specifications (SRS). Throughout this chapter, the name "CFPMan" shall refer to the document management system and set of subsystems that are developed under project 19, [Pro].

This document will be kept up to date as changes are made and as we gain more knowledge about the domain, the analysis does not attempt to be domain-exhaustive; completeness is not essential in this phase.

- Before we can design the software, we must know its requirements.
- Before requirements can be expressed, we must understand the domain to which the application belongs.

So it follows, from this dogma, that we need to :

- first establish precise description of the domain(s);
- then from such, derive at least the domain requirements;
- and from those and other requirements[1] outline the design of the software.

Despite the limited time and the abundance of the tasks that are expected accomplished in the project, we will attempt - by employing some effort - to compose this through the software engineering process that I (hopefully) have accumulated - specifically referring to the concept of "TripTych [Bjø05] software development process model" and lately, to thoughts obtained through Prof. Dines Bjørner's "Software Engineering" course. This treatment is going to be (mostly) informal, but precise and limited to the significant parts of that concept.

In chapter 2, we will analyze a "grand-scale" of the project, and take a closer look at the domain of this project to unravel some of the complex structure that lie behind the infrastructure components. A pragmatic description of the process is expected in this and the subsequent (design) chapters.

[1]Be it Interface, Machine or Maintenance requirements

1.1.1 Purpose

Let's begin with the project subsystems' description in *verbatim*,[Pro]:

- Design of a XML schema definition,
- Development of a stand-alone client for creating documents in that XML dialect,
- Development of a web-based interface,
- Development of a web service for handling documents in the XML dialect, and,
- Integration with a database.

1.1.2 User Characteristics (Audience)

The paper management service is is intended primarily for scientists, professors and post-docs who are seeking an easy-to-use web-based tool to handle CFPs, but who are not necessarily familiar with the technologies or the semantics of the tasks done at the webservice level. Thus, great emphasis is placed on applying simple, accurate and detailed methods to access the data within the system. These implied set of users can be extended at a later point.

1.1.3 Product Scope

In this section, we will try to build a common understanding if what is included in, or excluded from, this project - i.e scope, limitations and expectations. This is done so the participating parts have the same perception of the scope of this project, and the client reviews the developer's interpretation - and validates it - i.e the goal is to introduce mutual understanding.

1.1.3.1 Stakeholder Details

Stakeholders are the parties who affect, or can be affected by the proposed solution.

This service is developed by B.Sc. student Said Nuh.
The project was supervised by Christian Probst, Assistant professor at the Informatics and Mathematical Modelling institute, DTU.

User representative :
Christian, in his capacity as a professor and project proposer, is also the *client* representative, and has been acting as the sole contributor to the requirement extraction process.

In this context a very narrow set of stakeholders are considered: familiarity with tools - such as specifically required browser types - is assumed. In an eventual extension to this application, a full set of stakeholders might be appropriate, such as professors and academic institutions that might use the system.

1.1.3.2 Current status

The following description is solely based on knowledge acquired during conversations with Christian, and may be incomplete but is applicable to most, but not necessarily all, the targeted audience.

The current CFP management systems are based on mailing-lists, where Call For Papers are usually distributed. This mechanism is characterized by:

- High possibility of having too much information to remain informed about topics of interest.
- Large amounts of archived information to dig through, should a need to find an old entry arise.
- Low *signal-to-noise* ratio, ie. the ratio of useful information to irrelevant data.
- Data being in *stacked*, non-visualizable format, where searching is the only facility to obtain data about relevant conferences.

- Event flood: presenters usually submit papers to different conferences as to have other options, should their paper be rejected at reviewing. As papers submission deadlines may interleave, having to keep an eye on these *important dates* may be overwhelming or demanding, at best.

Other non-electronic methods are also utilized: CFP may be printed out and put on places where potential users have access to.

1.1.3.3 Details

Since the project is intrinsically multi-parted, I have decided to start with the essential sub-systems and proceed down-wards as to overcome potential time and/or resource constraints that may surface. These sub-projects are expected to be brought to closure in sequential phases that are deemed as "prototypes". Having consulted my supervisor, these are the sub-projects in descending order of precedence, or rather in descending order of 'deliverables':

Deliverables: Apart from the release candidate of the paper management service, the deliverables for the project include (subject to changes):

- First Prototype
  - Design of a XML schema definition,
  - Development of a web-based interface,
  - Integration with a database,
- Second Prototype (phase #2) The 2nd prototype includes, but is not limited to:
  - Development of a stand-alone client for creating and validating documents in that XML dialect,
  - Development of a web service for handling documents in the XML dialect.

  The components completed in the first prototype are likely to be revised in subsequent phases.
- User Manual,
- Client acceptance plan.

1.1.3.4 Limitations, Expectations

See an elaborate description in 2.1.1

- **Disabled components**: If the user has disabled active scripting in their browser, the system may not function properly, if at all.
- **Persistent connection**: During request-intensive operations, such as browsing the directory or submitting data, a persistent Internet connection is assumed.

1.1.3.5 Vision: goals and objectives

The objective of this project is to build a web-based application with a multi-tier software architecture. More precisely, we are concerned about a three-tier architecture with the following layers:

- The user interface,
- Functional process logic,
- Persistent data access and storage.

Further objectives include: to build a user-friendly, platform-independent web-based tool that:

- conforms to current web standards and regulations,
- responds to user inquiries in real time: data filtering, queries supported,
- is multi-interfaced, extensible paper management and tracking system,
- can be incorporated into large educational institutions or mass public environments,
- is capable of modularization and is visualizable in a highly customizable manner.

1.1.3.6 Stakeholder Profiles

Below is a short description of the relevant benefits that this resource shall provide. From the users' perspective:

a) Automation of previously manual tasks, such as labeling, sorting, etc.

b) Improved productivity and efficiency by reducing time spent on shuffling through mail content to find a specific CFP.

c) To obtain feasible benefits as a result of streamlined document handling operations, information handling and provision: all of which should result in enhanced usability.

d) To remedy deficiencies in existing systems or methods used by the targeted audience.

e) To expand the business by enabling easy interaction between conference organizer and presenters.

f) Increased ability to both distribute and ingest Call For Paper documents easily, thus increasing user satisfaction.

1.2 Overall Description

1.2.1 System features

This section illustrates organizing high-level functional capabilities and requirements for the product by system ´features', delineating the major services provided by the product. This section also covers features that were assigned to Prototype #2. Priorities are given in order of importance or urgency of the feature.

These features will be used to extract the necessary software capabilities that must be present in order for the user to carry out the services provided by the feature.

Besides some descriptive text, the following are provided:

Stimulus/Response Sequences: List of the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.

1.2.1.1 Ensure well-formedness

Description and Priority : Prior to any processing of a document, well-formedness is assumed. This feature is given a high priority.

Stimulus :
User chooses to transfer a document - of a proper type - to the specified remote computer. User initiates transfer by applying relevant actions on an appropriate UI, such as the UPLOAD form provided to him/her.

Response Sequences :
a) if the document is found to be in violation with the well-formedness constraints defined in 2.1.5.1, then process is halted and user given an appropriate description.
b) otherwise, the user is forwarded to an appropriate response page.

1.2.1.2 Ensure adherence to Schema

If only well-formedness is required, XML can be used as a generic framework for storing any amount of data that can fit into the document tree. But, besides well-formedness, validity is a prerequisite for proper data post-processing. Input data must be correct both in context and content. This feature ensures adherence to a given DTD or XML schema document.

Description and Priority :
Before the data can be stored into the database, it is urgently necessary that the data be validated through a carefully planned sequence of procedures to increase application security and circumvent unanticipated or invalid data to get propagated into the database. This feature has a hight priority.

Stimulus/Response Sequences :

Stimulus : A request is be initiated through one of the interfaces form, upload, or mail (1.2.1.3, 1.2.1.4, 1.2.1.3, respectively)

Response :
Document is validated against a schema

Response :

User is forwarded to an appropriate response page.

1.2.1.3 Web Interface: input form

This system feature pertains to capabilities of the web-form.

Description and Priority : A user, who has been granted access as either a registered user or an administrative *super-user* may add CFP documents into the system. A user, whose identity has been established as "guest" is not authorized to request this functionality.

The web form interface has a data entry page, where the user is able to submit data related to the CFP, such as title/name, acronym, location, etc. This feature is of a high priority.

Stimulus/Response Sequences :

Stimulus : User enters some data into the form, and submits the entry form by interacting with an appropriate UI element, e.g. the "submit" button or by pressing the ENTER key

Response : Input values are transformed into an XML document

1.2.1.4 Web Interface: upload form

Users **must** be able to transform documents into the system. This transfer must use an web-form to upload documents as users are accustomed to. An upper limit might be put on the uploaded document's size, enforced by the machines at the receiving end.

1.2.1.5 Mail-interface

Interface mechanism: an email is dispatched from an authorized email address to an administratively assigned harvesting email address.

Fetching and processing newly arrived emails should be done at an acceptably short intervals. Two main concerns are addressed: **Exhaustion**: not less than once every 15 minutes, and no more than every 30 minutes. **Coherence**: If the intervals are far part, information coherence may be lost. Users who have sent document in, and have received a confirmation stating that validation is pending, might not get the expected response in due time. Also, if a high rate of new documents are being sent to the mail-server, capacity of the mail account or the number of messages that can be processed at once, may restrict the use.

1.2.2 Rules And Regulations

Characterization. By a domain **rule** we shall understand some text (in the domain) which prescribes how people or equipment are expected to behave when dispatching their duty, respectively when performing their function.
Characterization. By a domain **regulation** we shall understand some text (in the domain) which prescribes what remedial actions that are to be taken when it is decided that a rule has not been followed according to its intention.

- **Rule**: Only registered users can submit document to the system.
  Regulation: Users who do not enough privileges to either a)submit documents or b)edit documents that are already submitted, are only granted privileges to view submitted document.
- **Rule**: Users may not submit incorrect details concerning themselves or conferences.
  Regulation: The system is impowered to rectify or erase any incomplete, inaccurate or outdated personal data retained by the system in connection with a)harvesting of incoming mails, b)submission of documents through the web-interface.
- **Rule**:Personal data might be made available to the public.
  Regulation: By providing information to this application, users acknowledge and consent to the collection and disclosure of personally identifying data of the type and for the limited purposes described below:
  - Name, institute and email-address are used to identity members who either organizing or presenting contributions at a conference. Peer-reviewers might also listed on conference pages.

Chapter 2

Requirement Model for Document Management System

Requirements are used to establish the basis for agreement between the users and the developer(s) on what the software product must or is expected to do.

Feedback from the stakeholders and iteration have been used to gain consensus about the requirements of the project. when outlining the requirements, these stakeholders are not likely to be able to provide the developer with a set of requirements. Two sets of requirements are considered:

- Stated requirements: these are the requirements explicitly put forward by the users of this application,
- Implied requirements : these are the expectations or assumptions that the user had in mind, but not necessarily stated.

If a requirement is *stated*, nonconformity, ie. nonfulfillment of that specified requirement, is easy to establish, whereas conformity of implied requirements are quite difficult to determine.

To signify the *weight* of the requirements in the sections below the prescriptive keywords "**must**", "**must not**", "**required**", "**shall**", "**shall not**", "**should**", "**should not**", "**recommended**", "**may**", and "**optional**" in the sections below are to be interpreted as described in RFC 2119.

`Definitions,`[Bra97]:

MUST : This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

MUST NOT : This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.

SHOULD : This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

SHOULD NOT : This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

Knowledge acquired from the stakeholders during the domain analysis dictates and helps determine the functionality and behavior of the system. Requirements and system features are both measurable and prioritized.

To make sure that the solution closely meets the functional and usability needs of the actors, traceability requirements are imposed between the different phases of the development. Traceability from *needs* to *features* to the final *product*. Figure 2.1 shows such a process.

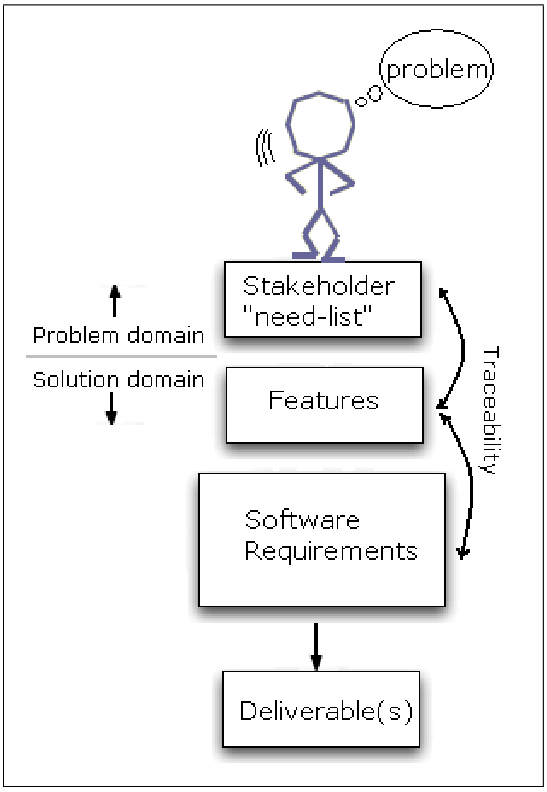


Figure 2.1: Requirement acquisition and software features

Analogous to Maslow's "Hierarchy of Needs" theory, a pyramid of user-needs is composed to asses and evaluate if the application covers these needs adequately. These *quality of service* requirements we're aiming for should have these characteristics (importance: bottom up):

- **Intuitiveness**: does it feel natural, and does not "make me think" ?,
- **Usability**: is it user-friendly? is it easy to maintain and evolve?,
- **Efficiency**: does it let me do what I need without long workarounds? appropriate to its audience?,
- **Learnability**: can I learn it quickly? is the manual good?,
- **Functionality**: does it do what I need? Sufficiently powerful to satisfy my requirements?,
- **Correctness**: does it do it correctly?,

2.1 Requirements

Functional and non-functional requirements.

2.1.1 Machine Requirements

To establish a suitable environment for the `WebAppCFPman` application, the following sections describe the minimum hardware and functional requirements expected.
To provide reliability and operational continuity under satisfactorily constant conditions, the hardware system should preferably be a Unix-like operating system. Furthermore, some, but not all, of the requirements assume server is running specific types of software, such as process scheduling mechanisms, e.g `crontab`.

2.1.1.1 Performance

a) Storage: To accommodate a considerably large collection of documents, we need a high capacity database.

b) Time: Extraction of information from the database should not cause excessively high response time.

c) Hardware: Unexpected process failure or exhaustion of system resources **should** be avoided

2.1.1.2 Dependability

The system, as a single entity, must have a combination of the following attributes to be dependable.

a) Accessability:
The `WebAppCFPman` system and it's subsystems, notably the mail-handling interface, should run at all times. Access to the system should be granted to all users. Privileges to alter and submit documents shall adequately be limited to authorized users. Administrative personnel whom have acquired access to the system shall be granted "super-user" privileges.

b) Availability:
Access to the `CFPMan` system, from the users' point of view, is done over a network, through a client that meets the requirements outlined in 4.1.2.1. The administering staff **should** have access to the system, with regards to updating and maintaining it with proper privileges. The system must perform consistently according to their design requirements and specifications. Availability implies reachability over any given network that's connected to the Internet.
Incidents such as system bugs severe enough to interrupt proper operation, crashes, and network outages can not be concordant with this attribute.

Hardware: The underlying hard **must** worthy of reliance and **must** be fault-tolerant. Unforseen mission-critical failures **must not** cause extensive downtime.
Information: The information contained in, and conveyed by the system shall be reliable and be usable with high confidence. Information-wise, reliability implies that what you put in is what comes out.

c) Usability:
As a the service-providing application, these usability criteria are deemed essential: time to learn, speed of performance, retention over time, rate of errors by users and (subjective) user satisfaction. (Visual) Appearance of the application is also of significance.

d) Reliability, credibility:
Credibility entails consistency. What goes into the system must come out having the same *uniformity*, ie. the form may change (dates)

Assessments of these attributes, as well as most of the other functional requirements, **may** be incorporated and tested through simulation at the design and prototyping stages, but **must** be verifiable on the end product through some methods: analysis, inspection, demonstration, test or review of design.

2.1.1.3 Maintenance

By machine maintenance requirements we understand a combination of requirements with respect to:

a) Corrective maintenance:
By corrective maintenance we understand servicing operation that corrects any occurring error, be an internal application or an error introduced through user-interaction. Such maintenances and calibrations may be done locally or remotely by an operating staff.

b) Preventive maintenance:
By preventive maintenance we understand such tasks as monitoring and updating of the system hardware in order to prevent system faults and failures.

Any downtime period caused by a planned outage due to maintenance services, must be minimized to an acceptable level.

2.1.1.4 Platform compatibility

Browser :
Due to inconsistent implementations of browser specifications that display web pages on the users' screens, a constraint is introduced: this system, as of Prototype #2, is not **required** to run under all browsers (see **??**)

These constraints are accordingly justified by knowledge acquired during prototyping. Knowledge that pertains to user-behavior and use of browsers, see user classes 1.1.2

To use the `CFPMan` application, users will need a computer workstation with:

- Mozilla FireFox (version 2.0+) on any platform,
- Javascript enabled,
- Cookies enabled
- Monitor set to a resolution of at least 800 pixels x 600 pixels for proper viewing.

2.1.2 Interface Requirements

By an *interface* we mean a clearly defined interaction protocol which allows external applications to interact and possibly alter the state of the system, likewise with the order reversed (mutual interactivity)

When submitting documents to the system, users must be given the tools necessary to complete the task at hand.

2.1.2.1 Validator

Having crucial relevance for the whole process, the validator must have all the requirements and implement most, but not all specifications laid out by the authorative bodies (W3C, ISO, etc.). These standards include, but not limited to, the XML standard (1.0, 1.1), namespaces in XML, W3C XML Schema (1.0).

The validator must have a combination of the following attributes:

Predictability (Replicability) :
Should an error be encountered, the system must behave consistently. A document may pass/fail validation under certain conditions, with certain inputs. An operations on that particular document, given the same inputs and conditions, must give the same result on successive trials. For an "`invalid`" document, error-reporting **must** be persistent until the error causing condition is remedied.

2.1.2.2 Mail-handling Service

In addition to the requirements listed in 2.1.1.2, this subsystem must have a combination of the following attributes:

Interoperability :
The mail-handling subsystem **must** inter-operate with other subsystems that facilitate functionalities that are vital to the system as a whole entity. These subsystems include, but not limited to, validating and database interfaces.

Maintainability :
In case of completely unexpected and/or unreasonable adverse events,

administering stuff should be able to restore the system to a required level of operation, this is also *restorability.*

To be queued for processing, the following must be true about incoming mails;

- "Subject" field contains a predefined identifier token, e.g "CFP:"
- Mail has an attachment. Document must have the extension assigned to documents with the XML MIME type: .xml or .XML

2.1.3 Security And Privacy

Two sets of customers are expected to use the system:

- Internal trusted systems and users,
- External trusted business partners

By *trusted* we understand entities that expected or more firmly, trusted, to not behave in malicious or otherwise can cause adverse events. These malicious actions may include submitting documents containing incorrect data, or deleting documents amass, intentionally or not. This can harm the dependability requirements stated earlier.

External entities may include other institutions or universities that have reached an agreement on the use of this application. For both groups, access should not granted in bona fide; identification, authentication and auditing measures are necessary. Accountability or blameworthiness is essential in this requirement.

2.1.3.1 Access Control Policy

The information contained in the system is available to the public for complete consumption, with few exceptions: sufficient access control policies should be enforced to obtain positive identification of users and, based on their membership in predefined groups, grant or withhold privileges.

2.1.3.2 Identity Management

Access to the information contained in the documents should be granted to all a case by case basis. Access should be granted through a role-based credential system in a hierarchical fashion such that:

- Super user: Has complete and unrestricted access. Though neither practical nor advisable[1], this role can be assigned to administrative members.
- Regular user: A registered user, to whom privileges to access/edit/delete own Call For Papers are granted.
- Guest: Have fewer rights than a regular user. Guests are only given viewing privileges.

2.1.3.3 Privacy

Users must acknowledge and consent to the following:

- the collection, use and disclosure of personally identifying information, such as name and email address, of the type and for the limited purposes described in the policy document.

2.1.4 Inter-Machine Dialogue Requirements

This section pertains to requirements that are specific to machine-machine interactions. This would include data transferred between the Business Logic Tier and the Data (base) Tier (i.e. different layers, but the user doesn't have a direct interaction in this context) and data transfer to and from remote machines.

2.1.4.1 Mail Service Providers

To be able to effectively bring the storage and mail-repository in sync, the following are expected.

[1]breaks the "principle of least privilege" objectives

- Mail-service provider(s) (MSP) should enable complete or partial access to a querying applications as long adequate authentication is provided by the requesting part(ies).
- When a request is made to a remote MSP, a response is expected. Response should clarify whether the preceding directives were performed correctly or not.
- Simple data communication protocol will be sufficient for the system to communicate properly with inquiring agents - both human and automated applications (harvesting).

2.1.5 Document Requirements

The construction of the XML documents should be human-legible, semantic and reasonably clear: this entails meaningful document tree structure, e.g nesting participant data under the appropriate committee that the participant belongs to.
We introduce two vital constraints for documents:

Well-formedness :
Simple notation: *If it is malformed, it is not XML.*
This attribute concerns rules that apply to all *well-formed* XML documents. A document is well-formed when it is structured according to the rules set forth in 2.1.5.1

Validity :
Rules that apply to all *valid* XML documents. A document is semantically valid when it is structured according to the rules set forth in section 2.1.5.2

Any violations of either of these constraints are deemed "fatal" errors, with a few modifications.

Violations of well-formedness rules : The application **must** promptly terminate.

Violations of the validity rules : After encountering a fatal error, the processor **may**, at user option, continue processing the document in search for more violations - an effort to minimize multiple re-validations after the user has made a corrective step in response to a previous run[2].

[2]That is, to avoid re-runs in case of multiple violations in a document.

2.1.5.1 Well-formedness constraints

With the above informal descriptions in place, we can now migrate the grammar validity constraints into a precise document model.

There is exactly one, and only one, document information item in the information set, and all other information items are accessible from the properties of the document information item either directly or indirectly through the properties of other information items.

Non-formal rules that must apply:

- An XML document must contain one root element (no more, no less). Furthermore, no whitespace characters **should** precede the XML declaration.
- A non-empty elements must have a start tag and an end tag,
- conversely, self-closing tags like XHTML's `<br>`, `<img>` , and optional elements in an XML document, e.g a Call For Paper's URL element `<url>`, must end with `/>`, that's: `<br/>`, `<img/>` and `<url/>`, respectively.
- Special chars need to be escaped when not used in their their literal form; thse include the ampersand character (`&`) and the angle brackets (`<`).
- Element names are case-sensitivity, e.g.: this does not conform: ¡name¿..¡Name¿,
- Whitespace characters in an XML element name are not allowed.
- All attribute values must be in quotes, double or single, e.g as in `<url relative="yes" rank='high'>..<url>`
- An attribute name **must not** appear more than once within an element

All modern browsers can partake in testing for these well-formedness constraints, so ideally, users are **recommended** to view the document with a web browser prior to submission.

2.1.5.2 Validity constraints

For a document to be semantically valid, som informal constraints need to be specified. The following structure is syntactically valid in an XML document but may not be semantically valid:

```
<cfp>
..
 <title>Conference title</title>
 <acronym></acronym>
..
</cfp>
```

These are some of the informal validity constraints:

- all required elements are present, e.g `acronym`,
- that the hierarchical structure of these elements are maintained.
- that these elements have the appropriate type(s),
- and that no undeclared elements have been added.

2.1.5.3 Other constraints

In addition to the syntax and semantics rules above, some other constrains apply:

- **Identifier**: The acronym is used as a unique document identifier. A document that is submitted to the system must have an acronym that must be unique within the scope, and during the lifespan of the application. By `uniqueness` we understand *single instance* of a name, only one of its kind.
- **Document transfer**: No assumptions are made regarding limitations that impact a user's ability to transfer documents to and from the system. These limitations **may** include those imposed by mail services (file attachment size), web upload (maximum upload file-size).
- **Document size**: Documents that are uploaded, posted, transmitted through mail or otherwise made available to processing in the application are required to within a reasonable limits – due to database or XML processors. The size of a document is subject to limitations that we can not foresee at the time being.

Chapter 3

Software Architecture and Design Model

In this chapter some of the main design decisions are justified. Along the way, we will introduce some design constrains that are justified by factors that lie outside - but have a direct impact on - the system design.

The primary concern is on accomplishing stated requirement goals, but secondary concerns such as auditing and logging are discussed but not necessarily implemented further in the application (subject to future extensions).

In section 3.8 a brief semi-formal description of the grammars that govern the XML documents are provided. An intermediatory knowledge about context-free grammars is assumed.

3.1 UI Design

Most modern web applications, like Google's mail application, place the User Interface engine on the client-side. Instead of having to reload the entire UI after each request action.

Regular web applications work on a synchronous model, where one web request is followed by a response that causes some action in the presentation layer, this action mostly locks down the UI, effectively blocking any further input sequences. This traditional "click and wait" behavior limits the interactivity and the usability of the application. Some applicable techniques are proposed to enable clients to exchange data asynchronously with the web server, without changing the behavior of the currently viewed page.

Other web intrinsics that were given particular attention:

Fluid/elastic layout :
With the help of cascading stylesheet (CSS), intricate page layouts can be achieved. The main objective is to design a page that retains it proportions and renders relatively the same on different screen resolutions. Statistics show that 50%+ of users navigating through a site will have a standard resolution of 1024x768 pixels. Initially, this application was made to comfort this majority[1], but has since been changed to an *elastic* design. This elasticity is achieved by assigning the layout components with percentage values, instead of fixed pixel-sizes. This approach has a minor draw back: as the page scales to fit the screen, lines can become so long that *readability* is decreased. Readability, albeit important in other contexts, is not an issue here: as few as two pages contain long passages of text, none of which are the frequently accessed pages.

Hierarchical navigation :
This allows the application to behave more intuitively,

Broken conventions :
With traditional web-pages, users are used to the traditional "click and

[1]Users with higher resolution would get an according smaller image, that can be difficult to view, as initial tests with Chris' big office screen has shown

wait" behavior where pages are reloaded at an arrival of a response from the server. Recent web-applications use techniques that break these conventions. Appropriate visual effects must be added to the page interface to provide feedback to the user, to let them know that something has happened. Some examples of these visual enhancements include:

a When a user requests deletion of a CFP document, deleting the paragraph element that contained that CFP (removed from the page).

b When a schema violating error is discovered during the submission of a document, an appropriate approach is taken to highlight the violation, e.g if the mechanism used is a web form, then the originating input field is highlighted and is auto-focused, see more in

c When user finished editing a particular CFP, intermediate graphical elements are shown to convince the user that the action has been requested. A subsequent server-response, either failure or success, is also shown on the same manner.

Simple usability tricks :
There are some simple, yet subtle and even seemingly dull/trivial tricks that are used to enhance the usability of the application. These presentational markups are conceived:

a Flexibility: instead of considering a fixed date format, some intrinsic date-formatting object is used to allow a bewildering array of date formats, even relative ones expressed in a natural language, for instance: "*tomorrow*", "*+1 week*", "*next month*" are all accepted.

b Focusing: As customary in web-applications, when the submit pages has done loading, the cursor jumps straight to the acronym field, ready for input.

c Enhanced form fields: Text entry fields are made large, and given high contrast relative to the surrounding objects. Moreover, when a user returns to a field that has already been filled out, the field's content is auto-selected since the user is most likely to a)edit the content of the field or b) copy and paste the text.

d Form hints: To improve both accessibility, usability and prevent unnecessary re-submissions caused by an invalid input, each field in the web form is coupled with a conveniently hidden away text snippets. Once a particular field receives or looses focus, this text `hint` is display or hidden, respectively. The behavior of this technique is equivalent to the native markup of the "title" attribute in HTML tags.

e Messages: When important notes and error-messages are conveyed to the user, we will do what is customary: use **verbose** methods, such bold fonts combined with bright backgrounds.

f Labels: XHTML label elements increase the focus area of an input field. To associate a form inputs with corresponding legends, labels are extensively used on the submission page. Furthermore, screen reader users can, in theory, read forms that have labels.[2]

Inline processing is used to validate form inputs. Prepared pairs of variable names and values are then sent to application logic, where the input is transformed into XML and validated against the schema. The above usability enhancements can be seen in appendix B.4,p. 121

To author a new CFP document, a template XML document is provided through the web-interface, but users may also export an existing conference data as an XML and edit it to fit their purposes.

3.2 Application Design

Implied constraints and requirements dictate our choice of application design and architecture. Some of these design constraints imposed on the implementation concern the choice of language(s), interface design, and use of external services.

3.2.0.4 Mail-interface

3.2.0.5 Harvesting Emails

To avoid redundant, unnecessary processing and maintain consistency, some attention is given to a proper labeling of incoming emails. Not all incoming mails are marked for processing. These are the rules that applied when ascertaining whether a message should be queued for harvesting:

Subject :
A *magic* prefix is used as an identifier to mark relevant messages. This prefix is configurable and solely used within Gmail. As of prototype #2, this prefix is "**CFP:**"

[2]Though the submission page itself is completely unusable for screen reader users, rest of the application behaves accordingly. (Some, but not all pages tested with Lynx)

Attachment :
Message must contain an attachment file that satisfies the following;

a) Format: the extension of the attachment document must be either of ".xml", "XML",

b) Clear text: document should not be compressed.

Unlike centralized systems where events are bubbled, ie. events are transparent, subsystem A can execute subsystem B, which then - on a given condition, can execute subsystem C. In a distributed system, events do not propagate. A subsystem does not have an inherent ability to decide whether an event has occurred on a remote subsystem. The harvesting module needs to be scheduled as a recurring task.

Crontab :
`crontab` command, found in most Unix and Unix-like operating systems can be used as a daemon to schedule the task of mail-harvesting as often as we see fit. These pre-determined regular time intervals should be chosen with care, so as not to stress the remote system. A crontab entry that sets up a cronjob that is run at every half an hour may look similar to this:

```
0,30 * * * *  php ~/apps/cfp/mail.daemon/run.php > /dev/null
     2>> ~/(...)/cronMailErrors.log
```

Any errors outside the scope of the script are piped to a log-file, this might be sufficient at the time being, but some corrective steps should be added as mitigation at a later point.

3.2.0.6 Security in design

As an external entity, Gmail enforces authentication only through industry standard security measures. Messages are passed though Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL).

Internal Authentication Without creating creating the impression of bullet-proof secure system, adequate care should be taken to avoid unauthorized use of the assets in the system, particularly through such widely deployed, third-party interface as Gmail. Through authorization we verify that the parties are who they say they are.

Blind Credential

Status quo:
It is quite difficult to establish the true identity of a sender.Currently, we have no adequate means to authenticate either the message or the sender, other than relying on the *from*-address specified in the email-headers appended to the message.

Alternatively, some high security schemes like identity certificates or a digital signatures to provide authentication of messages/senders. By accepting secondary attachments of the above type(s), this would eliminate unauthorized access, albeit being arbitrarily complicated to implement and maintain in the scope of the version.

By an authorized mail address, we understand an address that is sanctioned or given permission to access the system. Addresses that belong to registered users are deemed "authorized".

A third party that has gained access to an authorized mail address may try to incorporate possibly destructive and/or inaccurate information into the system. In cases where it's difficult or impractical to "authenticate" a user, or there may be a valid reason to commission impersonation, i.e allow a user to act on behalf of another (trusted third party), we may need to enforce authorization through blind credentials: Identity can be established if a document qualifies under these criteria:

- Direct: document is submitted through an authorized mail address,
- or by proxy: the author has access to that mail, and can subsequently verify his authority on request.

As such, an authorized mail address, by virtue of its possession by a user, is used as a Blind Credential to enforce a rather weak authorization. For an improved approach, see the extension discussed in 4.5.1.3

Persistent Connection : Since the database is a remote resource, a persistent connection is used as to avoid creating multiple connections, a connection is opened once and kept in pool for the application's entire lifespan (see 3.3.2.3)

3.2.1 Data Visualization

Conveying information in an intuitive and meaningful manner is a key concepts in interaction design and visual design. Through a visually enriched UI, the presentation of the data is made familiar to the user by means of object visualization.

Information coherence is an integral part of the user-interface. Coherence is attained by providing multiple interpretations of the same data, such as "events in time": Figures 3.1 and 3.2 show two subtly different ways of representing the same data. The functionality of the hyperlinks on both figures are coupled with a timeline that displays corresponding data upon a user-click.

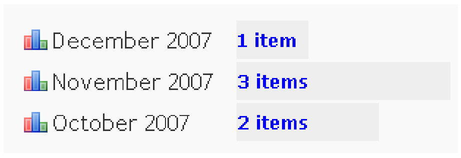

December 2007 1 item
November 2007 3 items
October 2007 2 items

Figure 3.1: Visualization: Aggregating events

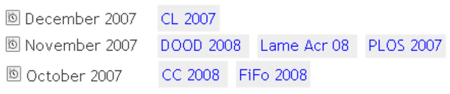

December 2007 CL 2007
November 2007 DOOD 2008 Lame Acr 08 PLOS 2007
October 2007 CC 2008 FiFo 2008

Figure 3.2: Visualization: clustering events

Particularly important event-dates, e.g deadlines, are spotlighted using density functions to calculate time proximity. Likewise keywords specified across conferences are made into click-able bundles that are *weighted* in their order of frequency.

Appendix B.8 and B.7,p. 124 show how time proximity (deadlines) and keyword densities are visualized.

3.2.1.1 Timeline

A `timeline` is a useful widget that can display linear representation of the events over time. With information denoted at key points, users can pan this chronological display horizontally where multiple events come to the view.

The SIMILE Timeline contains three, possibly more, "bands" that can be panned/dragged independently and whose event-bindings are synchronized: the topmost band, which usually has the least temporal unit, pans the next topmost band, but with a slower speed. Descending units of time have descending panning speed. Moreover this stacking provides increasing wider window of time.

3.2.1.2 Multiple Timeline Instrances

Most of the time, users are interested in viewing multiple dates at once. The Timeline API does not have intrinsic ability to render multiple timeline instances at the same time. This is an issue that is a bit tricky to negotiate.

One is to bridge the user-request: determine which type of date is requested for a CFP, then fetch that information from the database and push it onto the XML source document, the API is then (re)initiated with that source, templates reset, and the desired type of events will appear on the timeline without the page being reloaded.

A variant of this method is (per request) applied to the actual design. Important dates are clustered into `deadlines` and `other` types, where the user can switch between viewing either of these two sets. This empowers presents to *watch* events of several conferences at the same time.

Appendix B.17, p. 129, shows a timeline with conference dates group by months. Hyperlinks are cause the timeline to pan/jump to the appropriate event date. This unique coupling between the application's intuitive, point-and-click interface and the timeline API helps users observe and track important dates in an interactive and meaningful way.

3.2.2 Geocoding

Geocoding is the process of defining the position of geographical objects relative to the standard reference grid. Latitude and longitude geographic identifiers are used to map these geographical Locations as datapoints on a map. Users can click on these datapoints, which are numbered in descending order of event dates, and acquire more information about the conference or navigate to it's page.

Appendix B.6,p. 123 shows geographical locations alongside textual annotations about the events. The map is fully interactive: zooming, panning and event-annotation can be toggled (on/off).

3.2.2.1 Visually-arranged Clouds

This setting is used on the weighted lists: keywords, important dates, etc. The font-size is used to depict either frequency or time proximity. Visual enrichment is used to spotlight significant keywords or submission deadlines that are of interest to the user.

keyword density :
This is a function of density of the words used as keywords within a conference. More commonly used keywords are displayed with a larger font for stronger emphasis. Can be seen as visually-arranged cloud of words. Appendix B.7,p. 124 shows a weighted list of keywords. Once the user clicks on a keyword, similar CFPs that share that characteristics are revealed.

Temporal density :
This expresses the time proximity of submission deadlines. The acronyms for upcoming conferences are dimensioned as function of "nearness in time". The nearer the deadline, the bigger the acronym. See appendix B.8,p. 124

3.2.2.2 Content Recommendation

Content Recommendation can be described as a function of density of interconnections: If a conference is connected to a `site A`, and `site B` has some shared characteristics with or is linked from `site A`, then `site B` will most likely be relevant for the user. This approach usually reveals the network of connectivity between websites that are elated in one way or the other. In particular, organizers usually link to sites related to the conference: guidelines, submission page, joint groups, or even pages related to the conference venue.

Using the URL of the conference as a point of departure, links to relevant documents are displayed. Some of these relations are contextual, e.g the content recommended for a conference called "*International Conference on Compiler Construction*, CC 2008" contains a hyperlink to a document called "*Compiler Construction using Flex and Bison*", this relation is purely contextual. More useful are documents suggested through association: atop the result list are two pages that are closely related to this conference - a conference called "*Tools And Algorithms For The Construction And Analysis Of Systems*" and a page called "*ETAPS 2007*", both of which belong to the organizers of the "CC 2008" conference.

An example of a recommended content can be seen in appendix B.10,p. 126

3.3 Application Architecture

To ease the task of updating and extending the application, the number of modules have been intentionally kept low. These are sub-packages within the application: *modules*,*xml*,*mail-service*, *config* and *display*:

- **modules**: These modules include some components:
  - **Validator** The subsystem responsible for ensuring documents' adherence to schema.
  - `Transformer`, part of the system that transforms XML documents conforming to the schema into readily executable SQL statements.
  - `SQL Builder`, merges these SQL statements into model that enforces some constrains, e.g by inserting relation variables, such primary/-foregin keys, into other statements that reference that row.
  - `XML Database Views`, these modules generate some specially formatted XML documents that are used in combination with the external APIs.
- **xml**: This package contains all the classes that manipulate and process requests to and from web-interface. Files in this package have the prefixes `XML` and `XSL`.
- **core**: Here, the indispensable application logic classes and subclasses are kept. Package classes inhibit modularity and encapsulation: content classes separated from presentation and data-processing (model) classes (MVC pattern).
- **mail-service**: Using Gmail as mail repository, this subsystem handles submissions that are sent to the system. Mail processing: in addition to schema validation, submodules also check whether submitted documents meet certain criteria, like correct document format.
- **config**: This package contains configuration files that contain both sensitive and insensitive configuration variables. The sensitive data, such as database username and password are kept off stage in a document. An XML document is used to hold other config variables in clear text.
- **display**: The application's user interface which contains means to both distribute and ingest documents. Has all the submodules needed for presentational purposes: stylesheets(CSS/XSLT), behavior (Javascript) and template documents.

The architecture of the important document-handling components is shown in figure 3.3.

3.3.1 The LAMP Stack

The application makes use of `LAMP` stacks combined with a remote email-service provider. The `LAMP` refers to a combination of open-source technologies that have gained popularity during recent years and are used in most web-oriented software development projects. With `LAMP`, ubiquity is essential.
These stacks are **L**inux/FreeBSD, **A**pache, **M**ySQL, **P**HP/**P**ython/**P**erl.

3.3.1.1 PHP5

I have chosen PHP, mostly because of familiarity, ubiquitousness and widespread support across server platforms, and not least for the bewildering number of extensions that can be compiled with it. PHP comes pre-compiled with fair amount of default packages such as DOM/SAX/XSLT, etc. The recent versions being more so.

3.3.1.2 MySQL 5

Due to it's performance and widespread adoption, MySQL is ... A major milestone was reached when version 5 of this popular SQL database management system was released, supporting features that were only available to users of proprietary databases (Oracle, SQL Server Express, DB2 Express, among other enterprise RDMBS products). These capabilities included stored procedures, triggers, server-side cursors, views and integrity constraints.

3.3.2 Separation of Concerns (SoC) and MVC

To achieve maintainability and adaptability and cope with the complexity of the application, in it's current version and in future extensions, divide and conquer strategy is used to separate the user interface from the business logic of the dynamic web application.

3.3.2.1 Application Logic Tier

Objects in this layer (also known as *business logic*) are responsible for performing the required data processing, making logical decisions and evaluation.

Document processing in response to requests made via different interfaces (web-form, mail, etc) is put in effect in this layer.

The business logic comprises two components in the MVC paradigm: the model and the controller component.

Model :
The model is an single-class representing data or activities that pertain to the data(base). This component is solely used to handle a *model* that emulates a CFP, with all the basic ABCD database functions: add, browse, change, delete. All communication with the physical database is routed through this component;

Controller :
This component is composed of several classes that are closely related. User interface components:

Front Controller :
Contains simple controller responsible for handling server requests and passing back server results to other parts of the user interface. This subcomponent is only used in combination with asynchronous calls made through the browser. A simple access-policy mechanism is inserted at the top of the class to avoid unauthorized and/or direct requests made through http GET requests (*file.php?var=val*).

Page Controller :
This is the interface's control module, which channels URL requests, trapping all user actions and passing them on to the appropriate sub modules within the interface. A little re-writing trick is used to transform URL requests similar to `/?cfp.id.12`, `/?cfp.keyword.compiler` and `/?toolset.schema` into an object, where `cfp` and `toolset` are page identifiers, `keyword` and `id` are keys, and `compiler` , `12` are these keys' respective values.

The class diagram 3.4 shows the structure of the object-oriented server-side model: the object classes (super- and subclasses), their internal structure, and the relationships in which they participate. A node corresponds to a single class, and edges corresponded to links between them. Lines with hollow triangles denote inheritance (*is a*) relationship between classes, while dotted lines are a dependency (*uses*) relationships (aggregate classes that require inclusion of other classes at runtime). Exception classes are left out for brevity. Statically accessed classes are bundled at the bottom of the diagram.

Remote mail server :
When a user submits a document through email, a notification of arrival is sent with little or no delay. An auto-responder dispatches an automated reply to incoming messages. For persistent users who keep sending submissions (justification: either the harvesting system is down or this is due to impatience on the user's part), this automated reply will be sent at most once every 4 days. Some pre-determined time elapses, then the harvesting system is invoked and starts the syndication process.

Three labels are used to categorize emails at the mail-repository: All incoming email are labeled "Pending" till processed, successfully processed mails are labeled "Harvested", while erroneous documents that failed the validation are labeled "Error".

Messages that did not match the requirements outlined in 2.1.2.2 are skipped. During the subsequent processing, two events can be fired:

OnError :
Caused by two error types: document errors and internal errors.

Internals errors: Resource exhaustion (e.g dead database links or validation process being blocked), capacity threshold (e.g too large document), network failure and abnormal process termination. None of these are exposed to the user, instead:

- the user is notified and advised to await further instructions before submitting anew,
- A description of the error is send to an administering staff, whose email address is specified in the config file,
- Emails that have been fetched but not yet processed are put back into the queue: marked **unread**, and labeled "Pending". The application is then terminated.

document errors: The document submitted does not conform to the schema grammar. The user is notified and provided with a link to a page that contains detailed descriptions, that we choose to call *stack trace*. A short description of how this *stace trace* is implemented is found in section 4.3.2.2 of the implementation chapter.

onSuccess :
The message is labeled accordingly, and the user is given a link to leads to

the submitted documents page in the directory. At request, administering personnel can also be notified.

In the later case, processed emails can be moved to the archive, an option that can be disabled/enabled with a simple on/off switch provided through the configuration file.

A third logical error may also be expected to occur: requirement 2.1.5.3 implies that documents need to have a unique identifier and can not be submitted more than once: If this is the case, the processing of that particular document is halted, and the user notified. This requirement is enforced in other interfaces as well. This could be extended in future versions of this application: users might be asked if they wish to overwrite existing documents, given that they possess enough credentials to do so.

3.3.2.2 Presentation Tier

The top-most level of the application is the user-interface, whose function is to translate user-requests and server responses into an accordingly ordered flow: accepting input from the user, and processing instructions to render the interface itself.

A view is some form of visualization of the state of the model described above. As seen on figure 3.4, this component has subset of different views that each express the state of the model in a unique way: geographical locations of conference venues (map), timeline of upcoming events or a simple flat CFP listing. Through the model, data fetched from the database is processed here and formatted accordingly before being sent to the templating engine. This simple templating system enforces SoC as well: coders code, developer design. A certain level of logic (ie. php code) is required within the XHTML code.

Call For Papers are edited in a unique way: users do not submit forms back and forth, users click on the paragraph that is needed : the content of that paragraph is duplicated into a multi-line text area. When satisfied, user presses a ‘save’ button that saves the data and the *edit* session end, all of these event sequences happen without the (visual) state of the page being changed. This feature may not be apparent to first-time users, as web-forms are more intuitive to users: some familiarization is necessary.

A screen of this process is shown in appendix **??**, p. **??**. Some implementation details about this *inline editing* mechanism are found in section 4.3.3

3.3.2.3 Data(base) Tier

The model component is strongly coupled to the database object to execute the four aforementioned basic functions of persistent storage. Apart from error-handling, the database object is merely a channel with few functionalities that enable packing/unpacking of data, fetching/storing query resources and the execution of queries on these resources.

Persistent Connections Instead of maintaining a connection for each database request, persistent connections are used.

> **Language-specific behavior**: Before establishing a new connection, PHP will try to find a (persistent) link that is already opened with credentials similar to the one provided; If one is found, an identifier for it will be returned instead of opening a new connection, other wise a new connection is established.
>
> With a regular connection to a MySQL server, users are advised, but not required, to close connections after as to free resources. Not required, due to the Java-like garbage collector that reclaims resources no longer in use. However, Persistent database connections are affected neither by this reclaim nor an explicit call to the "close" function.

- **Good**: reduces latency on connect/disconnect/reconnect operations
- **Good**: almost all page requests of this application needs to fetch data from the database, justifying the use persistent connections.
- **Good**: Most LAMP stacks(3.3.1) rely on a distributed architecture: database and application on separate hosts/servers.
- **Good**: No connection and authentication overhead. The later is performed only once.
- **Bad**: If you host multiple applications that use persistent connections on multiples databases[3], and MySQL's connection limit is set low, you could be in trouble.

[3]MySQL, usually, imposes limitations on how many connections a user may have.

To be or not to be persistent Performance benefits from persistent connections are minimal at best, but efficiency is increased considerably. Persistent connections have a one-to-one mapping to regular connections in PHP. Rolling back to regular connections is easy as removing one character from a function call: *msql_pconnect* vs *msql_connect*. This may (and probably will) change the efficiency of the application, but not its behavior.

3.4 Role-based Access Control

Each CFP object is paired with a list of access permissions that is granted to user groups considered within the scope of this application:

Access control and credential validation mechanisms currently implemented are very simple, but can be extended with group privileges at the application tier.

3.4.1 Access Policy

Through group affiliation, users have levels of access for specific CFP documents: no access, read-only, read-write, and submit-data (mail specific). This is a rudimentary mechanism that effectively restricts access to specific functions within an application. These groups, on which the security clearances are given, could also be extended at a later point. Figure illustrates how the role-based access control and sessions are used to to group application functionalities.

3.5 Database Design

Unless otherwise noted, all database mentions in this document refer to MySQL (version 5 or later). Despite all queries being SQL, there's a heavy use of aggregate and sorting functions that may only be obtainable in MySQL's SQL dialect.

A conference has different committees whose members have different roles: program chairs, steering committees, and so on. The relation of these members to a conference can be modeled in two ways:

- Static list: the list of the members involved in conference are kept static, that is, each conference has it's own, regardless of how few/many members they may have in common.
- Users are registered without being appointed to a conference. Only then are users coupled with a certain conference. Though not strictly necessary, this would enable reuse of member-lists across conferences, and many-to-many relationships between conferences and members in the list. Furthermore, these members may have varying roles with regard to conferences they are participating in. To eliminate this redundancy, roles are not static, but are closely bound to conferences.

3.5.0.1 Preserving Referential Integrity in MySQL

To get quick view of the database tables and the inter-table relations that exist, do take a look at appendix B.18.

There is a need for some logic at the database layer: all tables but one have a key inherited from the primary key of referenced table(*CFPEntryPoint*). This foreign key constraint enforces integrity: we are ensured that no *stray* entry can exist without a corresponding entry in the referenced table.

```
ALTER TABLE `CFPImportantdates`
  ADD CONSTRAINT `CallForPapers_ibfk_3` FOREIGN KEY (`cfpID`)
      REFERENCES `CFPEntryPoint` (`cfpID`) ON DELETE CASCADE;
```

Cascading deletion: If a parent row is deleted, its associated (child) records are deleted as well - otherwise the existence of an associated record would not allow the DELETE operation of the referenced record.

This means uniformity within the application: delete an entry in the referenced table *CFPEntryPoint*, and all the referencing entries are deleted from child tables (dates, keywords, details, header). Data integrity is maintained.

3.5.0.2 Database Engines

MySQL supports two database engines: `innoDB` and `MyISAM`. The later supports many advanced features such as *full-text indexes* but misses one all-important feature: foreign keys. No foreign keys, not enforce referential integrity.

Natural Language Search : Full-text search supports some of the features in Natural Language Processing, particularly relevant in this scope, Query Expansion (QE) processing is supported. QE provides *fuzzy* searches:

```
SELECT * FROM CFPHeader
 WHERE MATCH (title,acronym)
 AGAINST ('Compiler' WITH QUERY EXPANSION);
```

First, search is done using *Compiler*, then words from rows with the highest *relevance* score are added to the search term, and the search is performed again. If a conference containing '*Compiler*' and *Construction* is found on the first pass, the second search concatenates the result with CFPs that contain *Construction* in either of the fields. This fuzzy search could be used to find related conferences that may not necessarily have the same keywords or category. This fuzziness becomes even more powerful if one or more of the search phrases is spelled wrong; the second pass would match.

In addition to the natural language search mode, the boolean mode is equally useful:

```
  mysql> SELECT * FROM CFPHeader
    -> WHERE MATCH (title,topics)
    -> AGAINST ('+language -compiler' IN BOOLEAN MODE');
```

Since referential integrity is essential, innoDB, which lacks all of these goodness is chosen. If integrity could be achieved through other means, innoDB can be happily abandoned. A possible extension to this module would abandon innoDB and consider the use of conditional execution and pre- and post-INSERT database `triggers` to synchronize tables:

```
mysql> CREATE TRIGGER foo
AFTER INSERT ON t
 FOR EACH ROW
  BEGIN
  INSERT INTO s SET column = NEW.column;
    ..
END;
```

This trigger would be invoked on `s` *after* an INSERT on `t`, and rows are sync'ed.

3.5.0.3 Dealing with duplicate keys in (My)SQL

Fault-tolerance is a key property when dealing with (semi)automated systems that have input mechanisms user interact with. In the event of the failure, some parts of the application must continue operating properly without halting the application as a whole. Unlike the rules that are enforces by the schema, unexpected minor errors should be put up with: when processing documents malformed dates or duplicate user names (already registered) are tolerable.

If duplicate keys are observed during a query, the database halts that particular operation, and the application possibly terminates. To introduce some Fault-tolerance into the document processing, errors raised by duplicate keys can be handled in various ways. In particular, three approaches with varying degrees of are considered:

INSERT IGNORE.. :
This simply ignores the INSERT operation should the unique key already exist in the database.

INSERT INTO .. ON DUPLICATE.. :
This approach is a bit more ¡¡reactive¿¿ and involves a second statement that is executed in the event of a duplicate. Assuming we created a unique index on `acronym`: INSERT INTO .. ON DUPLICATE KEY UPDATE blah='blah'. renders the `insert` neutral (this clause is supported in versions later than MySQL 4.1)

Internal error codes :
This approach involves use of a `try/catch` with the internal error-codes returned by the database to remedy any adverse causes. The naming of these error-codes vary, in MySQL duplicate entries raise error-code with the identifier "1062". At the database layer, this number is used to throw an appropriate exception called "*DuplicateEntryException*", that is caught sufficiently high in the class-hierarchy.

The last method is used to handle duplicates in two particular cases:

- Duplicate acronym: to comply with the constraints outlined in 2.1.5.3, duplicate acronyms are not be accepted. The *DuplicateEntryException* is simply re-thrown.

- Duplicate user name: This is an acceptable non-fatal scenario. We have a username but no user ID. As such, we are interested in retrieving the unique key (user ID) with which the INSERT statement had a conflict. In SQL pseudo-syntax, this:

  ```
  IF EXISTS
     (SELECT personID FROM CFPPersona WHERE firstname='Alice' AND
         lastname='Bob')
  ELSE
   (INSERT INTO tb(firstname, lastname) VALUES ('Alice', 'Bob');
  ```

  In this context, we want it to return *personID* if the record already exists, otherwise insert into database. Alas, due to an architectural limitation in SQL (rather than MySQL), this twofold read/write[4] operation is not supported. To deal with this, the INSERT is parsed with regular expressions to extract the user name.

3.5.0.4 Aggregating in MySQL

This section discusses some details dealing with visualizing nested repeating groups in MySQL. By ‘aggregating' we mean condensing a set of tuples to form a cluster; a single value.

Assume that an interface requirement stipulates that the resultset be a list of months, and each month in the list contains a list of the conferences whose deadline is in that month. First shot:

```
       SELECT acronym, DATE_FORMAT( FROM_UNIXTIME( `submisionOfPaper` )
             , '%M %Y' ) AS temporal
       FROM CFPImportantdates AS d, CFPEntryPoint AS e
       WHERE d.cfpID = e.cfpID
       ORDER BY submisionOfPaper ASC
```

Keep in mind that dates are modeled as a Unix or POSIX time, which enables sorting and ad-hoc date-formatting across all layers of the application. The above query yields:

```
+---------------+-------------+
| temporal      | acronym     |
+---------------+-------------+
| October 2007  | FiFo 2008   |
| October 2007  | PLOS 2007   |
| October 2007  | Lame Acr 08 |
```

[4] I suspect that SELECT returns a set of tuples, while INSERT returns an integer (number of affected rows), hence a conflict

```
| November 2007 | DOOD 2008    |
| November 2007 | CC 2008      |
| December 2007 | CL 2007      |
+---------------+-------------+
6 rows in set (0.01 sec)
```

That covers a lot of ground, but it is repetitious and contains unnecessary tuples. We were expecting 'November 2007' alongside '*CC 2008*', '*DOOD 2008*'.

Until this project I resolved this issue by creating a temporary variable (on the application layer) containing the value of the last element, and then comparing with the value of the next element to establish a grouping. Then I discovered a rather remarkably powerful MySQL aggregate function: GROUP_CONCAT;

With this function, we can produce a delimiter-separated, string concatenation of the non-NULL values of the 'acronym' column. Unfortunately, we can't group the resultset of this function with aliases - in this case 'temporal' is an alias - so we have to write a subquery with nested SELECTs and derived table with alias (`A`):

```
SELECT temporal, GROUP_CONCAT( CAST( acronym AS CHAR ) ) AS info
FROM (
  SELECT acronym, DATE_FORMAT( FROM_UNIXTIME( `eventStart` ) , '%M %Y'
      ) AS temporal
  FROM CFPImportantdates AS d, CFPEntryPoint AS e
  WHERE d.cfpID = e.cfpID
  ORDER BY temporal DESC
) AS A
GROUP BY temporal
```

This would yield the desired result. Moreover, we are particularly interested in retrieving additional columns; the ID is necessary for building the link that users must click on to pan to the event, and the deadline date is used to mark the position of the event on the timeline, thus we construct

```
SELECT temporal, GROUP_CONCAT( CAST( info AS CHAR )
ORDER BY info ASC
SEPARATOR '\n' ) AS info
FROM (
    SELECT CONCAT( acronym, '\n\t', e.cfpID, '#', submisionOfPaper )
         AS info,
        DATE_FORMAT( FROM_UNIXTIME( `submisionOfPaper` ) , '%M %Y' )
             AS temporal
    FROM CFPImportantdates AS d, CFPEntryPoint AS e
    WHERE d.cfpID = e.cfpID
    ORDER BY submisionOfPaper ASC
) AS A
GROUP BY temporal
```

```
+---------------+-----------------+
| temporal      | acronym         |
+---------------+-----------------+
| December 2007 | CL 2007        |
|               |   12#1197327600 |
| November 2007 | CC 2008         |
|               |   14#1206745200 |
|               | DOOD 2008       |
|               |   26#1196290800 |
             ........
+---------------+-----------------+
```

which builds a repeating group for each month. An appropriate delimiter is introduced separate the fields. The GROUP_CONCAT function is applied to the required columns. For each tuple, there maybe multiple fields, and CONCAT concatenates their values, separating the individual names by a new-line and shebang character.

3.6 XML As a Rendering Tool

In this section, we will discuss methods suitable for converting the information residing in the XML document to a SQL representation: a process we denote as ”transformation”.

Overview:
In this and the next section an array of languages and interfaces associated with XML are address:

Structure, data typing :

- DTD
- XML Schema

Presentation technologies :

- XHTML
- XSL
- XSL-FO

Document manipulation technologies :

- XSLT
- XPath
- XQuery

Application manipulation technologies :

- DOM
- SAX

The greatest asset XML has is perhaps portability and cross-application: after retrieving a well-formed document, the ASCII file is deploy-able in all sorts of applications, even ones that are not necessarily database-oriented.

We are going to examine in detail three methods of transforming a user-submitted xml-document into some executable SQL queries, two of which involve client/server-side XML processing. Since the validation of the document (logic, syntax and structure) is done in advance, we assume inherent validity here.

- **Native XML**: Storing and querying the XML document, using native XML Functions within MySQL.
- **XML Programming Interfaces**: Using an application component from the client/server to process the XML tree and produce SQL statements therefrom.
- **SQL with XSLT**: Transforming the XML document into some SQL blocks before execution.

A comparative and methodical description of these approaches follows:

3.6.1 Native XML Databases

To ease the management of large repositories of XML data, both databases designed for persistent storage and retrieval of XML documents, and relational database management systems (RDBMS) with enhanced XML support have emerged.

The two formats of XML databases are known as Native XML Databases(NXD) and XML Enabled Databases (XEDB), respectively.

NXD's have XML-based internal model and their fundamental unit of storage is XML, retaining their tree structure, whereas data is mapped and transformed between XML format and the underlying database structure in XML-enabled databases. With an NXD, XML data can be searched with XPath, transformed with XSLT, presented with CSS, and created with DOM.

Recent versions of the MySQL RDBMS has undergone a major architectural change and can now be described as XEDB, since it has a bootstrap XML mapping layer, where data manipulation is done via specific technologies (XPath) and SQL.

Note: The use of a native XML databases (NXD) – like Apache Xindice or eXist – would be justified in this project, but my impression of the notion "in conjunction with a database", as the project description declares it, has somehow defeated this purpose: thus using NXD alone is not considered. Besides, this would force us to model the data hierarchically instead of relationally, which would, intrinsically, restrict the implementation of some of the functionalities intended to satisfy stated or implied needs of this project. See limitations below.

3.6.1.1 XML And Native Functions

MySQL (as of ver. 5.1) provides some XML functions that enable querying XML documents, or fragments thereof (stored in the database) to locate specific elements and attributes using the XPath expression notation. Most relevant of these functions is the 'ExtractValue' function:

ExtractValue(xml, expr) : Given a fragment of XML markup *xml* and an XPath expression *expr*, this function returns the content of the first text node which is a child of the element(s) matched by the XPath expression *expr*.

Assume that we have invoked an INSERT statement, and the database table `T` now contains a document fragment with an internal hierarchical structure similar to "`<cfp><header><title>Some title</title></header></cfp>`", where cfp is the root element. The following SELECT statement returns a result set of CFP titles:

```
SELECT ExtractValue(T.content,
'//header/title') AS title FROM T
```

The use of MySQL as XEDB has some advantages:

Advantages :

- As we intend to use XML as exchange format, there is a need to extracted data from database and put into XML document template and vice-versa. If the database contains the XML tree, this conversion is superfluous, in terms of processing and conversion costs, this is quite suitable.

– Data need not be tightly coupled with a certain database schema. We don't need to adopt to a certain database design or establish relationships between fields and tables. We can inject the XML document - templated so that it conforms to the rules in the schema- directly into the relational model without further processing.

nd disadvantages :

– A major disadvantage is concerning the overhead of having to store the whole document tree in the database, or, if we choose to migrate all the data onto the database, having to create nested queries like this:

```
INSERT INTO table (acronym,title,..)
 VALUES (
    SELECT ExtractValue(T.content, '//header/acronym'),
           ExtractValue(T.content, '//header/title'),
    ..
  )
)
```

– The relational model is a better way to model data than the hierarchical model, unless we are dealing with native XML databases: cross-document indexes – stated as data integrity requirements – are not feasible. Also, suppose that that a member **M** has multiple roles in a conference, then all the data pertaining to **M** would have to be repeated or verbose elements such as reference IDs would have to be introduced, making the schema overly verbose than necessary.

– Not all XPath expressions and functions are currently supported, notably: relative locator expressions and variable assignment are not there yet.

– Furthermore, querying is subject to XPath's inherent limitations - compared to SQL - which, to the knowledge of the author, include lacking support for grouping, aggregating, sorting, and cross document joins. Consequently, some vendors has added more advanced XML query engines to their NXD, such as XQuery or XPath 2.0.

XQuery remedies many of these deficiencies through it's SQL-like query constructs, called FLWR[5] which is composed of FOR, LET, WHERE, ORDER BY, RETURN:

- *FOR*: an iteration construct that binds a variable to a sequence of tuples (fetching/looping),

[5]pronounced "*flower*"

- *LET*: binds a sequence to an intermediate variable (assignment),
- *WHERE*: filters the tuples by applying some predicate(s) to eliminate some of the nodes (conditional processing),
- *ORDER BY*: sorts the tuples with a given function/expression.
- *RETURN*: this final clause returns sequence of nodes or primitive values that have matched the preceding predicates.

Fig 3.6 shows a simplified syntax diagram for XQuery's FLWR expressions. XQuery has many traits that are common to SQL such as arithmetic and logical operators, collection operators UNION, INTERSECT and EXCEPT. Aggregation, sorting, user-defined functions (analogous to stored procedures) are also supported. In general, FLWOR expressions are analogous to the SELECT-FROM-HAVING-WHERE statements of SQL:

```
FOR $b IN //cfp
WHERE SOME $p IN $b//acronym SATISFIES
    contains($p, "compiler")
    AND contains($p, "language")
RETURN $b/info/title
```

In the current design, when we wish to export a certain document as XML, we have to construct a document that is valid in accordance with the schema, THEN, retrieve the information from database and push them onto that template. If this method described in section 3.6.1.1 is applied, this step would be unnecessary, since a database dump of that particular document would suffice.

Despite the aforementioned limitations of this method, it can be adequate for general-purpose XML operations on non-mission-critical MySQL applications. Once a wider deployment of the XQuery data model into the NXD realm is reached, and there's enough developer familiarity, NXD could well evolve into the next "big-thing" in areas where the relational is not suitable, i.e domain models that are not easily normalizable, [6].

Rather than using a NXD or XEDB, the next sections describe two different ideas that imply use of some processing mechanisms that can be performed on XML documents to transform them into some ad hoc SQL statements.

[6]More on normalization in **??**

3.6.2 XML Programming Interfaces

For handling XML documents, applications use either the tree model or the event model. Two major APIs are available for traversing and manipulating XML Documents:

- SAX: Simple API for XML - an event-based model. XML parsers raise events for start or end tags.
- DOM: Document Object Model - a tree-based model. XML parsers create an in-memory tree. Application programs access that tree through the DOM API.

3.6.2.1 SAX

SAX is an event-based interface for processing XML documents. It simply uses a design pattern to notify client applications through handler call backs (in Java/PHP: startElement, endElement, getEntity, etc.) when it encounters start-tags, end-tags, text nodes, etc. The client is responsible for creating it's own custom object model. As the parsed document elements are lacking orderly continuity, we have to check element names inside these call back routines, making the SAX require more code than the DOM interface.

3.6.2.2 DOM

Document Object Model parser uses a tree-based approach and creates an object-oriented hierarchical representation of the document. Predating even the XML language, DOM was initially conceived as proprietary models for rendering HTML pages on Web browsers, until the W3C began developing it in the mid-1990s. Compared to SAX, a simple cost-benefit analysis (plus/minus):

- Builds an internal tree (the original model is discarded at parse-time)

- Gready: Significantly large documents[7], require relatively more system resources.

+ Backtracking and navigation allowed (statements similar to nextSibling/parentNode are o.k)

[7]"*Large*" meaning documents several megabytes in size, or containing more than 1,000 nodes, according to a survey made by IBM/O'Reilly [McL03]

+ Implemented in nearly all programming languages (being an early W3C recommendation)

Since DOM is platform- and language-neutral interface, we will be using it throughout the project to parse and traverse XML and other SGML-derived documents like (X)HTML.

Limitation :
Before document processing with DOM parser is initiated, the entire XML tree must be in memory. Thus, in cases where the document is streamed or is a log file that needs monitoring, or in extreme cases where the document does not fit into the available memory, SAX is the right tool for the job.

3.6.3 SQL with XSLT

This approach concerns the use of XSLT meta-stylesheet as a simple SQL code generation tool: through XSLT, we can define a set of XPath filters that can transform the document into SQL.

What makes XML a powerful language is it's well-defined yet extensible structure, with emphasis on the "extensibility". As a subset of XSL[8], XSLT is a declarative language – written using an XML grammar– whose internal logic is based on pattern-matching and templates that is capable of processing nodes of an XML document into another, usually XML-based, format. Although not a general-purpose language, XSLT can - without substantial tweaking - be used to serve non-XML data, like a SQL document. A simplified model of the transformation process is shown on fig 3.7.

XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another (XML) document that uses the formatting vocabulary.

But since we are transforming a *source* XML tree into a *result* tree, we can use the XSL vocabulary to diverge the resulting tree into any format we see fit.

[8]An XML presentational vocabulary for specifying formatting is also part of XSL: XSL-FO.

3.6.3.1 Rationale for XSLT

In the early versions of this application, server-side DOM traversal was used to extract the data from XML into some adhoc SQL statements. Since these documents are intended to be exchanged, this approach was quite restrictive and of little or no practical bearing.

Like its XML-based predecessors, XSL is relevant for data-exchange since it enforces *separation of concerns*, enabling a clear separation of the data flow from presentation within the application. XLS defines *how* the document should be processed, the rest is left to the developer. As such, there is an abundance of languages, platforms and technologies that can be applied for that purpose, some of which are:

* Java: SAXON or Xalan
* Multi-interfaced: The XSLT library from the GNOME project is the most widely implemented library I have come across. Written in C, it is ported to almost all interpreted languages (including Python, PHP).
* Web browser: Most modern browsers support client-side XSL transformation, if you direct the user to a document that has an XSL stylesheet URL embedded, the browser will render that document into some output. Moreover, Javascript has also a cross-browser XSLT library that one can invoke programmatically. Transformation done through interpreted Javascript in the browser runs significantly slower than cases where the processor is compiled (as above).
  Since JS is inseparable from the browser, we will often use them interchangeably to mean the same thing.

3.6.3.2 Summary

Having different interfaces interpret the same document implies substitutability: both the stylesheet documents and the processing tools can be swapped at libitum. The templates exist outside of the application logic and may even be under the control of a separate team of developers.

3.7 XML schema Design

Schemas are formal notation of the syntax of an XML language. With a schema processor, we can check adherence of an XML document to a schema.

There are two prominent schema languages, RELAX NG and W3C XML Schema. Document Type Definition (DTD) has been effectively replaced by XML Schema. Unlike DTD, both former languages support data typing (datatypes other than character data), regular expression content model, constraining facets, namespace and are able to reference complex definitions with varying nesting depths. DTD also does not itself use an XML syntax. Namespaces entail modularity: two elements (with the same type, name and content model) can reference different URIs: no ambiguity (different NS, different scope).

Some features of XML Schema include:

* Intrinsic data types: *integer*, *string*, *boolean*, *dateTime*, *Base64Binary*, etc. Other data types can be derived from these built-in types.
* Constraining facets: These facets can be applied to datatypes to constrains different aspects of the value.
  – Length: minLength, maxLength,
  – Occurrence: combinations of min/max-Inclusive/Exclusive. also "nullable",
  – Cardinalities: minOccurs, maxOccurs,
  – Value: pattern(regex), enumeration, whiteSpace, totalDigits, fractionDigits,

Due to limitations of the XML Schema specification, dissatisfaction has driven independent groups (and even some of W3C's own working groups, XBL, SVG 1.2) into another equally powerful alternative: RELAX NG.

3.7.1 RELAX NG

REgular LAnguage for XML Next Generation is another schema language that is developed by an "independent" group. RELAX NG is relatively easy to read, easy to write, but doesn't have all the advanced features of XML Schema. Besides that, it's compact, alas non-XML syntax, is far cry from other complicated syntaxes, particularly the one of the XML Schema. Given a document with a structure similar to the one in table 3.1, we can write a RELAX NG grammar in compact syntax:

Table 3.1: XML document subtree

```
<committees>
<committee>
  <member>
    <name>John</name>
    ..
  </member>
  ..
</committee>
<committees>
```

Table 3.2: RELAX NG in compact syntax

```
start      = element committees { committee* }
committee  = element committee { member+ }
member     = element member { name }
name       = element name { text }
```

A RELAX NG parser then translates the compact syntax in to a corresponding schema in XML syntax without losing significant performance. XML Schema's expressiveness comes at a cost: complexity. It is inherently verbose (syntactically) and difficult to understand and implement correctly, a compact syntax similar to the aforementioned would be well-suited.

3.7.2 Definitions

Some terms that are used in the following sections are introduced here.
By a "global" element, we understand a node that is a direct child of the root "`<schema>`" node. Conversely, a "local" element signifies a node that is nested within other elements and types (complexType, simpleType XML Schema structures).

SimpleType : `<cfp:simpleType..>`
Defines a family of Unicode text strings (i.e. no markup). Simple types can be derived, either by restriction or extension.

Complex types : `<cfp:complexType..>`
Defines a content and attribute model.

Element declaration : `<cfp:element..>`
Associates an element name with a simple or complex type.

Attribute declaration : `<.. name="keywords">`
Associates an attribute name with a simple type.

3.7.3 Language Vocabulary Design

In the following, some design choices concerning the data types of schema are mentioned. Different schema designs are discussed and evaluated.

3.7.3.1 Salami Slice Design

`Summary`: global elements and local types.
The Salami Slice design can include arbitrary levels of nesting and complexity with mixed content and semi-structured data that one might find difficult to refractor or extend.

3.7.3.2 Russian Doll Design

`Summary`: Local elements and local types.
Like it's namesake of the wooden figures, the *Russian doll*, this design contains stacks (elements and types) of decreasing sizes placed one inside another. This design cannot define recursive elements, where an element can be included within an element of the same type as immediate child or indirectly in a child element. This is due to the unrestrictive depths these interleaving nestings may have.

3.7.3.3 Venetian Blind Design

`Summary`: local elements and global complex types.
Here, type definitions are used to make references to other components. This feature allows data types to be defined once and reused, preferably using the `type` reference rather than `ref` which is restrictive as elements that are referenced once can not be re-used in other elements.

3.7.3.4 Garden of Eden

The Garden of Eden is a combination of the Venetian Blind and Salami. All elements and types are defined in the global namespace with the elements referenced as needed.

3.7.4 Best practices

Some practices make schema authoring easy, but are bound to complicate any updating, reuse or extension of the schema. Some designs (Salami Slice, etc) enforce reusability through element reuse (involves replicating: bad) while others, like the Venetian Blind, do this task through reuse of types (no replicating, only referencing: good). Russian Doll and Salami Slice, being on each end of the design spectrum, require more effort to update and extend than the "Venetian Blind" design.

W3C XML Schema has a steep learning curve. One of the more poignant *DO's and DON'Ts* of XML Schema I read suggested that one should not *try to be a master of XML Schema. It would take months.*

3.7.4.1 Current Schema

Elements of extensibility and reusability are introduced into the current schema model. The current schema makes use of the "Venetian Blind" (VB) design, a modular approach where all type definitions are global. All constructs (schema, attribute, element, complexType, sequence, etc) declared within the document are explicitly made namespace qualified (:cfp).

VB contains a single global element, in our case, this element is conveniently called "CallForPaper", which nests local elements (that nest further local elements). These local elements are defined in terms of simple and complex types that are within the global namespace.

Hierarchy: the stacking bears a faint resemblance to the "Russian Doll". Elements relationships are nested in logically descending order:

e.g `<callForPaper>`→`<committees>`→`<committee name="progChair">`→`<members>`→`<name>`, where the arrow signifies depth, and the attribute means that this particular person is member of the program chair (committee).

This design is primarily chosen in reference to it's simplicity, extensibility and modularity. Since no assumptions have been made about incorporating schema into other applications, we can not afford to make assumptions about the scope of the elements.

Table 3.3: Venetian Blind Design

```
<cfp:complexType name="personType">
<cfp:sequence>
  <cfp:element name="name" type="stringtype"/>
  <cfp:element name="institute" type="stringtype" minOccurs="1"/>
  <cfp:element name="url" type="stringtype" nillable="true" minOccurs="0"/>
  <cfp:element name="mail" type="stringtype" nillable="true" minOccurs="0"/>
</cfp:sequence>
</cfp:complexType>

<cfp:complexType name="CommitteeType">
<cfp:sequence>
  <cfp:element name="member" type="personType" minOccurs="0" maxOccurs="
      unbounded"/>
</cfp:sequence>
<cfp:attribute name="name" type="CommitteeNames" use="required"/>
</cfp:complexType>


<cfp:complexType name="CommitteesType">
<cfp:sequence>
  <cfp:element name="committee" type="CommitteeType" minOccurs="0" maxOccurs
      ="unbounded"/>
</cfp:sequence>
</cfp:complexType>
```

This schema snippet in listing 3.3 contains an extensible content model for a `<member>` element. The name and the institute are required, while the url and email can either be left out, or present without their normal content (nillable is equivalent to production rule ”EmptyTag”, see 3.8.1.1).
The use of ”required” with the attribute ”name” signifies that committee names have a fixed attribute model defined by type ”CommitteeNames”. The prefix ”Type” and capitalization of the complex types is merely a convention: type definitions and element names do not collide, in the above snippet, complexType name ”CommitteeType” could readily be substituted with ”member”.

This type of modularization has some benefits:
Multi-schema coupling:
content model of element can be derived beyond what was specified by the schema - derivation by either restriction or extension. If we needed a more restrictive design, e.g where a committee is required to have at least one member, a third party or another developing team could import a secondary schema and use ”derivation by restriction” to impose restrictions on that particular type and to change *minOccurs=”0”* to *minOccurs=”1”* without modifying the main schema.

Type derivation by restriction :
Restriction defines a more restricted datatype by applying constraining facets to the base type.

In table 3.3, values of child elements under a person are constrained to the type "stringtype", which is a *simpleType* with no constraints. In other locations we may need to get rid of excess whitespaces. This is done through "derivation by restriction". Here, the elegance lies in the modularization:

```
<cfp:simpleType name="stringtype">
  <cfp:restriction base="cfp:string"/>
</cfp:simpleType>

<cfp:simpleType name="CollapsedString">
  <cfp:restriction base="cfp:string">
    <cfp:whiteSpace value="collapse"/>
  </cfp:restriction>
</cfp:simpleType>
```

The white space normalization rule '`collapse`' interprets consecutive white space characters into a single space character.

Type derivation by extension :
Extension is an operation that involves adding extra attributes or elements to a derived type. In this scope, *reusability* is achieved by using types as building blocks within the namespace of both the current schema and in external schemas.

An invited speaker, `invitedSpeaker` element, is derived from a `<member>` type, with an additional element "topic", this is "derivation by extension". Notice the snippet "`<cfp:extension base="personType">`"

```
<cfp:complexType name="speakerType">
  <cfp:complexContent>
  <cfp:extension base="personType">
    <cfp:sequence>
    <cfp:element name="topic" type="stringtype" nillable="true"
        minOccurs="0"/>
    </cfp:sequence>
  </cfp:extension>
  </cfp:complexContent>
</cfp:complexType>
```

Type substitutability: The contents of an element of type `A` can be substituted by any element of the type `A` or of any type that derives from `A`.

Beyond enabling extensible content model and logically nested architecture, the "Venetian Blind Design" also implies lack of rigidity in the document hierarchy: in the snippet below, the `<committees>` element can be moved up or down by merely moving it's type, opposed to the "Russian Doll" approach where it would be the lengthy content model that must be moved (some elements are left out for brevity).

```
<cfp:complexType name="callForPaperType">
  <cfp:sequence>
    <cfp:element name="info" type="eventInfoType"/>
```

```
      <cfp:element name="importantDates" type="importantDateType"
           maxOccurs="unbounded"/>
-->   <cfp:element name="committees" type="CommitteesType" maxOccurs="
    unbounded"/>
     </cfp:sequence>
   </cfp:complexType>
```

Regular expressions in XML Schema :
One of the constraining facets for the `string` type accepts regular expressions. One particular pattern used checks emails for validity. A crude regular expression pattern that I used earlier:

```
[^@]+@[^\.]+\..+
```

which simply matches anything that has: token(address) followed by @, followed by another token(domain), then a dot, then more tokens.. ad libitum. The one below uses meta characters and grouping to be more prudent:

```
([\.a-zA-Z0-9_\-])+@([a-zA-Z0-9_\-])+(([a-zA-Z0-9_\-])*\.([a-zA-Z0-9_
    \-])+)+
```

(lifted from a thread on the XMLBEANS mailing list).

The dot (scaped by a backslash) denotes any character defined in the Unicode standard. Initially, dates were also matched by regular expressions, but were removed in favor of a more powerful parsing at the server-side.

The design of the schema has gradually changed through the development of this application. To visualize the composition of the document tree, a radial tree layout is used: the hierarchical structure is conceptually interpreted as pattern where nodes ”radiate” from the apex (root element) and outwards towards child elements. Tree nodes are oriented radially, ie, with their length axis pointing towards the center of the graph. Figures 3.8 and 3.9 illustrate the gradual change of the design.

Note the excessive nesting in the visualization of the former version (figure 3.8), so deep that some elements poke out of the visualization window. This would suggest that the initial design was more of a ”salami” than the later version which inhibits appropriate stacking where the ”`<name>`” of a person is nested within a ”`<member>`” node, that is nested within a ”`<committee>`” node, that is in return nested in a ”`<committees>`” node and so forth, i.e a mixture of ”Russian Doll” and ”Venetian Blind” design.

In large schema documents like the current version, the ”Salami Slice” and ”Russian Doll” designs should be used with a healthy degree of scepticism; if the schema document is worth 250 lines of code, it might be hard to convince the reader that practicality of extensibility or reusability of this document is attainable with either of these designs:
”Russian Doll” is excellent for small, single application schemas where types need not be reusable, while ”Salami Slice” is suitable for fixed/static schemas where modifications to the standard elements are either unlikely or unnecessary. Also noteworthy in figure 3.9 is how local elements expand much like a flower: a root element contains a global element, that again has other global/local descendants, minimizing the number of entry points. A similarly visualized template document is found in appendix B.3.

3.7.4.2 Design Iterations

Authoring documents is intended to be elementary: no advanced authoring or editing tools are assumed. Some steps are taken to introduce simplicity and continuity into the schema design. The initial design was similar to the flat structure, illustrated on table 3.4, which morphed into a more logically consistent hierarchical structure. The difference is illustrated below.

Table 3.4: Initial flat design

```
<date>
 11/10/2007
</date>
..

<person role="steerComm">
 <name>John Doe</name>
 <email>j@doe.net</email>
</person>
```

Table 3.5: A more tree-like robust design pattern

```
<date>
 <day>11</day>
 <month>10</month>
 <year>2007</year>
<date>
    ..
<committee name="steerComm">
 <member>
  <name>John Doe</name>
 </member>
 <member>
    ..
</committee>
```

While the first inception might be easier to author and less verbose, integrity, or the lack thereof, is a primary concern. The second design structure removes ambiguity: is the date-format DD/MM/YY or MM/DD/YY? It also supports bulk-editing, ie. we can insert multiple members into a committee, by means of copy-and-pasting, without editing member roles inline.

The schema document is found, in it’s entirety, in appendix A.2. The graph shown on figure 3.10 illustrates the structure of an arbitrary XML document that matches the proposed schema. Labels on relationship arrows correspond to the cardinalities in the relationships.

3.8 Regular Tree Grammars

Expressiveness can be seen as a token of the order in which schemas have evolved in, DTDs are less expressive than XML Schema which is less expressive than RelaxNG. This degree of expressiveness in rooted in the type of grammar that govern these languages. XML Schema is a single-type tree grammar.

Formally,a regular tree grammar is a 4-tuple[Com07]:
$G = (N, T, S, P)$, where

* N is a finite set of nonterminals
* T is a finite set of terminals
* S is a set of start symbols, where $S \subseteq N$
* P is a finite set of production rules of the form $X \rightarrow a[r]$ where $X \in N$, $a \in T$, and r is regular expression over N

A single-type tree grammar is a regular tree grammar such that

* for each production rule, non-terminals in its content model do not compete with each other, and
* start symbols do not compete with each other.

Two different non-terminals `A` and `B` are said competing with each other if:

* one production rule has `A` in the left-hand side,
* another production rule has `B` in the left-hand side, and
* these two production rules share the same terminal in the right-hand side.

This lack of non-terminal competition means that the data types can be *determined* from the *names* of the globally available pool of elements. This determinism is elaborated on in the next section.

3.8.1 Document Grammars

Every XML document that is well-formed and satisfies the namespace constraints is said to have an XML Information Set(Infoset). An *Infoset* describes the abstract data model of the information that is stored in an XML document. A non-empty set of items in that *Infoset* contains some[9] *information items* that constitute the abstract representation of one or more elements in the document. In return, each information item has a set of associated named properties. As of the current specification, there are 11 information items, 8 of which are relevant in our schema:

* Document Information Item,
* Element Information Items,
* Attribute Information Items,
* Processing Instruction Information Items,
* Character Information Item,
* Comment Information Items,
* Document Type Declaration Information Item,
* Namespace Information Items.

Following is a simple, yet somewhat formal description of a language grammar that well-formed documents must abide by.

3.8.1.1 Context-free Grammars

By a grammar, we understand a set of rules that govern how **valid** sentences in a language are constructed. By grammar validity, we understand two sets of rules:

Syntax :
Rules that only concern the *form* of a document. An informal description previously given in section 2.1.5.1. A slightly more formal one is given in the next paragraph.

[9] Two or more information items.

Semantic :
Rules that concern the *meaningfulness* of a well-formed document. As we have seen in 2.1.5.2, Semantical correctness implies syntax correctness. Informally, this correctness is enforced by some validity constraints. By schema validation, the following definition is understood:

> Let σ be an instance schema expressed in the grammar of XML Schema and let $\delta(\sigma)$ be a function that defines the set of XML documents that are valid relative to σ:
>
> $$\delta(\sigma) = \{X|\text{X is valid relative to } \sigma\} \tag{3.1}$$
>
> Given an XML document X, we need to determine the membership of X, i.e $X \in \delta(\sigma)$.
>
> Context-free Grammars (CFG), introduced by Chomsky in mid 1950's, describe the phrase structure of natural language sentences. A CFG deterministically describes all possible 'sentences' so that a valid sentence has a single interpretation. This decidability is important when the above *membership* is verified.

Below, well-formedness constraints are specified using a simple Extended Backus-Naur Form (EBNF) notation: if an XML document grammar does conform to these rules, its language is not well-formed XML.
Regular Expressions: operators and closure operators are used as POSIX extended regular expression notation; non-special characters match themselves.

R|E : The union notation matches any string that is either in R or S,

,: R,S comma denotes concatenation,

*****: R* matches 0 or more repetitions of R,

+: R+ matches 1 or more repetitions of R,

?: R? matches 0 or 1 repetition of R,

(..): Grouping operators to their arguments.

Production rules follow:

```
document     ::=  prolog element Misc*
prolog       ::=  XMLDecl? PI* Misc* (doctype  Misc*)?
XMLDecl      ::=  '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
element      ::=  EmptyTag | StartTag content EndTag
```

```
content     ::=   CharData? ((element | Reference | CDataSect | PI |
     Comment) CharData?)*
Encoding     ::=  S ‘encoding’ Equals (’"’ EncodingType ‘"’ | "’"
     EncodingType "’" )
EncodingType::=  [A-Za-z] ([A-Za-z0-9._] | ‘-’)*
VersionInfo ::=  S ‘version’ Equals ("’" XMLVersion "’" | ‘"’
     XMLVersion ‘"’)
Equals          ::=  S? ‘=’ S?
XMLVersion   ::=  ‘1.0’
doctype      ::=  ‘’
Misc        ::=  Comment |  | S
Comment      ::=  ‘<!--’ ((Char - ‘-’) | (’-’ (Char - ‘-’)))* ‘-->’
Char        ::=  CarrReturn | Tab | Space | LineFeed | UniCodeChars
UniCodeChars::=  [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
CharData   ::=  [^<&]* - ([^<&]* ‘]]>’ [^<&]*)
WhiteSpace   ::=  (CarrReturn | LineFeed | Tab | Space)+
CarrReturn   ::=  ‘\r’ | #x9
LineFeed     ::=  ‘\n’ | #xD
Tab          ::=  ‘\t’ | #xA
Space        ::=  ‘ ’  | #x20

StartTag     ::=     ‘</’ Name  S? ‘>’
EndTag       ::=     ‘<’ Name (S  Attribute)* S? ‘>’
Attribute ::=      Name  Equals  AttValue
Reference ::=      EntityRef | CharRef
EntityRef ::=     ‘&’ Name ‘;’
CharRef      ::=     ‘&#’ [0-9]+;
```

Adding Namespace Support:
Given a namespace `ns` with a valid URI, an element identifier may appear as `<ns:element>value</element>`

```
    QName     ::=    (Prefix ‘:’)? LocalPart
    Prefix      ::=    NCName
    LocalPart   ::=    NCName
```

Adding Processing Instructions (PIs):
The XML declaration, which is itself a special PI, signified by the rule `XMLDecl` above, is not matched here. Since wee need to a append transformation grammar or style sheets (that is: either XSL or CSS) to an XML document, we **must** justify the use of `xml-stylesheet` processing instruction in the prologue of the document:

```
    PI        ::=  ‘<?’ Target (S (Char* - (Char* ‘?>’ Char*)))? ‘?>’
    Target    ::=  ‘xml-stylesheet’
```

The target names ”XML”, ”xml” constitute the reserved words in the XML specification. Under normal circumstances, all PIs, except any combination of `((‘X’ | ‘x’) (‘M’ | ‘m’) (‘L’ | ‘l’))`, would be allowed, but to simplify the notation, no PIs except XSLT stylesheets are expected, it’s target being ‘*xml-stylesheet*’. Also document type declaration is not deemed necessary at the current stage.

Adding extended character data:
Most of the time, there’s a need to store characters that would otherwise be recognized as markup, without having to escape them. Data within a ”CDATA”

section are interpreted as characters, not markup or entity references. Also, big chunks of text data, such as the whole content of a CFP document, **may** be stored within CDATA.

```
CDataSect  ::=    CDStart CData CDEnd
  CDStart    ::=    '<![CDATA['
  CData    ::=    (Char* - (Char* ']]>' Char*))
  CDEnd    ::=    ']]>'
```

Notice these two features:

Threshold :
: The minimal items defined by the ”document” rule: one *document information item* and one *element information item*, is the smallest XML document conforming to the *Infoset* paradigm.

Unicode :
: The XML version 1.0 specification, currently in its fourth edition, only allows characters which are defined in Unicode 2.0. The ”Char” production rule implies all Unicode characters, except the surrogate blocks, FFFE, and FFFF [XML06]. Despite this abundance,

The XML specification seems to indicate that valid character data in XML comprises all Unicode characters, excluding the surrogate blocks. The production rules `UniCodeChars` defines such a range (definition from the XML specification)

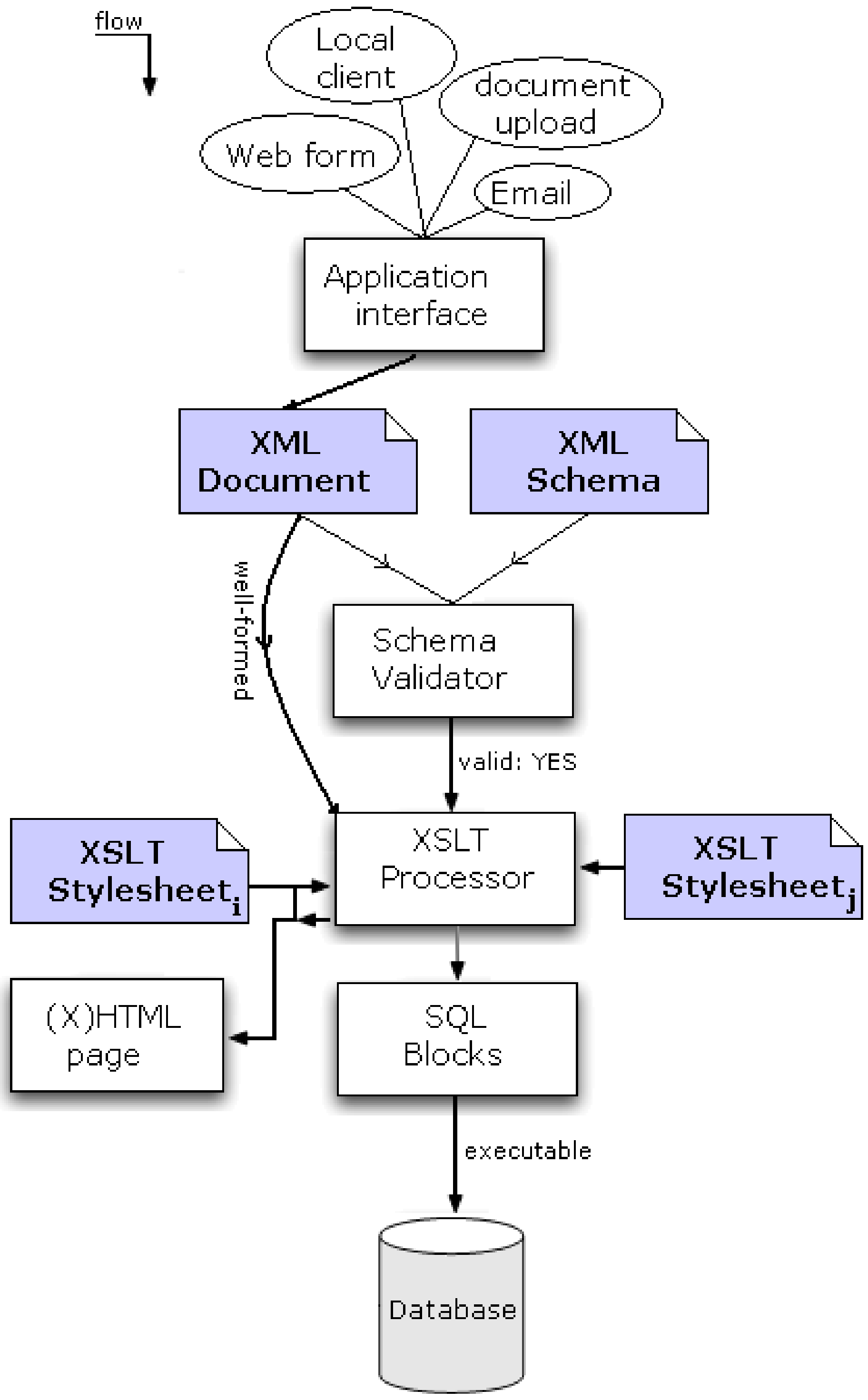


Figure 3.3: Application architecture

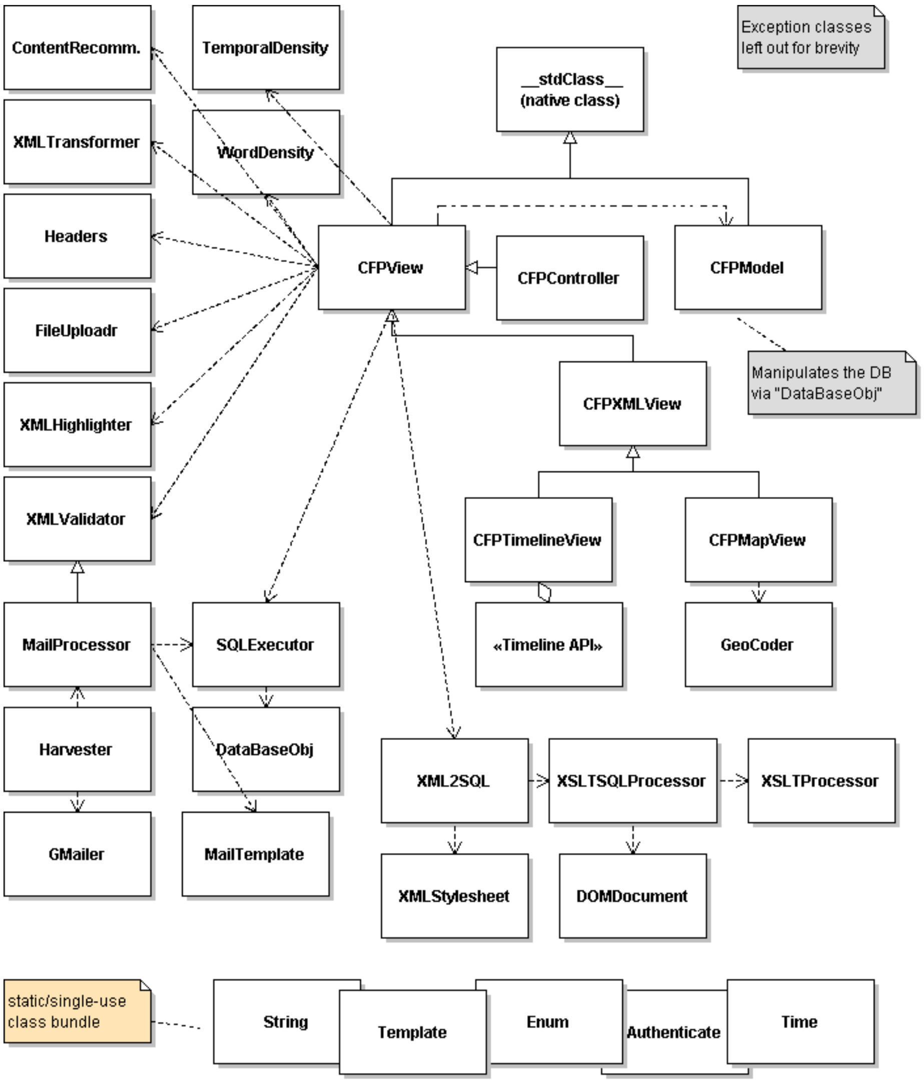


Figure 3.4: Hierarchy and dependency relations between classes

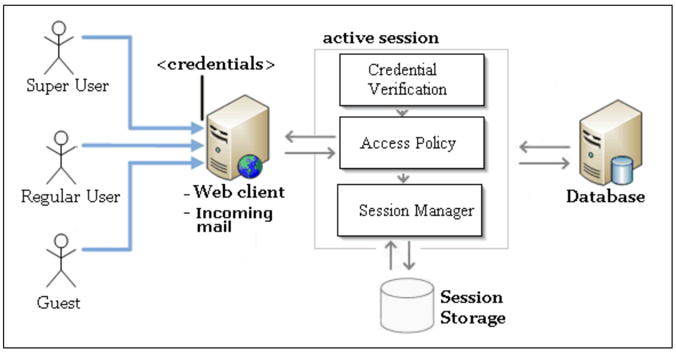


Figure 3.5: Simple role-based access control

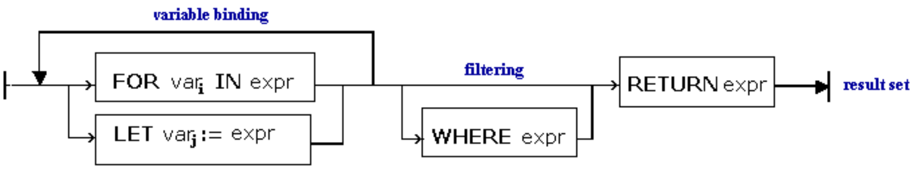


Figure 3.6: Processing sequence of FLWR expressions

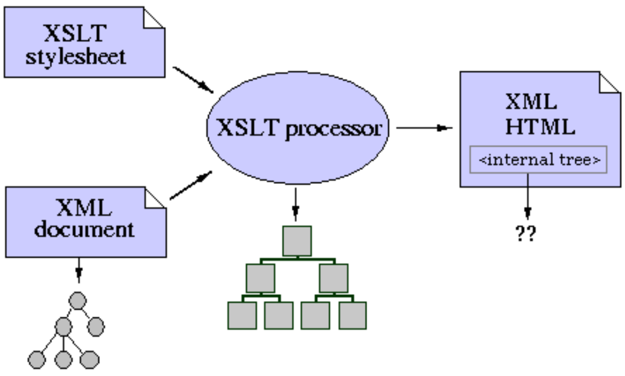


Figure 3.7: XSL Transformation scheme and components used.

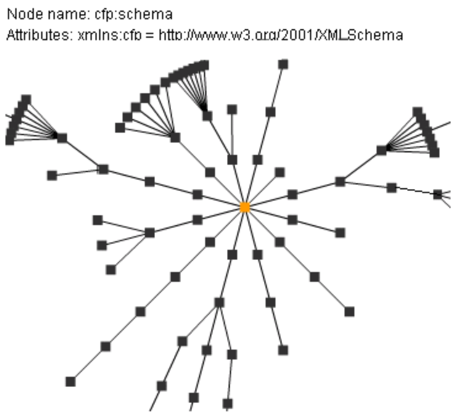


Figure 3.8: Visualization of the first schema design: Russian Doll

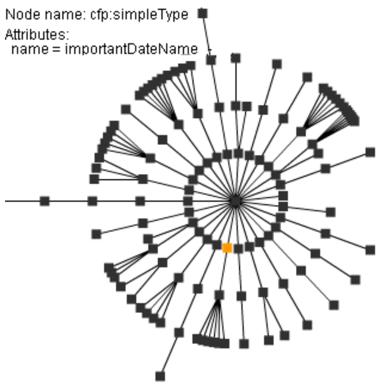


Figure 3.9: Visualization of the second, more lenient version of the schema: Venetian Blind design

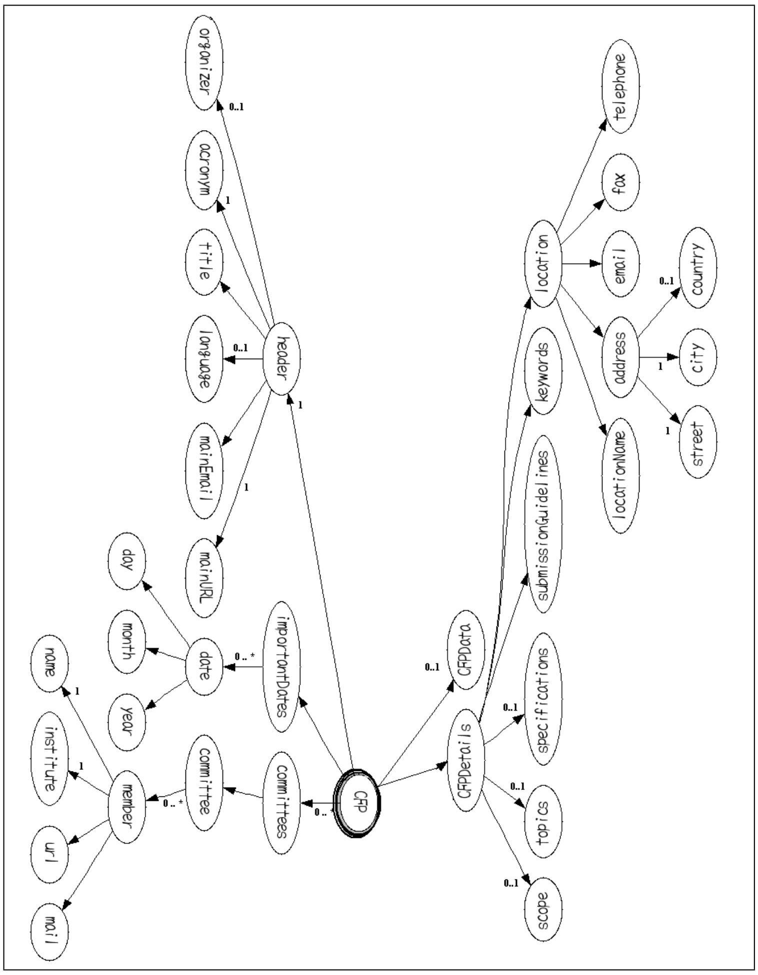


Figure 3.10: Document elements hierarchy, shown with cardinalities

Chapter 4

Implementation Details

In this chapter we're going to describe a number of important issues within the implementation. Some amendments or additions to the conditions mentioned in the previous chapter are also found here. Not every line of code will be examined and explained, but we'll try to explain the main ideas and include code when it can help ease the understanding of the proposed solution.

Before we delve into the details, for the sake of reference:

Prerequisites:

* PHP5 with default extensions (CURL, XSL, SimpleXML), *safe_mode* `off`
* MySQL 5
* Mozilla Firefox (Javascript enabled)
* libXML (part of GNU Project, see 4.3.2.1)

libXML needs to be compiled in the local environment, no dependencies on additional libraries were observed. See appendix A.1 for a short installation-guide.

4.1 Mail-interface Application

In the following sections, we will look into how parts of the domain problem can be solved using remote servers and local configurations as a part of a distributed system.

4.1.1 DTU Mail Service (IMAP, POP3)

DTU provides mail-services to it's users on a per user basis, both students and staff can access their accounts through mail-clients, but no further means are provided. I have tried in vain to connect through IMAP/POP3 through http (imap.student.dtu.dk, pop.student.dtu.dk, port 110, port 995, etc), and according to UNI-C, POP3 is going to be phased out as it is *inherently* insecure.

This lacking connectivity poses a design constraint. Since the application relays data through the Application layer with HTTP Request Messages, any connectivity to a remote system should ideally be achieved through the same protocol. Driven by these needs , we are persuaded to seek a reliable retrieval/storage environment; if this service could be provided locally within DTU's server-farm, so much the better.
In the meanwhile, we have to take advantage of a service operated by a third party, that is known to provide reliable and timely service: Google.

4.1.2 Distributed Application

The integration of a remote mail-handling application and some data-visualizing APIs that enhance overall comprehensibility, makes the system distributed (decentralized).

The approach taken in this section comprises a central mail repository, means for ascertaining the of newly arrived mails, means for harvesting these mails, and, most importantly, relevant procedures to process, validate and propagate this data into a database.

In this distributed architecture, the application logic is no longer in one large package. Instead, it's distributed across heterogeneous computing services and applications, and a database. As such, these subsystem need a coherently bridged interfaces for such a system to be feasible.

The instances of performance requirements imposed on this subsystem are of those mentioned in 2.1.1.2

4.1.2.1 GMail As An API Service

This subsystem of the application is built on top of an API that provides basic remote Gmail functions. These functionalities include: labeling, filtering, archival and dispatch of emails.
Furthermore, it's powerful filtering system can be used to perform arbitrary queries, which spawns several new ways of interacting or fetching mails.
The connection and traffic are channeled through HTTPS using native Libcurl[1] to safely transfer messages, attachment documents and sequences of other operations, such as DELETE, ARCHIVE, UNREAD, etc.
To remedy the deficiencies as described in 4.1.1, Gmail has a functionality called "Mail Fetcher" that retrieves mails from any service provider that offers POP access to it's users, effectively eliminating the syncing problem between mail accounts, should we choose to store incoming documents anywhere else.

Minor limitations, albeit irrelevant in this discourse, include: no support for retrieving mail from IMAP servers, nor does it support sending messages through an external SMTP server. Appendix B.15 shows how emails are retrieved through a secure connection and labeled at arrival.

In distributed applications reliability is crucial. Why Gmail is the near-perfect choice:

Abundant storage :
Gmail offers more storage than any other free service. '*Lots of space. 4.38 gigabytes (and counting)*' as it says on their front page.

Reliability :
Availability of service is not guaranteed, but experience suggests likelihood of continued availability.

[1] cURL library that is available in most languages, supports http, https, ftp, gopher, telnet, dict, file..

Connectivity :
They offer free POP access to Gmail from other email applications; Thunderbird,Outlook.
Access third-party, POP-enabled email accounts from Gmail. This application could be extended to use secondary email account as storage, google would then act as an interface, since it has an open API like no other. Alas, no advantages are foreseen with this setting.

Extensible :
Uses different approach to managing email: mails are treated as ”conversations”. Labels, filters applied to these conversations. Arbitrary queries like *get unread mails, with attachment, from a@b.dtu.dk* are supported.

Let us consider two scenarios that an administrator is likely to consider:

* Loose administering: submissions are accepted from a narrow subgroup of users, say within a particular institute at DTU. No reviewing is necessary, data is processed promptly (subject to harvesting intervals).
* Notification: the email address in use is solely used for the purpose of collecting Call For Papers, and all incoming mails are assumed relevant. An administering staff
* Remote mail server: A remote email account is used as the primary source. Submissions must be first fetched.

With these scenarios in mind, let’s set up two simple filters, expressed in natural language:

```
a) from:(*@imm.dtu.dk OR *@math.dtu.dk ) subject:(CFP:*) has:
    attachment
b) subject:(CFP:*) has:attachment
```

The first filter matches Call For Paper submissions made by all staff members at either of the *IMM* or *Math* institutes, given that the message has a subject prefix ”CFP:”, and contains an attachment. The later is less restrictive, and forwards a copy of the message to a designated address, (a screen showing both filters are found in appendix B.16, p. 129)

Extended filter expressions :
While the above rules serve our intended purpose, occasionally other needs may arise. To apply a rather strict rule, only accepting submissions for a particular conference, the filter

```
filename:.xml subject:(CFP: CC 2008) from:(*.dtu.dk OR cs.aau.dk)
```

catches messages in which the subject has the correct identifier and contains both the word ”CC” and the word ”2008”, and has an XML document as an attachment (extension, not the mime-type is significant). Furthermore, submissions sent through addresses at DTU or the department of Computer Science at Aalborg University are accepted. Rules are delimited by space, implying an `AND` operator between two successive rules. A little trick that removes that `AND` involves grouping: `(from:(ab@dtu.dk) OR (Compilers))` matches messages send by an individual or containing the word ”Compilers” in their body.

Also, when querying Gmail remotely, the filters `is:starred`, `is:unread`, `is:read`, `label:some-label` and `in:inbox` are extensively used. Apart from `AND` and `OR` operators, words are grouped with (), quotes, i.e ”foo”, search for an exact phrase, while hyphens are used for exclusion.

Filters can be grouped and formatted conveniently:

```
{
  subject:CFP:*
  has:attachment
  filename:.xml
  from:{
    *@imm.dtu.dk
    *@math.dtu.dk
  }
}
```

Since GMail uses a single-line input-field to formulate filters, the above notation is not applicable, as such. Line-breaks disrupt the input: content would truncate to the first line (i.e a ‘{’). Some ad-hoc *tricks* involving XML binding and Javascript are used to hide the input-field and instead, create a textarea containing sufficient amount of rows to accommodate the above notation.

XBL :

XML **B**inding **L**anguage[IH07] is an XML-based markup language used to associate executable content (script, event handlers, CSS) with an XML tag, that content is then executed within the context of the tag. This simple use involves adding a new DOM element into the XHTML markup. This is the XML markup used:

```
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl">
  <binding id="filter" styleexplicitcontent="true">
    <implementation>
      <constructor>
        ..executable content..
      </constructor>
    </implementation>
  </binding>
</bindings>
```

Javascript :

The executable content that introduces the new DOM element is short and precise. Markup selectors, such as width, height, font and margin settings are left out for brevity.

```
        var textarea = document.createElement("textarea");
        textarea.name = this.name;
        textarea.value = this.value;
                 ..
        textarea.rows = 15;
        this.parentNode.replaceChild(textarea, this);
```

The The last line replaces, through the parent of the specified element, the input-field with the newly created child node (textarea). The input-field element to which the binding has been applied is now rendered as a *bound element*.

Binding :

Binding involves event handlers that watch for events both on the document level and on the *bound element*. Elements matching a specific selector are implemented by a particular binding:

```
@-moz-document domain(mail.google.com) {
  input[name="cf1_has"] {
    display: none;
    -moz-binding: url(http://cfp.smallmeans.com/js/gmail-field-
         binding.xml);
  }
}
```

This binding is triggered by two events: the current domain of the page matches "*mail.google.com*" and there's an input-field named "*cf1_has*" on page, only then would the rules apply:

a) the element matched is no longer needed, thus hidden. b) the content expressed in the document specified by the URL is bound to the element.

This code would be invoked through a bookmarklet, that injects the above css rules into the current page and unloads it whenever page is left or reloaded. To enable a permanent binding, one can add the above css markup to *userContent.css*[2], which sets the display rules for web content.

[2]usually located in the sub-folder called chrome in one's profile folder, Mozilla FireFox specific

XBL is currently a W3C Candidate Recommendation (as of March 2007) and a Mozilla proprietary language, the only implementation being the Gecko layout engine, thus only browsers based on this engine are supported. This limitation is inline with the machine requirements specified in

4.2 Java Client

In this section a short description about the Java client is given. Note that terms and notations used so far, such as *class* or *interface* may no longer mean or refer to the same concepts.

4.2.1 Initial Design: Classes and Interfaces

Traditionally, if we need to create an XML document that is intended to be schema-aware, we either concatenate strings (`s+="<elm>value</value>"`) or we use some programming interface, SAX/DOM in one form or another, to create each element, determine the appropriate nesting, then add each child element into the Document Object Model tree. Validate. Does it validate? if not, it's back to fiddling with the tree: create, nest, repeat. This process is repetitious and unnecessary.

Initially, these interface objects were made: `CommitteesInterface`, `SpeakerInterface`, `MemberInterface`, etc. Each of the basic components where made of classes that *implemented* these abstract types. General class objects such as `Committee` were then be *extended* into specialized subclasses such as `ProgramCommittee`, `OrganizingCommittee`, and so forth. Also, the structure of the XML document was, and still is, based on this notion of hierarchy.

This initial concept, combined with the design of the GUI components consumed a huge amount of time, and would probably have taken more time, would I have not stumbled upon XMLBeans.

4.2.2 XMLBeans

XMLBeans, part of Apache Software Foundation XML project, is a Java-to-XML binding framework that does what the typical XML parsers do, and is schema-aware on top of that: schemas are compiled into Java types. Given a schema document[3], XMLBeans binds XML to Java types by providing interfaces, classes, and schema binaries that be used as building blocks when documents that **must** conform to that schema are authored, accessed or modi-

[3]Whether schema languages other than XML Schema is supported is unknown to me

fied. These object-based types mimic native Java types with accessor methods *setters*, *getters* and other typing constructs, for each Infoset in the XML model.

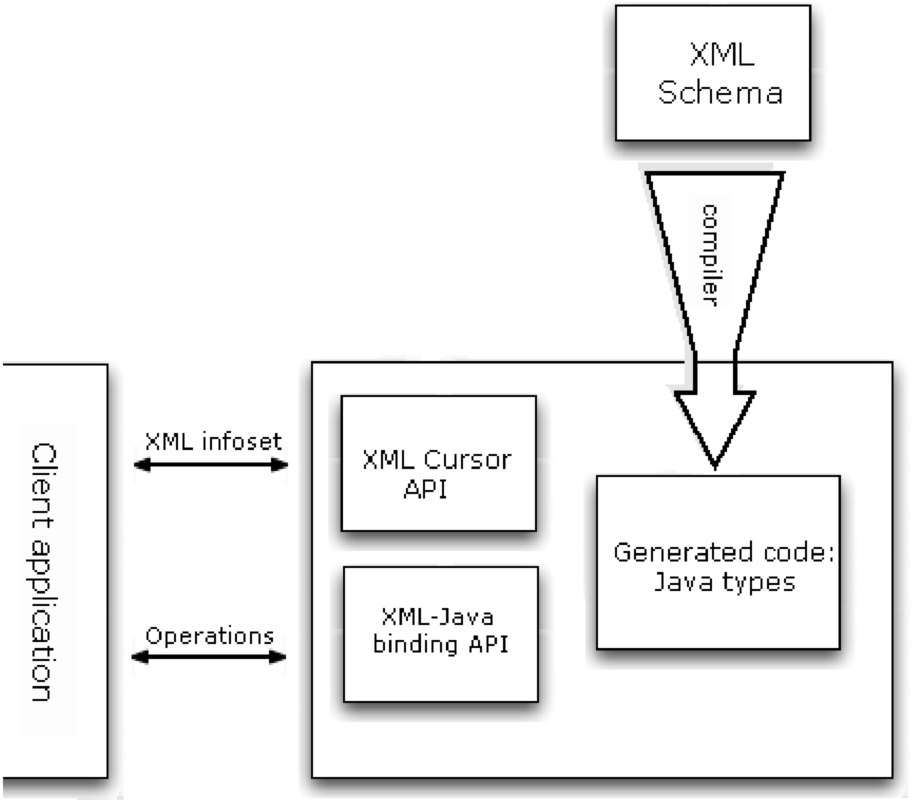


Figure 4.1: XML-Java data binding and code-generation

Figure 4.1 shows how the the schema is compiled into XMLBeans classes (jar file) and instances of the schema bound to Java types that can be invoked as any other type. Apart from these types, the full XML infoset is also accessible through an instance of XmlCursor, using XQuery. Good design coincides with good performance. Let's examine how useful this novel approach is by extracting some XML nodes from the schema:

```
<xml>
 <callForPapers>
  <cfp eventID="">
   <committees>
    <committee>
     ..
   <keywords>
    <keyword>
```

A straight-forward, yet extremely elegant way of creating these instances is as follows:

```
CallForPapersDocument doc = CallForPapersDocument.Factory.newInstance
     ();
CallForPaper cfp = doc.addNewCallForPapers();
cfpContainer = cfp.addNewCFP();
cfpContainer.setEventID("1889923");

CommitteesType committees = cfpContainer.addNewCommittees();
CommitteeType org = committees.addNewOrganisingCommittee()

EventInfoType info = cfpContainer.addNewInfo();
info.setTitle(title);
info.setAcronym(acronym);
...
CFPDetailsType details = cfpContainer.addNewCFPDetails();
details.setScope(scope);
details.setSpecifications(specs);
..
KeywordType keywords = details.addNewKeywords();
keywords.addKeyword("Automata");
keywords.addKeyword("Languages");
```

Notice how convenient these objects are named, with appropriate prefixes such as *addNew*,*set*,*add*: these reflect the structure of the schema. The `simpleType` elements are *derived* from `XmlObject` (base interface), thus there is no need to create an instance of them. Meanwhile, Complex types *are* `XmlObjects`: note the difference between `KeywordType` and it's content model, `keywords` (complex vs. simple type).
Loading in documents that conform to the schema and traversing/modifying it's infoset is equally convenient:

```
for(PersonType member : committee.getMemberArray()){
  member.setName("John");
  member.setCountry("DK");
  System.out.print("member: "+member.getName()+", "+member.
       getInstitute());
     ..
}
```

If an advanced integrated development environment with syntax suggestions (eclipse) is used, authoring schema-based XML documents with Java applications is elegant, extremely fast and requires little or no effort. The only prerequisite being the XMLBeans framework, that's used once to compile the schema (binary: scomp).
A minor drawback is that the schema becomes obsolete once modified, necessitating a re-compile.

4.2.2.1 GUI

The graphical interface is *clean* and intuitive.
As outlined by Chris very early, the idea was to import the whole content of

a CFP into some text container, the relevant data can then be put into the appropriate input-fields , in a pick-and-choose fashion.
The main window it self contains two inner windows: a contentarea where users can paste text from the Call For Papers, and an input area containing fields, in which this data is typed. The input area has a tabbed container that also has a *splitpane* component where users can view the document tree (implemented as a expandable/collapsable *JTree*), save the content of the XML document or validate against the schema.

Regular-expression patterns are interpreted by a class (RegExParser) that is implemented to allow, or rather restrict, user to type data in a predefined format. Participating committee members are expected to stated as: *Name (institute, country)*, failing to do so results in an exception that is accordingly displayed at the user interface. When all the necessary fields are filled, the aforementioned Java types are used create a `CallForPapersDocument` XML object that contains all the typed data as child elements.

Appendixes B.1,B.2 on page 118 illustrate how the UI of the Java client is designed. Also the inherent tree-like notation of the `XmlObject` is used the render the XML tree, visible on the left side of appendix B.2.

4.2.3 Summary

With it's presence, this client would have a diminishing effect on the overall usability of the proposed application. Despite the abundance of the interfaces that are implemented, the Java client can still be useful in occasions where persistent internet connection can not be assumed.

4.3 Web-interface

4.3.1 Data Visualization

Some of the visualizations are provided by using an external API. Google Maps^{TM}and Timeline provided by the SIMILE[4] Project under MIT's CSAIL lab.

These APIs are chose in reference to their:

* Agility and completeness,
* Mergeable modules,
* Good power-to-weight ratio,
* and ease of implementation, customization.

Both APIs require familiarity and in-depth knowledge, this is even more so for the Timeline API. While Timeline can be used without restriction, Google Maps require an API Key that must be registered with a particular domain. The current API Key will only work with the domain URL specified in , pursuant to Google Maps API's Terms of Use.

4.3.1.1 Database "Views"

Both Timeline and GMaps need an XML document as data-source. Timeline uses a relatively simple proprietary XML format where all the data must be put in pre-defined placeholders, such as `<event start=".." end=".."..>`.

Google on the other hand, has a very loose approach to being an API: it provides a programmable interface onto which all the pieces are "put" on. Not do we only need to propagate the data onto a self-authored XML document, but the well-formedness and parsing of that document is on our part as well. A a single CFP instance looks like this in the data-source:

```
<callForPaper id="14" deadline="Nov 14th, 2007">
  <title>International Conference on Compiler Construction</title>
  <city>Budapest</city>
```

[4]SIMILE (Semantic Interoperability of Metadata and Information in unLike Environments) is a joint project whose aim is to develop *Semantic Web technologies that improve access, management and reuse among digital assets,schemata/vocabularies/ontologies, metadata.*

```
    <country>Hungary</country>
    <lat>47.498403</lat>
    <long>19.040759></long>
</callForPaper>
```

This document document is loaded and parsed with Javascript using DOM to dynamically access and update the content of the map, and create markers and annotations. Appendix B.6,p. 123 shows a map used to visualize geographical locations of event venues.

This mapping could be accomplished using few classes of action-script in Flash, but major key functionalities in Google's rich interface would be difficult, if not impractical to implement in Flash.

4.3.2 Document Validation

This subsystem is concerned with verification of the correctness of XML documents according to the required grammar.

4.3.2.1 libXML

libxml is the XML C parser and toolkit developed for, but also used outside the Gnome project. libXML provides binding for other languages, and contains tools and libraries that are portable to many platforms.

libxml is extremely fast: Of all available multi-platform XML parsers, libXML is the fastest in parsing – both DOM and SAX – and XSL transformation benchmarks, see benchmarks B.11, B.12, B.13, B.14, starting on page 126. These benchmark data are from the *XML Benchmark Project*.

The colors used in these charts denote the content of the XML files used in the benchmark:

* **XMLGEN**: autogenerated simple XML file filled with random values, have 3 levels of deep and very simple XSD schema.
* **OPCGEN**: autogenerated sequence of OPC messages filled with random data. This messages have 1-5 levels of deep and complex xsd schema.

There are 15 different type messages which dramatically differs from each other.

* **RDF**: A large RDF document (11MB).

This library should be compiled on the installation environment; no particular library-dependencies were encountered during the build, but it may be otherwise for other environments. See a short installation guide in A.1 or seek full documentation at the GNOME XML Project's website.

4.3.2.2 Error-handling Ad Libitum

An XML schema processor can generate an awful lot of different errors, some of which are very unintelligible, or confusing at best. An example:

```
Element 'country': [facet 'minLength'] The value has a length of '0';
this underruns the allowed minimum length of '1'
```

A simple error-handler that uses pattern-matching to recognize and categorize errors is implemented. The above error can be deciphered to mean: "*the content of the element* `<country>` *is not of the expected type*" (mostly raised by an empty string). Due to the heterogeneity of the system, errors should be handled consequently. Some effort has been put to introduce simplicity into the error-handling process. Both parser errors – caused by ill-formed documents– and schema validity errors are interpreted to intelligible, to-the-point descriptions, such as:

Line 14: Element **mainEmail**'s content '**asd@d**' is not accepted by the pattern of **valid-Email**
Line 117: Element '**location**' was not expected on line 117, instead one of '**keywords**' element was expected

The idea is to use the markup of the schema and the XML document as a hint to give users a bearing of where and what causes the error, no ad-hoc checking or field-to-error mappings are made at the application level, which would complicate any schema or server-side application updating/extension process at a later point, instead the pattern-constraints and the XML nodes are named accordingly: the names **mainEmail**, **valid-Email** are directly bound to the XML document element and it's corresponding pattern constraint in the schema, respectively.
Besides being succinct, these lines also provide a useful way to bring the viewer's

attention to focus and hopefully remedy the error: a hyperlink or visual highlighting is used, whichever is applicable. If the document data is submitted through the web-interface (ie either the web-form or file-upload), the input-fields are given the same IDs as the XML nodes that they are transformed into at the back-end: at the event of an error, the client-side (browser/mail-processing module) is provided with an error-message and the name of the violating input source, this identifier is then used to either visually highlight input-fields(form) or provide a point-and-click method to identify violations.

Screen dump B.4,p. 121 shows a web-form that contains some violating input and error-messages atop. Also note that explanatory hints are provided when an input-field receives focus. First-time users may use this mechanism as a useful entry-point, but regular users might need a way to switch hints on/off.

Stack Trace :
Additionally, documents submitted through the mail-interface are forwarded to a page that contains the source of the XML document with the violating elements highlighted.
This component implements a simple error-handler that imitates the functionalities and appearances of the W3C XML/CSS markup validation services. Both a visual and textual descriptions of violating inputs are provided as hints. Appendix B.5,p.122 contains a screen dump that shows how these mechanisms are presented at the UI.

4.3.3 Inline Editing of Documents

Conventionally, *viewing* and *editing* of web-pages are disconnected: separate pages are used for each purpose, increasing the perceived complexity of the web-page and the time spent on the task at hand and loss of continuity (from user's point of view). With *inline editing* the composition of the page is changed to accommodate both tasks and provide enhanced 'single-page' experience. The underlying mechanism has a two tier paradigm: a) user interface manipulation, and b) asynchronous call to the database to change data. The behavior of the overall procedure is as follows:

a Paragraph appearance changes on mouse-events: *onMouseOver*, *onMouseOut* events,

b If and then the user clicks on paragraph, hide and replace it with a

`<textarea>` and save/cancel buttons (any previous edit session must be canceled, on a request by Chris),

c Undo all of the above steps if the user *cancels* the operation,

d When the *save* button is clicked, commit the changes to the database (server request). Show that the request is processing, in interim,

e When server response arrives, update the page accordingly

Due to the security concerns related to the use of GET requests, during these *commits* server calls are made with POST.

4.3.3.1 Event Attachment in Javascript

Some of the functionalities described in this section require Javascript. In general, most pages will function properly without JS, but editing/validation/visualization pages use techniques that will fail silently in the absence of JS.

In Javascript, events are the behaviors through which the internal application code can be invoked. The behavior that enables *inline editing* can be attached to the document model through either of two methods, an inline attachment and a DOM based attachment:

```
a)  <p id="specifications" href="#" onclick="javascript:edit(this);">
b)  getElementByClass('editable').addEventListener('click',edit);
```

Method a) is over-constrained, lacks modality and requires code put on each editable paragraph. Method b) is elegant, more robust, and modular.

DOM Attachment : Currently, Javascript DOM does not contain a native method for retrieving elements with a given class name. This operation is be equivalent to the XPath expression: `p[@class='editable']` that matches a possibly empty set of **p** elements with their class set to "editable". Practically, this is done through two-step algorithm;

* The native "*subtree*.getElementsByTagName"[5] is used to get a collection

[5]Since we have extensively dealt with namespaces in this document, search can indeed be restricted to XHTML elements belonging to a given namespace, look into *getElementsByTagNameNS*

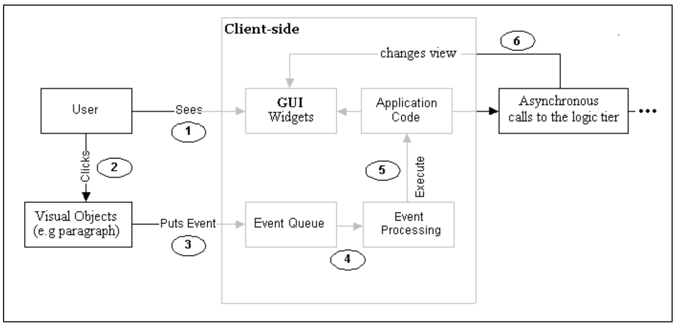


Figure 4.2: Client-side event sequence for user interaction

of paragraphs, `C`, in the document. *subtree* is the tree-node whose descendants are included in the search.

* For each element in `C`, A regular-expression check is performed on the class attribute. Elements for which that predicate returns `true` are pushed onto a stack.

* The stack is returned

Figure 4.2 how these events are propagated from the UI through to the application logic and back to UI.

Advantages entailing the use of regular-expressions include: elements may have multiple class attributes. The regex respects class names delimited by space (regex is enclosed within a "boundary" operator, "_). Furthermore, all the magic tweaking inherent to regular expressions apply; partial names or wild-cards are ok.
There seems to be a considerable performance gain if the *subtree* is the immediate parent of the target element.

4.3.4 XSL Transformation

In this section, we will try to outline a non-trivial, multi-layered strategy to use XSLT to generate SQL code, reaping the benefit of combining XML, XSL [ea01] and SQL to get a more robust solution to the problem specified in 3.6.

4.3.4.1 XSLT's processing model

The core functionality of XSLT's processing model are the `templates`: XPath expressions (rules) that are triggered whenever a particular part of the source document is being processed. When there is match, the body of these templates contain directives that tell the XSLT processor how the content is formatted. From the specification:

> A node matches a pattern if the node is a member of the result of evaluating the pattern as an expression with respect to some possible context; the possible contexts are those whose context node is the node being matched or one of its ancestors.

A tool that applies XSLT stylesheets to XML documents to produce some output is called `XSLT Processor`. In this application PHP5's XSL extension that implements the XSL standard, is used to perform transformations (as many other languages, PHP uses the libxslt library, also part of the GNOME XML Project). LibXML's command line processing tool 'xsltproc', can also be used.

As users are supposed to both *export* and *import* documents from/into the system, the substitutability of the stylesheet documents can be exploited to serve one XML document in different formats using different stylesheets.

4.3.4.2 XML To SQL

The data model in XML has parent-child relationship that is not naturally represented in relational database tables. To circumvent this architectural difference, the SQL INSERT statements generated in this step need further processing to replace template variables, like #cfpID#, that specifies primary keys that are generated from other SQL statements. The template rules consist of literal fragments (the SQL) and computations (name space 'xsl') An example would suffice:

```
<xsl:template match="//CFP/info">
  <xsl:for-each select="../info">
      INSERT INTO CFPEntryPoint(acronym) VALUES ('<xsl:value-of
           select="acronym"/>');
      INSERT INTO CFPHeader(cfpID,title, language, contactMail,
           mainURL)
      VALUES ('#cfpID#', '<xsl:value-of select="title"/>', '<xsl:
           value-of select="language"/>'..
```

A unique primary key is appointed to the entry of the first INSERT statement, this key is then used to replace all subsequent foreign keys (`#cfpID#`)that reference that entry. As a readily executable SQL is expected, delimiters and terminators (‘,’,´;’) are added in XSLT.
The following template matches the `importantDates` data type and uses conditional processing, native functions and variables to calculate positions, concatenate the date and delimiter or terminate the INSERT, which ever applicable. Notice the proper use of HTML entities within the condition (&lt;), this, one should in mind, is still XML.

```
<xsl:template match="//CFP/importantDates">
  <xsl:for-each select=".">
      INSERT INTO CFPImportantdates(cfpID, submisionOfAbstract,
           submisionOfPaper, notification, cameraReady,eventStart,
           eventEnd)VALUES('#cfpID#',
    <xsl:variable name="dateNodes"><xsl:value-of select="count(./
         date)"/></xsl:variable>
    <xsl:for-each select="./date">
      <xsl:variable name="date"><xsl:value-of select="position()"
           /></xsl:variable>
      <xsl:variable name="dateType"><xsl:value-of select="@type"/></
           xsl:variable>
      <xsl:variable name="datePos"><xsl:value-of select="position()"
           /></xsl:variable>
      '<xsl:value-of select="concat(./day,'-',./month,'-',./year)"/>
           '
      <xsl:choose>
        <xsl:when test="$datePos &lt; $dateNodes ">
          <xsl:text>, </xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>); </xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
```

A collection of template rules - one for each database table - that produce all the necessary statements are applied. The XSLT stylesheet can be found in it’s entirety in appendix A.3.

4.3.4.3 Less Naivety, More Security

Despite the possibility that this application will most likely be an in-house tool, used by trusted users, there is an urgent need to clean user-submitted code before any execution. An impersonator or even rogue, trusted user can do damage by embedding malicious code in the xml document: particularly, SQL injections are distressing.

An architectural enhancement to the PHP5 parser enables calls to **any** function defined in PHP as a native XSLT-Function. This introduces incredible maintainability and handling boost: we can serve two versions of the same XSL document, one which has PHP functions embedded, while the other document does not contain PHP-code, and thus doesn't assume coupling with PHP and can used with any other XSL processor.

The former stylesheet puts emphasis on input sanitation and filtering. Some ad-hoc checks and parsing are also done on the server-side, an example:

```
  <xsl:value-of select="php:functionString('parseXMLDate', concat(./day
      ,'-',./month,'-',./year))"/>
```

The `value-of` instruction evaluates the XPath expression given by the select attribute and converts the result into a string that is passed to the function *parseXMLDate* and the result , a 10 digit unix timestamp is returned.

```
function parseXMLDate($str){
  try{
    return String::parseDate($str);
  }
  catch(Exception $e){
    return "";
  }
}
```

This function tries to parse a given string into a date object, delegating the input through a static call to a function defined in the "String" class, which is part of the application. Notice how the function gracefully returns an empty string on failure, instead of halting the process (the user can always correct this mishap on a later occasion). To get this xml-php binding to work, a *namespace* (here, *php*) with qualified URI is appended to the processing instruction:

```
xmlns:php="http://php.net/xsl" xsl:extension-element-prefixes="php"
```

At last, to make the processor aware of this embedded code, functions calls must be registered at compile-time:

```
  $xslt = new xsltprocesser();
  $xslt->registerPhpFunctions();
```

The parsing engine recognizes the namespace and executes these functions.

4.3.5 Summary

XSLT, being a Turing complete language [Kep04], can be used to solve document-to-document mapping among other things[6]. Albeit the declarative style of

[6] I have seen examples on solving the Towers of Hanoi problem, and creation of self-similar sets like the fractal Sierpinski triangle (involved recursive template rules.)

programming in XSLT being a bit complicated and with peculiar characteristics, XML's intrinsic promise of 'extensibility' has proven fruitful with this multi-faceted code-generation. Relative to the other transformation methods discussed earlier, this method can be used with reasonable speed and efficiency.

When your only tool is a hammer, everything looks like a nail. When your only (fundamental) tool is schema, widespread adoption and large inventory of supporting tools are significant.

4.4 Conclusion

The implementation presented here is adaptable, sufficiently extensive and powerful enough to accommodate the outlined requirements, but is not complete and may need some extensions.

Is this document, a multi-faceted use of XML in a document management tool has been discussed: XML as a delivery format between different interfaces, as a storage capacity (CFPs stored as XML on the server) and XML as a code-generating tool: transforming XML into a formats that serves another needs.

The proposed solution has been through iterative revisions, where design was created, evaluated, and refined until the desired consistency, functionality and performance was achieved. Efficiency is not easily demonstrable, and it can be perceived differently by different users. In this application, user satisfaction has been the key usability attribute. The proposed prototype closely imitates the final product, and has lots of detail and functionality that can help envision the capacity and extending functionalities that lie within reach. Some strong conclusions about how behavior will relate to use of the final product can now be made.

4.4.1 Contributions

The proposed solutions solves a domain-specific problems that (known of) alternative tools cannot do as well, if at all. The contributions made include, but are not limited to:

* A heterogeneous system that handles incoming Call For Paper documents. Currently, email repository and management is powered by a high performance and reliable third party, Google. The harvesting and validation process can be made entirely autonomous. Due to this autonomy, there is greater scope for extensions within the application.
* Functionally and aesthetically appealing web-interface with :
  - Consistent, schema-aware document handling: export and import of valid documents through various means.
  - A programmable, extendible application components accessible via standard web protocols
  - User-centric web functionalities with enriching end-user value,

  - Data visualization, some realized using existing semantic technologies to provide seamless interaction,
  - Unobtrusive inline editing with a simple interface.

* A Java client that can used in conjunction with the authoring of documents in that XML dialect.

4.5 Further Work

According to Confucius,a Chinese philosopher, "*A journey of a thousand miles begins with a single step*".

Although the current application includes many of the in-depth functionalities required by document management system, it has some caveats and limitation; more steps are necessary as there are extension and enhancement areas worth calling out specifically. In the following sections, a quick pass of some of these enhancements are made.

4.5.1 Possible Extensions

4.5.1.1 Event Subscription

As for most web-centric applications, extensibility is a corner stone. If this tool could be extended to not only include Call For Papers submitted, but also incorporate conferences that are submitted to other larger or significant CFP listings. Given a list of keywords that matched a users interest, one approach could involve fetching data for conferences that had resemblance or shared some criteria (say *topic* or *keywords*) with one or more items from that list. The target of this pattern-matching could be large(r) CFP directories. Some *reminder* mechanisms could then be employed to forewarn the user about any events scheduled.

An approach that is similar to *wikicpf.com* (section 0.1.2) where users can subscribe to RSS feeds with respect to categories, is also within the area of implementational feasibility.

4.5.1.2 Classification System

The above extension could be merged with a classification system that can use some characterization, e.g keywords, to systematicly group and organize subjects by area. The hierarchical classification scheme *ACM Computing Classification System*, devised by the Association for Computing Machinery is a very extensive directory that contains most subjects dealt with in the computing literature. Other classifications, such as *Mathematics Subject Classification* could also be incorporated if other scientific fields should come into focus.

4.5.1.3 Email Verification Token

Since the email-address is used an identifying mechanism, a more robust approach is needed to verify the authenticity of the ownership.
Within the bounds of what we may consider a feasible extension is to extend the harvesting subsystem with a module that would dispatch an email to the sender, as to verify the identify. This verification could involve a link that, once followed, automatically processes any documents that were put on hold. If a non-authorized person has a permanent access to that account, then there is not much we can do.

Combined with a password input once the link is clicked, this would constitute a not-easily forgeable reference that can used ad a token of authority.

4.5.1.4 Extended Access Policy

Instead of restricting privileges on single-user basis, a broader access-policy could be enforced by allowing a set of people, such as an organizing committee to modify and delete certain CFPs.

4.5.1.5 Natural Language Search

This extension would seriously consider the approach discussed in section 3.5.0.2, and the limitations of the chosen database engine; many possibilities lie within there. Besides the natural language search discussed there, this extension may also study other methods that can be used to establish links between the contexts of distinct query terms, specifically Burkhard-Keller Trees.

4.5.2 Browser compatibility

In this narrow context and even narrower set of targeted audience, a specific web-client was assumed.
As a stepstone towards a cross-browser interface, issues regarding browser compatibility, or the lack of, should be addressed, (support for more browsers, possibly including — IE – could be achieved)

4.5.2.1 Schema Accessibility

As I have experienced, authoring schemas by hand is neither easy nor nor is it necessarily productive. It is more like testing the depth of the water with both feet; either you like it's notation, or you do not.

Some inline documentation and explanatory text should be added to the schema. I am given to understand that the XML Schema vocabulary contains a mechanism for this purpose: top level `annotation` element which serves the purpose:

```
<xs:annotation>
 <xs:documentation>some info about an element be here</xs:
     documentation>
</xs:annotation>
```

Instead of these vague, inline annotations that seem to be part of the schema specification, I imagine the usual comment blocks would be more readable: putting a `<!-- comment -->` atop of element(s) should be fine. These blocks, in contrast to the `annotation` element, are rendered differently in most basic editors and browsers alike (mostly colored green).

Furthermore, instead of having a static, pre-generated XML template, a more modular approach to transform a schema to a sample instance by generating code skeletons, could/should also be taken into consideration.

4.5.2.2 User Manual

Due to time constrains, authoring a user guide was not feasible. This extension is a high priority.

4.5.3 Evaluation

If there is single criterion used as measure to evaluate this project, it would be *usability*. Traceability of the implied and stated needs in the features implemented would also count. Of all the projects I have done at DTU, this has been the most rewarding one, for a few reasons;

* One reason being the extended lifetime such an application may be given,
* I had a peripheral knowledge about schemas, and this excessive use XSL is completely new to me. Having to author schemas by hand has been both time-consuming and challenging. later in the project, I've come to know some schema authoring tools (oXygen, Stylus), whose existence have eluded me during the schema development.

4.5.4 Ending Remarks

While developing an xml-based application is not a virtue in itself, having an XML dialect that conforms to a established, agreed-upon `standard` is. Even with XML, data interoperability is not matter of fact; XML is not a magic bullet in and by itself, one can still make proprietary software/format in XML. The power of XML lies within it's capacity of extensibility, openness, omnipresence and standardization: If other document management applications [7] or user-driven communities, like the ones mentioned in the introduction, came together and created a schema, whose content model was agreed upon, then that is the peak of *standardization*.

■

Technology standards emerge, evolved and inevitably become deprecated; standards collide on regular basis. These frictions and the inherent conflicts of interests, have, more often than not, spurred evolutionary steps. In the first New York Times article mentioning "XML" in its context, it was stated there was "*.. growing fear that Internet patents threaten to undermine the very nature of the online world*", and went on to describe XML as "*HTML on steroids.*". XML has since become a glowing celestial body in the information-exchange universe, making competing technologies mere asteroids.

[7] specific to Call For Papers, that is

```
If I have seen a little further it is by standing on the shoulders
of giants.
```

Appendix A

Related Documents

A.1 libXML Installation Guide

These are the few steps necessary to successfully compile libXML in your environment.
Fetch the source files:

```
$ wget ftp://xmlsoft.org/libxml2/libxml2-cvs-snapshot.tar.gz
$ tar xvzf libxml2-cvs-snapshot.tar.gz
```

To build on an Unixised setup:

```
\$ ./configure
\$ make
\$ make install
```

Errors raised during installation should be remedied accordingly. libXML will try to copy binaries to */usr/local/include/*, If you do not have root privileges, other directories must be specified as destination.

Full documentation is available on-line at http://xmlsoft.org/

A.2 XML Schema

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cfp:schema xmlns:cfp="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
   <xs:documentation>

   Based on and compliant with XML Schema 1.1.

   ********************
  Said Nuh(s040903)
  Sept. 18, 2007
  ********************



  In this document are the rules by which documents
  submitted to the system must abide.

  </xs:documentation>
 </xs:annotation>

 <xs:annotation>
  <xs:documentation>
    NS   (Name space): the current name space is "cfp",
         may/may not change during future revisions.

    base (as an attribute name): denotes an attribute whose value
        provides a URI to be used as the base for interpreting any
```

```
         relative URIs in the scope of the element on which it
         appears; its value is inherited.  This name is reserved
         by virtue of its definition in the XML Base specification.

    id   (as an attribute name): denotes an attribute whose value
         should be interpreted as if declared to be of type ID.
         This name is reserved by virtue of its definition in the
         xml:id specification.
  </xs:documentation>
 </xs:annotation>

  <cfp:simpleType name="stringtype">
    <cfp:restriction base="cfp:string"/>
  </cfp:simpleType>

  <cfp:simpleType name="CollapsedString">
    <cfp:restriction base="cfp:string">
      <cfp:whiteSpace value="collapse"/>
    </cfp:restriction>
  </cfp:simpleType>

  <cfp:simpleType name="timestampType">
    <cfp:restriction base="cfp:string">
      <cfp:pattern value="[0-9]{10}"/>
    </cfp:restriction>
  </cfp:simpleType>

  <cfp:simpleType name="eventIDType">
    <cfp:restriction base="cfp:string">
      <cfp:pattern value="[0-9]{10}"/>
    </cfp:restriction>
  </cfp:simpleType>

  <cfp:simpleType name="valid-URL">
    <cfp:restriction base="cfp:anyURI">
      <cfp:minLength value="1"/>
    </cfp:restriction>
  </cfp:simpleType>

  <cfp:simpleType name="non-empty-string">
    <cfp:restriction base="cfp:string">
      <cfp:whiteSpace value="preserve"/>
      <cfp:minLength value="1"/>
    </cfp:restriction>
  </cfp:simpleType>

  <cfp:simpleType name="valid-Email">
    <cfp:restriction base="cfp:token">
<!--  <cfp:pattern value="[^@]+@[^\.]+\..+"/> -->
      <cfp:pattern value="([\.a-zA-Z0-9_\-])+@([a-zA-Z0-9_\-])+(([a-zA
          -Z0-9_\-])*\.([a-zA-Z0-9_\-])+)+"/>
    </cfp:restriction>
  </cfp:simpleType>

  <cfp:simpleType name="importantDateNameType">
    <cfp:restriction base="cfp:string">
      <cfp:enumeration value="submisionOfAbstract"/>
      <cfp:enumeration value="submisionOfPaper"/>
      <cfp:enumeration value="notificationOfAcceptance"/>
      <cfp:enumeration value="cameraReady"/>
      <cfp:enumeration value="eventStart"/>
      <cfp:enumeration value="eventEnd"/>
    </cfp:restriction>
  </cfp:simpleType>

  <cfp:element name="importantDateType">
    <cfp:simpleType>
      <cfp:restriction base="cfp:string">
<!-- RegEx matches dd-mm-yyyy -->
        <cfp:pattern value="[0,1]{1}[0-9]{1}-[0-3]{1}[0-9]{1}-[0-9]{4}
            "/>
      </cfp:restriction>

    </cfp:simpleType>
  </cfp:element>

  <cfp:simpleType name="CommitteeNames">
    <cfp:restriction base="cfp:string">
      <cfp:enumeration value="progChair"/>
      <cfp:enumeration value="orgComm"/>
      <cfp:enumeration value="progComm"/>
      <cfp:enumeration value="steerComm"/>
    </cfp:restriction>
  </cfp:simpleType>

  <cfp:complexType name="eventInterval">
    <cfp:sequence>
      <cfp:element name="eventStart" type="non-empty-string" minOccurs
          ="0"/>
      <cfp:element name="eventEnd" type="non-empty-string" minOccurs="
          0"/>
    </cfp:sequence>
  </cfp:complexType>

  <cfp:complexType name="dateType">
    <cfp:sequence>
      <cfp:element name="day" type="cfp:nonNegativeInteger" minOccurs=
          "0" nillable="true"/>
      <cfp:element name="month" type="stringtype" minOccurs="0"
          nillable="true"/>
      <cfp:element name="year" type="cfp:nonNegativeInteger" minOccurs
          ="0" nillable="true"/>
    </cfp:sequence>
    <cfp:attribute name="type" type="importantDateNameType" use="
        required"/>
  </cfp:complexType>

  <cfp:complexType name="importantDateType">
    <cfp:sequence>
      <cfp:element name="date" type="dateType" maxOccurs="unbounded"/>
    </cfp:sequence>
  </cfp:complexType>


<!-- At least one topic must be specified -->
  <cfp:complexType name="topicsType_OLD">
    <cfp:sequence>
      <cfp:element name="topic" minOccurs="0">
        <cfp:complexType>
          <cfp:sequence>
            <cfp:element name="name" type="stringtype" maxOccurs="
                unbounded"/>
            <!-- sub- and subsub- topics are both optional-->
            <cfp:element name="subTopic" maxOccurs="unbounded"
                minOccurs="0">
              <cfp:complexType>
                <cfp:sequence>
                  <cfp:element name="name" type="stringtype"/>
                  <cfp:element name="subsubTopic" maxOccurs="unbounded
                      " minOccurs="0">
                    <cfp:complexType>
                      <cfp:sequence>
                        <cfp:element name="name" type="stringtype"
                            maxOccurs="unbounded"/>
                      </cfp:sequence>
                    </cfp:complexType>
                  </cfp:element>
                </cfp:sequence>
              </cfp:complexType>
            </cfp:element>
          </cfp:sequence>
        </cfp:complexType>
      </cfp:element>
    </cfp:sequence>
  </cfp:complexType>


  <cfp:complexType name="addressType">
```

```
    <cfp:sequence>
      <cfp:element name="street" type="stringtype"/>
      <cfp:element name="city" type="non-empty-string"/>
      <cfp:element name="country" type="non-empty-string"/>
    </cfp:sequence>
  </cfp:complexType>




  <cfp:complexType name="contactAddresType">
    <cfp:sequence>
      <cfp:element name="locationName" type="stringtype"/>
      <cfp:element name="address" type="addressType" minOccurs="1"
           nillable="true"/>
      <cfp:element name="email" type="valid-Email" nillable="true"/>
      <cfp:element name="fax" type="cfp:positiveInteger" nillable="
           true"/>
      <cfp:element name="telephone" type="cfp:positiveInteger"
           nillable="true"/>
    </cfp:sequence>
  </cfp:complexType>


  <cfp:complexType name="keywordType">
    <cfp:sequence>
      <cfp:element name="keyword" type="stringtype" maxOccurs="
           unbounded"/>
    </cfp:sequence>
  </cfp:complexType>


  <cfp:complexType name="CFPDetailsType">
    <cfp:sequence>
      <cfp:element name="scope" type="stringtype"/>
      <cfp:element name="topics" type="stringtype" minOccurs="1"/>
      <cfp:element name="specifications" type="stringtype"/>
      <cfp:element name="submissionGuidelines" type="stringtype"/>
      <cfp:element name="keywords" type="stringtype" maxOccurs="
           unbounded"/>
      <cfp:element name="location" type="contactAddresType"/>
    </cfp:sequence>
  </cfp:complexType>

  <cfp:complexType name="personType">
    <cfp:sequence>
      <cfp:element name="name" type="stringtype"/>
      <cfp:element name="institute" type="stringtype" minOccurs="1"/>
      <cfp:element name="url" type="stringtype" nillable="true"
           minOccurs="0"/>
      <cfp:element name="mail" type="stringtype" nillable="true"
           minOccurs="0"/>
    </cfp:sequence>
  </cfp:complexType>


  <!-- Derivation by extension: extend the person type with an extra
       field-->
  <cfp:complexType name="speakerType">
    <cfp:complexContent>
    <cfp:extension base="personType">
      <cfp:sequence>
      <cfp:element name="onTopic" type="stringtype" nillable="true"
           minOccurs="0"/>
      </cfp:sequence>
    </cfp:extension>
    </cfp:complexContent>
  </cfp:complexType>


  <cfp:complexType name="CommitteeType">
    <cfp:sequence>
  <!-- at least one Committee member required. or ? -->
```

```
      <cfp:element name="member" type="personType" minOccurs="0"
           maxOccurs="unbounded"/>
    </cfp:sequence>
    <cfp:attribute name="name" type="CommitteeNames" use="required"/>
  </cfp:complexType>

  <cfp:complexType name="CommitteesType">
    <cfp:sequence>
      <cfp:element name="committee" type="CommitteeType" minOccurs="0"
            maxOccurs="unbounded"/>
    </cfp:sequence>
  </cfp:complexType>


  <cfp:complexType name="speakersType">
    <cfp:sequence>
      <cfp:element name="speaker" type="speakerType" minOccurs="1"
           maxOccurs="unbounded"/>
    </cfp:sequence>
  </cfp:complexType>


  <cfp:complexType name="eventInfoType">
    <cfp:sequence>
      <cfp:element name="organizer" type="stringtype" nillable="true"
           />
      <cfp:element name="acronym" type="non-empty-string" nillable="
           false"/>
      <cfp:element name="title" type="non-empty-string"/>
      <cfp:element name="language" type="stringtype" nillable="true"/>
      <cfp:element name="mainEmail" type="valid-Email"/>
      <cfp:element name="mainURL" type="valid-URL"/>
    </cfp:sequence>
  </cfp:complexType>


  <cfp:complexType name="callForPaperType">
    <cfp:sequence>
      <cfp:element name="info" type="eventInfoType"/>
      <cfp:element name="importantDates" type="importantDateType"
           maxOccurs="unbounded"/>
      <cfp:element name="committees" type="CommitteesType" maxOccurs="
           unbounded"/>
      <cfp:element name="invitedSpeakers" type="speakersType"/>
      <cfp:element name="CFPDetails" type="CFPDetailsType"/>
      <cfp:element name="CFPData" type="CollapsedString"/>
    </cfp:sequence>
    <cfp:attribute name="eventID" type="eventIDType" use="required"/>
  </cfp:complexType>


  <cfp:complexType name="callForPaper">
    <cfp:sequence>
      <cfp:element name="CFP" type="callForPaperType" maxOccurs="1"
           minOccurs="1"/>
    </cfp:sequence>
  </cfp:complexType>


<!-- Allow multiple cfp's in one XML document -->
  <cfp:element name="callForPapers" type="callForPaper"/>
</cfp:schema>
```

A.3 XSL Stylesheet

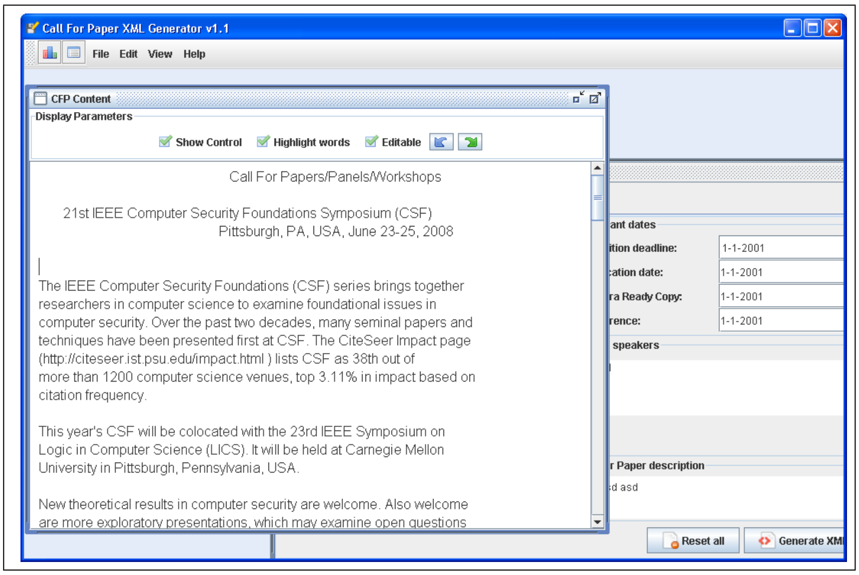
This version of the stylesheet contains no php-code that can filter user input. May include undesired characters or even malicious code, this version is used on browsers only.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
     Transform">
  <xsl:output method="text" encoding="UTF-8"/>
<!--
   Based on and compliant with XML Schema 1.1.

   ********************
  Said Nuh(s040903)
  Sept. 3, 2007
  ********************
-->
<xsl:template match="/">
-- SQL Queries
  <xsl:apply-templates select="//CFP/location/locationName"/>
  <xsl:apply-templates select="//CFP/info"/>
  <xsl:apply-templates select="//CFP/importantDates"/>
  <xsl:apply-templates select="//CFP/CFPDetails"/>
  <xsl:apply-templates select="//location/address"/>
  <xsl:apply-templates select="//CFP/CFPData"/>

  <xsl:apply-templates select="//CFP/committees"/>

<!--
  <xsl:template match="processing-instruction()">
    <xsl:copy/>
  </xsl:template>
-->

  <!-- VALUES ('xsl:value-of select="php:function('nl2br',string(text
       ()))"/>'); -->
  <!-- call-template name="rawData"/> -->
--
</xsl:template>

  <xsl:template match="//CFP/info">
    <xsl:for-each select="../info">
        INSERT INTO CFPEntryPoint(acronym) VALUES ('<xsl:value-of
             select="acronym"/>');

        INSERT INTO CFPHeader(cfpID,title, language, contactMail,
             mainURL)
        VALUES ('#ID#', '<xsl:value-of select="title"/>', '<xsl:value-
             of select="language"/>','<xsl:value-of select="mainEmail
             "/>', '<xsl:value-of select="mainURL"/>');
     </xsl:for-each>

  </xsl:template>

  <xsl:template match="//CFP/importantDates">
    <xsl:for-each select=".">
        INSERT INTO CFPImportantdates(cfpID, submisionOfAbstract,
             submisionOfPaper, notification, cameraReady,eventStart,
             eventEnd)VALUES('#ID#',
      <xsl:variable name="dateNodes"><xsl:value-of select="count(./
           date)"/></xsl:variable>
      <xsl:for-each select="./date">
        <xsl:variable name="date"><xsl:value-of select="position()"
             /></xsl:variable>
        <xsl:variable name="dateType"><xsl:value-of select="@type"/></
             xsl:variable>
        <xsl:variable name="datePos"><xsl:value-of select="position()"
             /></xsl:variable>
        '<xsl:value-of select="concat(./day,'-',./month,'-',./year)"/>
             '
        <xsl:choose>
          <xsl:when test="$datePos &lt; $dateNodes ">
            <xsl:text>, </xsl:text>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text>); </xsl:text>
          </xsl:otherwise>

        </xsl:choose>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="//CFP/CFPDetails">
     <xsl:for-each select=".">
      <xsl:variable name="scope"><xsl:value-of select="scope"/></xsl:
           variable>
      <xsl:variable name="topics"><xsl:value-of select="topics"/></xsl
           :variable>

      <xsl:variable name="specifications"><xsl:value-of select="
           specifications"/></xsl:variable>
      <xsl:variable name="guidelines"><xsl:value-of select="
           submissionGuidelines"/></xsl:variable>
      <xsl:variable name="keywords"><xsl:value-of select="keywords"
           /></xsl:variable>
        INSERT INTO CFPDetails(cfpID, scope, topics, specifications,
             guidelines, keywords)
        VALUES ('#ID#',
          '<xsl:value-of select="string($scope)"/>',
          '<xsl:value-of select="string($topics)"/>',
          '<xsl:value-of select="string($specifications)"/>',
          '<xsl:value-of select="string($guidelines)"/>',
          '<xsl:value-of select="string($keywords)"/>');
     </xsl:for-each>
  </xsl:template>

  <xsl:template match="//CFP/committees">
    <xsl:for-each select=".">
        <xsl:variable name="memberNodes"><xsl:value-of select="count
             (.//member)"/></xsl:variable>
      <xsl:for-each select="./committee">
        <xsl:variable name="roleName"><xsl:value-of select="@name"/></
             xsl:variable>
        <xsl:variable name="memberPos"><xsl:value-of select="position
             ()"/></xsl:variable>
        <xsl:for-each select="./member">
      INSERT INTO CFPPersona(name,email,url,institute)VALUES
        ('<xsl:value-of select="name"/>','<xsl:value-of select="mail
             "/>','<xsl:value-of select="url"/>','<xsl:value-of select
             ="institute"/>');|<xsl:value-of select="$roleName"/>

          <xsl:choose>
            <xsl:when test="$memberPos &lt; $memberNodes">
            </xsl:when>
            <xsl:otherwise>
              <xsl:text></xsl:text>
            </xsl:otherwise>
          </xsl:choose>

        </xsl:for-each>

      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>



   <xsl:template match="//location/address">
      <xsl:variable name="locationName"><xsl:apply-templates select="
           ../locationName"/></xsl:variable>
      INSERT INTO CFPAddress(cfpID, locationName, street, city,
           country)
      VALUES ('#ID#', '<xsl:value-of select="$locationName"/>', '<xsl:
           value-of select="street"/>', '<xsl:value-of select="city"/>'
           ,'<xsl:value-of select="country"/>');
   </xsl:template>


   <xsl:template match="//location/locationName">
     <xsl:value-of select="text()"/>
   </xsl:template>

   <xsl:template match="//CFP/CFPData" name="rawData">
      INSERT INTO CFPRawdata(cfpID, content)
      VALUES ('#ID#', '<xsl:value-of select="string(text())"/>');
   </xsl:template>

</xsl:stylesheet>
```

Appendix B

Screen dumps

This chapter contains few images that show parts of the application that I thought deserved more attention. Some are taken for the purpose of demonstration, others for documentation.



Call For Papers/Panels/Workshops

21st IEEE Computer Security Foundations Symposium (CSF)
Pittsburgh, PA, USA, June 23-25, 2008

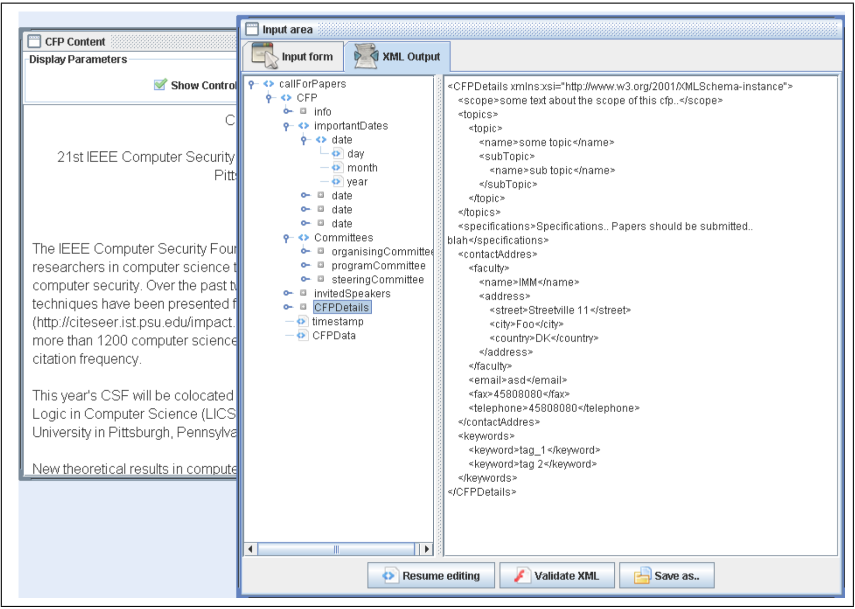
The IEEE Computer Security Foundations (CSF) series brings together researchers in computer science to examine foundational issues in computer security. Over the past two decades, many seminal papers and techniques have been presented first at CSF. The CiteSeer Impact page (http://citeseer.ist.psu.edu/impact.html) lists CSF as 38th out of more than 1200 computer science venues, top 3.11% in impact based on citation frequency.

This year's CSF will be colocated with the 23rd IEEE Symposium on Logic in Computer Science (LICS). It will be held at Carnegie Mellon University in Pittsburgh, Pennsylvania, USA.

New theoretical results in computer security are welcome. Also welcome are more exploratory presentations, which may examine open questions

ition deadline:	1-1-2001
cation date:	1-1-2001
ra Ready Copy:	1-1-2001
rence:	1-1-2001

Figure B.1: Java client - showing the content area , while the input fields are visible at the background.



```
C
21st IEEE Computer Security
Pitt

The IEEE Computer Security Four
researchers in computer science t
computer security. Over the past t
techniques have been presented f
(http://citeseer.ist.psu.edu/impact.
more than 1200 computer science
citation frequency.

This year's CSF will be colocated
Logic in Computer Science (LICS
University in Pittsburgh, Pennsylva

New theoretical results in compute
```

```
callForPapers
  CFP
    info
    importantDates
      date
        day
        month
        year
      date
      date
      date
    Committees
      organisingCommittee
      programCommittee
      steeringCommittee
    invitedSpeakers
    CFPDetails
  timestamp
  CFPData
```

```
<CFPDetails xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <scope>some text about the scope of this cfp..</scope>
  <topics>
    <topic>
      <name>some topic</name>
      <subTopic>
        <name>sub topic</name>
      </subTopic>
    </topic>
  </topics>
  <specifications>Specifications.. Papers should be submitted..
blah</specifications>
  <contactAddres>
    <faculty>
      <name>IMM</name>
      <address>
        <street>Streetville 11</street>
        <city>Foo</city>
        <country>DK</country>
      </address>
    </faculty>
    <email>asd</email>
    <fax>45808080</fax>
    <telephone>45808080</telephone>
  </contactAddres>
  <keywords>
    <keyword>tag_1</keyword>
    <keyword>tag 2</keyword>
  </keywords>
</CFPDetails>
```

Figure B.2: Java client: A generated document that is ready to be saved/validated. On the left: document tree structure that can used as a navigation tool.

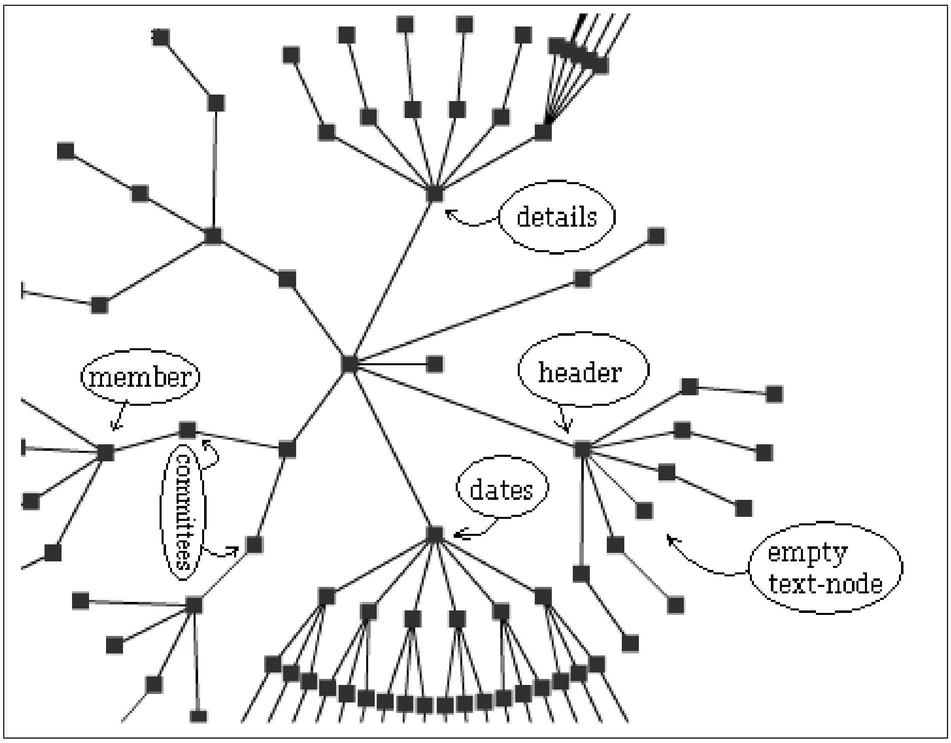


Figure B.3: A hierarchical view of a small template document with few optional elements

B.1 Web Form Design

B.2 Document Validator

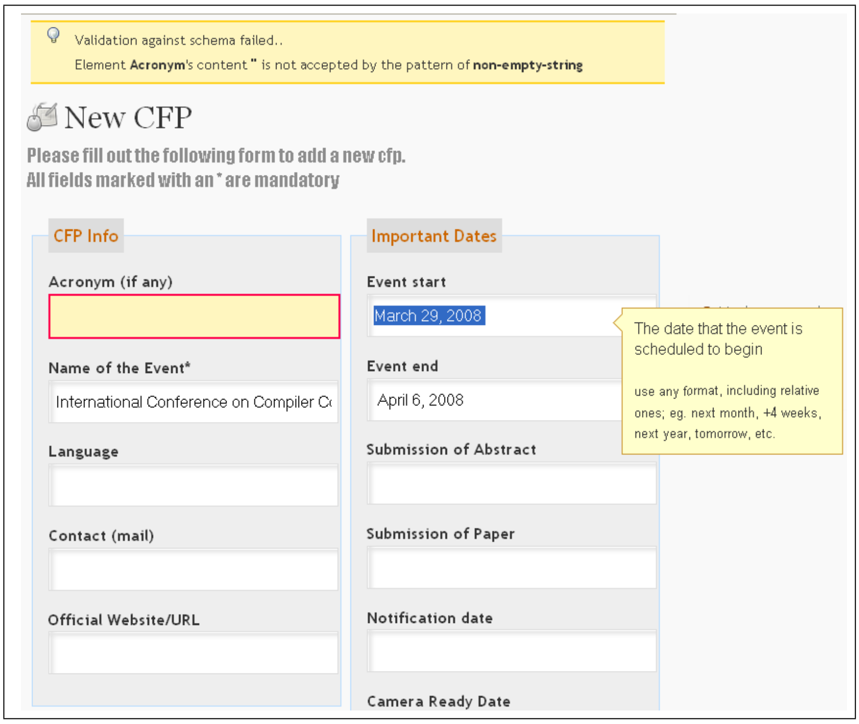
B.3 Geographical Locations

B.4 Keyword Density

B.5 Time Proximity (deadlines)

B.6 CFP Main Page

B.7 Content Recommendation



Validation against schema failed..
Element **Acronym**'s content '' is not accepted by the pattern of **non-empty-string**

New CFP

Please fill out the following form to add a new cfp.
All fields marked with an * are mandatory

CFP Info

Acronym (if any)

Name of the Event*
International Conference on Compiler C

Language

Contact (mail)

Official Website/URL

Important Dates

Event start
March 29, 2008

The date that the event is scheduled to begin

use any format, including relative ones; eg. next month, +4 weeks, next year, tomorrow, etc.

Event end
April 6, 2008

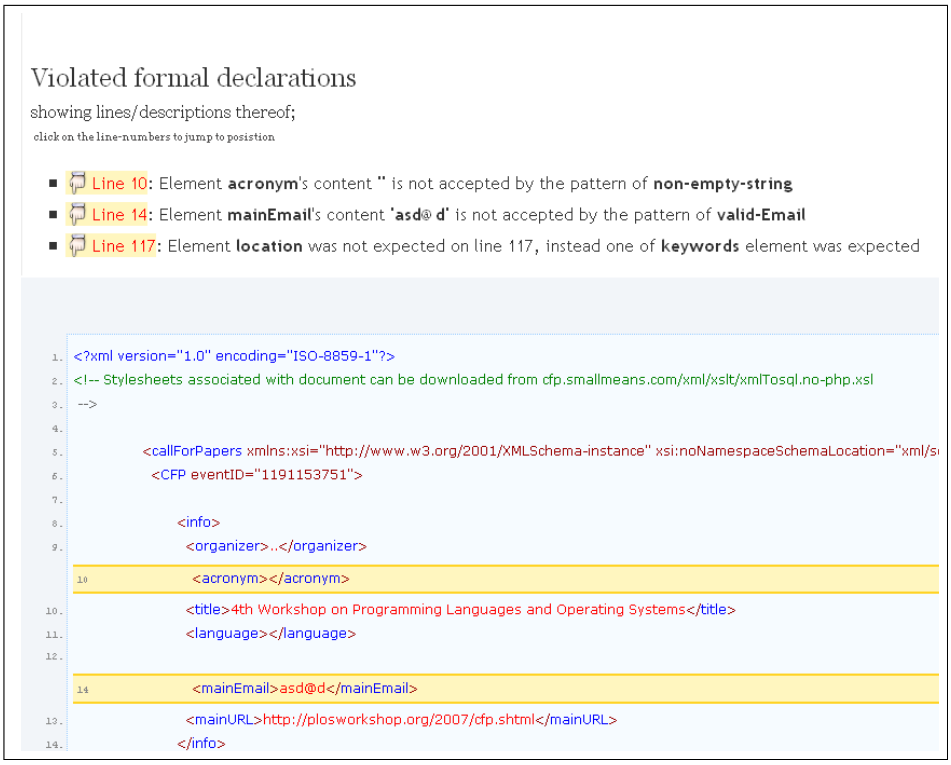
Submission of Abstract

Submission of Paper

Notification date

Camera Ready Date

Figure B.4: Submission form: schema errors, invalid-input field and form-hints all visible



Violated formal declarations

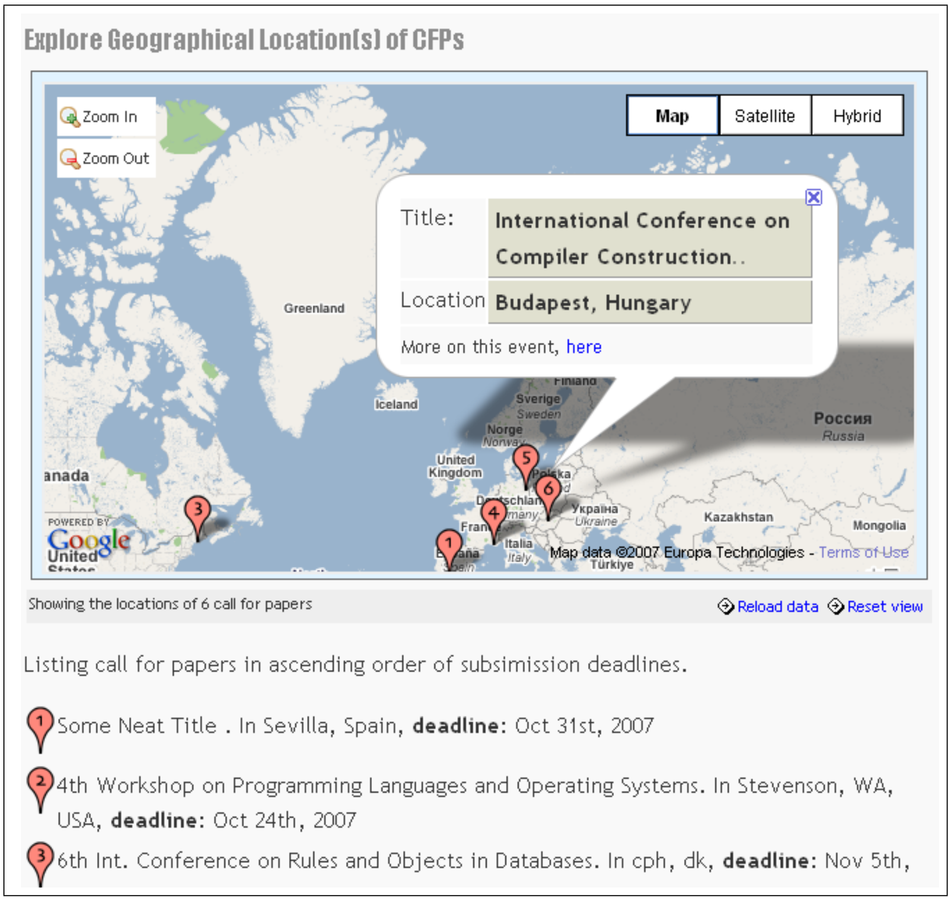
showing lines/descriptions thereof;

click on the line-numbers to jump to posistion

- Line 10: Element **acronym**'s content '' is not accepted by the pattern of **non-empty-string**
- Line 14: Element **mainEmail**'s content **'asd@ d'** is not accepted by the pattern of **valid-Email**
- Line 117: Element **location** was not expected on line 117, instead one of **keywords** element was expected

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <!-- Stylesheets associated with document can be downloaded from cfp.smallmeans.com/xml/xslt/xmlTosql.no-php.xsl
3.   -->
4.
5.          <callForPapers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="xml/s
6.           <CFP eventID="1191153751">
7.
8.              <info>
9.               <organizer>..</organizer>
 10               <acronym></acronym>
10.              <title>4th Workshop on Programming Languages and Operating Systems</title>
11.              <language></language>
12.
 14               <mainEmail>asd@d</mainEmail>
13.              <mainURL>http://plosworkshop.org/2007/cfp.shtml</mainURL>
14.             </info>
```

Figure B.5: Showing a traces of violated rules in document


Explore Geographical Location(s) of CFPs

Title: International Conference on Compiler Construction..

Location Budapest, Hungary

More on this event, here

Showing the locations of 6 call for papers

Reload data Reset view

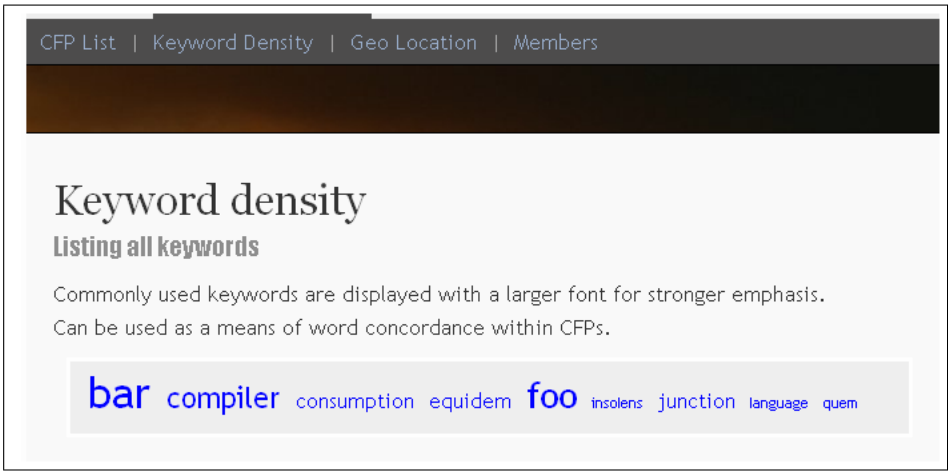
Listing call for papers in ascending order of subsimission deadlines.

1 Some Neat Title . In Sevilla, Spain, **deadline**: Oct 31st, 2007

2 4th Workshop on Programming Languages and Operating Systems. In Stevenson, WA, USA, **deadline**: Oct 24th, 2007

3 6th Int. Conference on Rules and Objects in Databases. In cph, dk, **deadline**: Nov 5th,

Figure B.6: Showing geographical location alongside textual description



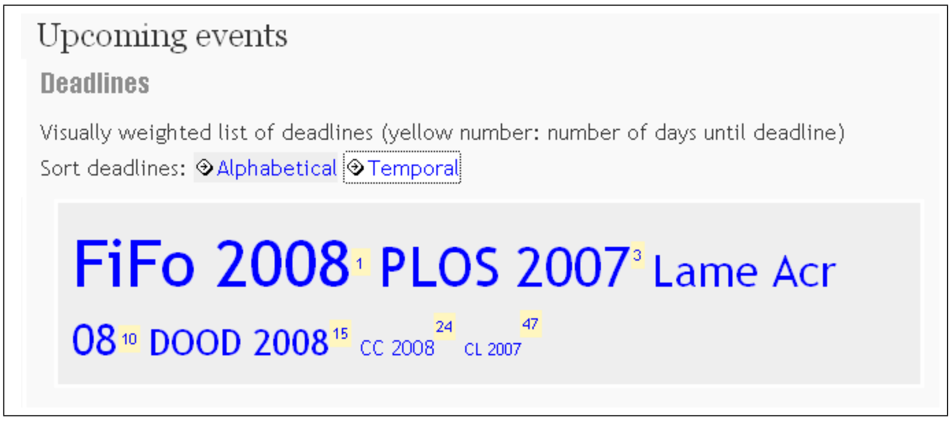
Keyword density

Listing all keywords

Commonly used keywords are displayed with a larger font for stronger emphasis.
Can be used as a means of word concordance within CFPs.

bar compiler consumption equidem foo insolens junction language quem

Figure B.7: Showing weighted list of keywords


Upcoming events

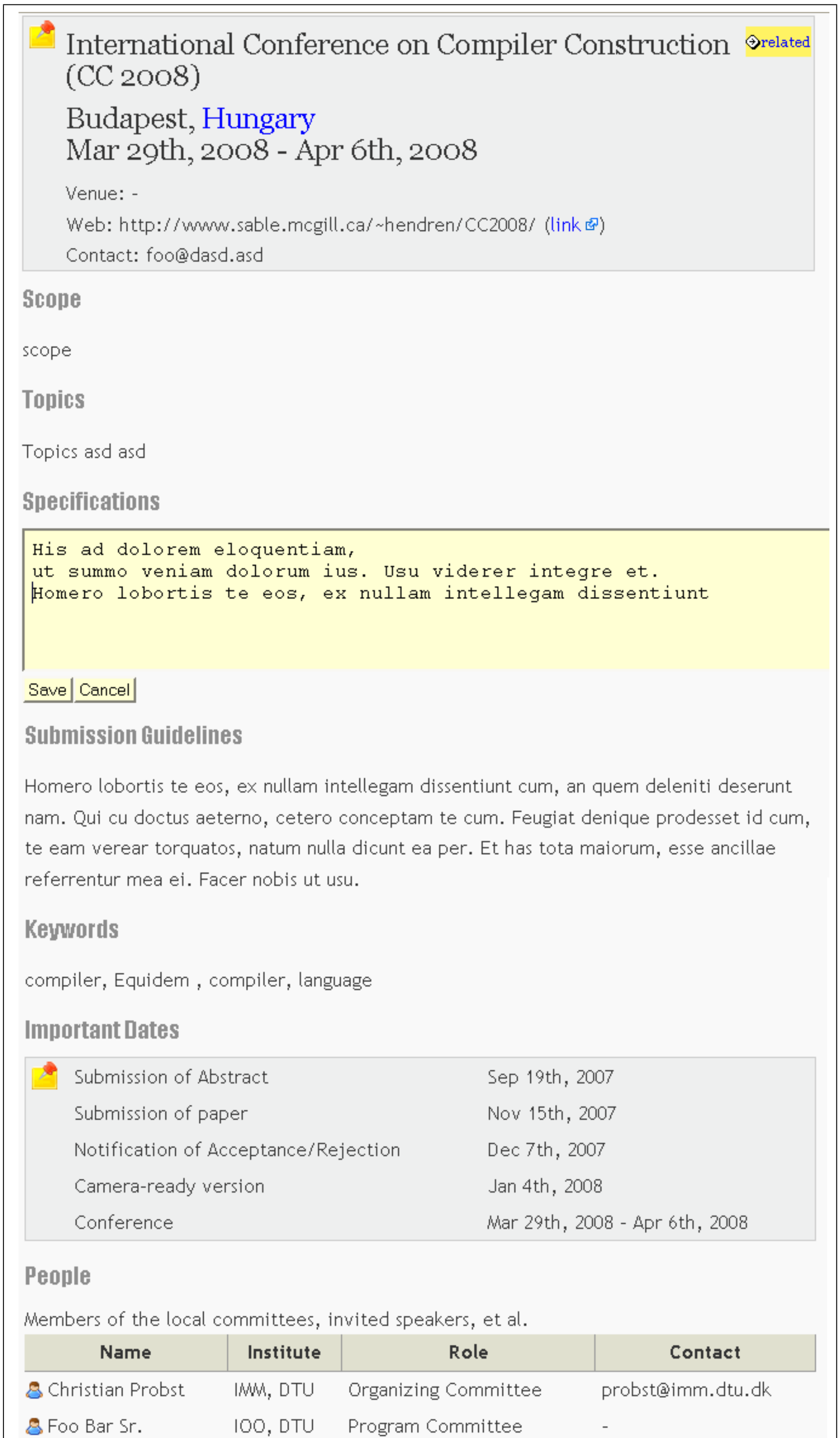
Deadlines

Visually weighted list of deadlines (yellow number: number of days until deadline)

Sort deadlines: Alphabetical Temporal

FiFo 2008 1 PLOS 2007 3 Lame Acr 08 10 DOOD 2008 15 CC 2008 24 CL 2007 47

Figure B.8: Showing size of conference as a function time proximity



International Conference on Compiler Construction (CC 2008)

related

Budapest, Hungary
Mar 29th, 2008 - Apr 6th, 2008

Venue: -

Web: http://www.sable.mcgill.ca/~hendren/CC2008/ (link)

Contact: foo@dasd.asd

Scope

scope

Topics

Topics asd asd

Specifications

```
His ad dolorem eloquentiam,
ut summo veniam dolorum ius. Usu viderer integre et.
Homero lobortis te eos, ex nullam intellegam dissentiunt
```

Save Cancel

Submission Guidelines

Homero lobortis te eos, ex nullam intellegam dissentiunt cum, an quem deleniti deserunt nam. Qui cu doctus aeterno, cetero conceptam te cum. Feugiat denique prodesset id cum, te eam verear torquatos, natum nulla dicunt ea per. Et has tota maiorum, esse ancillae referrentur mea ei. Facer nobis ut usu.

Keywords

compiler, Equidem , compiler, language

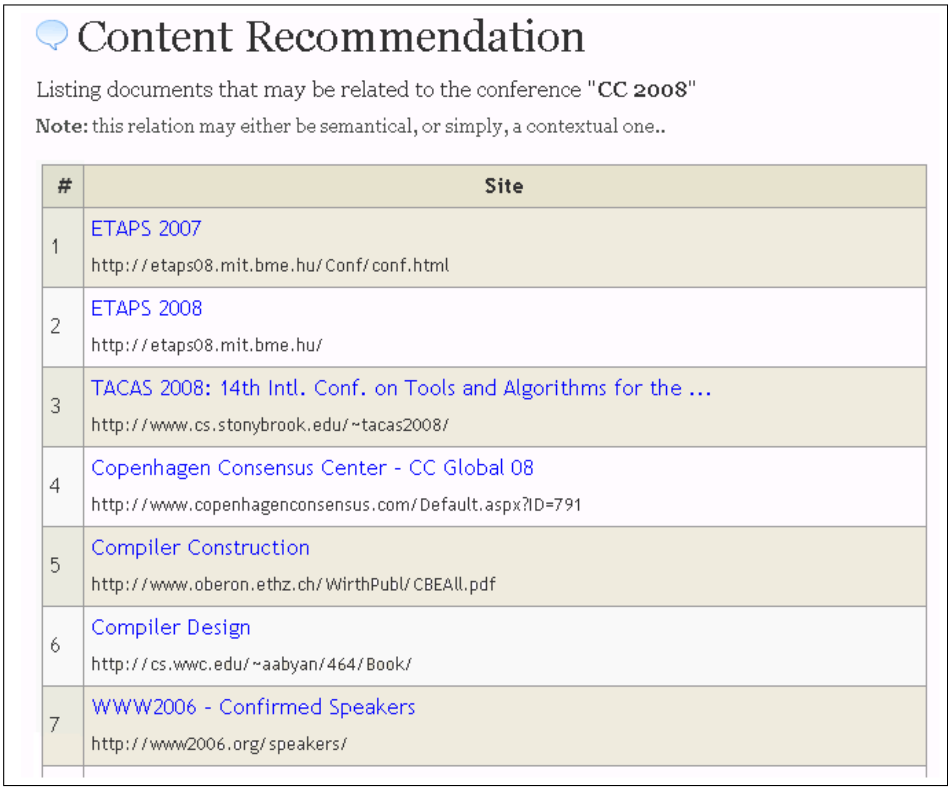
Important Dates

Submission of Abstract	Sep 19th, 2007
Submission of paper	Nov 15th, 2007
Notification of Acceptance/Rejection	Dec 7th, 2007
Camera-ready version	Jan 4th, 2008
Conference	Mar 29th, 2008 - Apr 6th, 2008

People

Members of the local committees, invited speakers, et al.

Name	Institute	Role	Contact
Christian Probst	IMM, DTU	Organizing Committee	probst@imm.dtu.dk
Foo Bar Sr.	IOO, DTU	Program Committee	-



Content Recommendation

Listing documents that may be related to the conference "**CC 2008**"

Note: this relation may either be semantical, or simply, a contextual one..

#	Site
1	ETAPS 2007 http://etaps08.mit.bme.hu/Conf/conf.html
2	ETAPS 2008 http://etaps08.mit.bme.hu/
3	TACAS 2008: 14th Intl. Conf. on Tools and Algorithms for the ... http://www.cs.stonybrook.edu/~tacas2008/
4	Copenhagen Consensus Center - CC Global 08 http://www.copenhagenconsensus.com/Default.aspx?ID=791
5	Compiler Construction http://www.oberon.ethz.ch/WirthPubl/CBEAll.pdf
6	Compiler Design http://cs.wwc.edu/~aabyan/464/Book/
7	WWW2006 - Confirmed Speakers http://www2006.org/speakers/

Figure B.10: Showing documents that are related to a conference

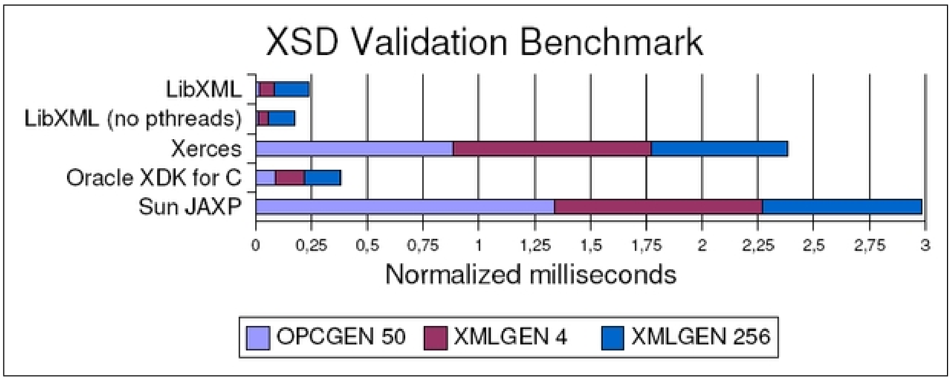


Figure B.11: Schema validation benchmark

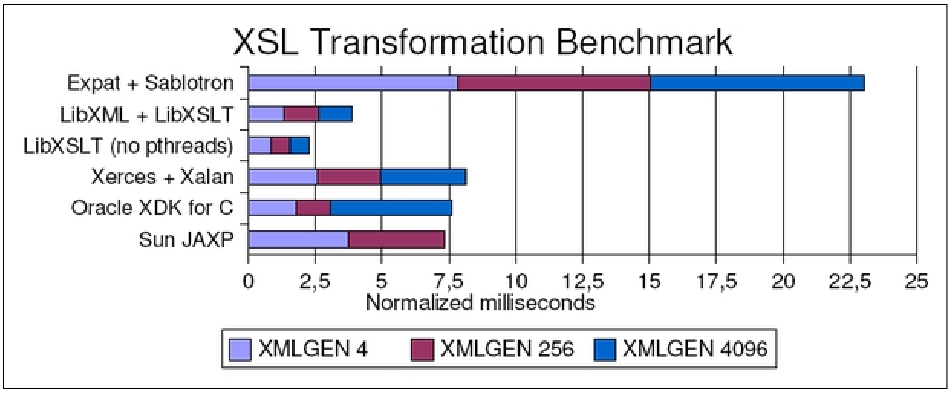


Figure B.12: XSL transformation benchmark

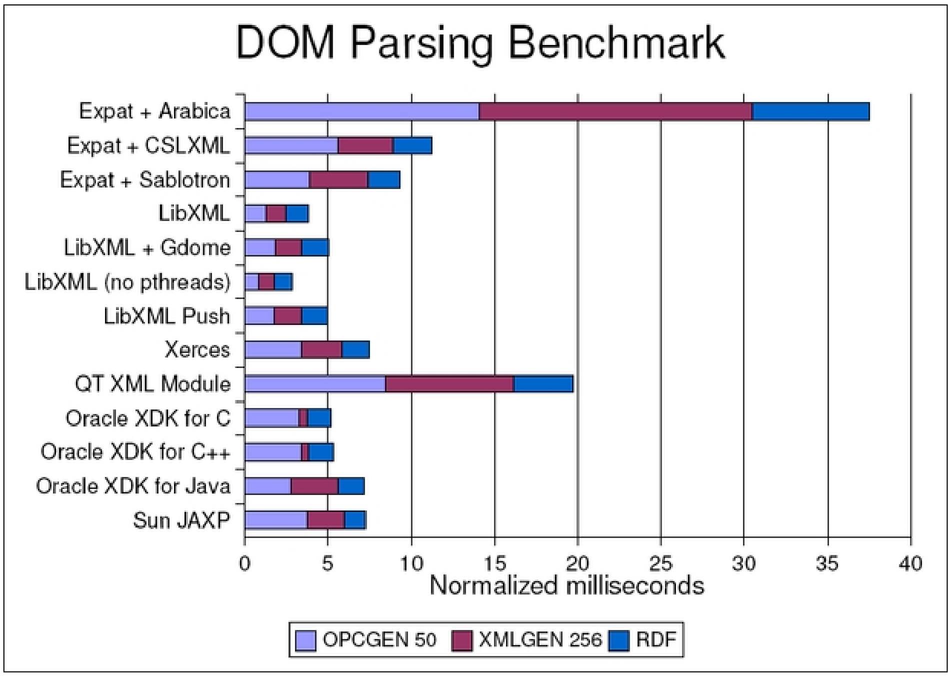


Figure B.13: DOM parsing benchmark

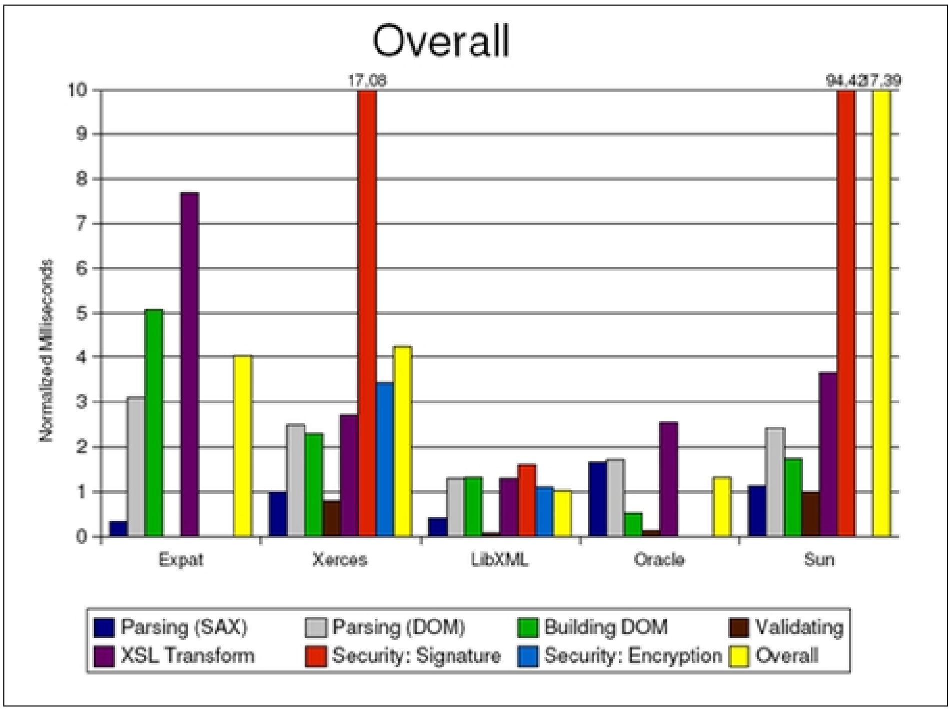
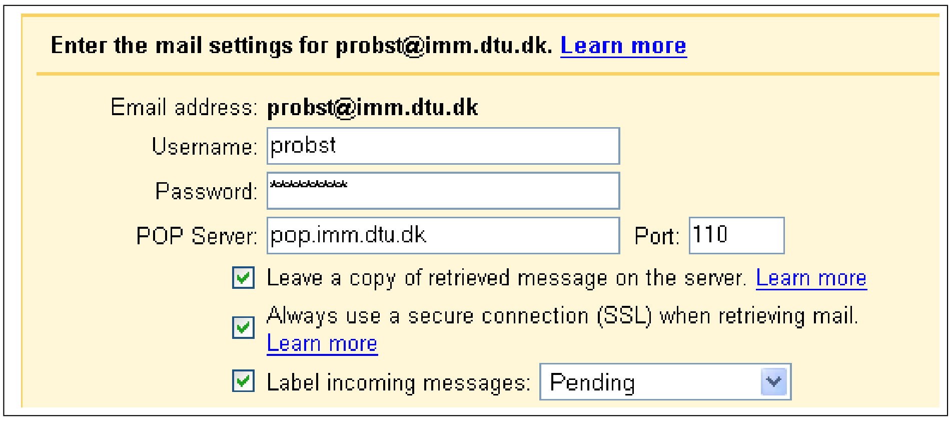


Figure B.14: Overall performance benchmark


Enter the mail settings for probst@imm.dtu.dk. Learn more

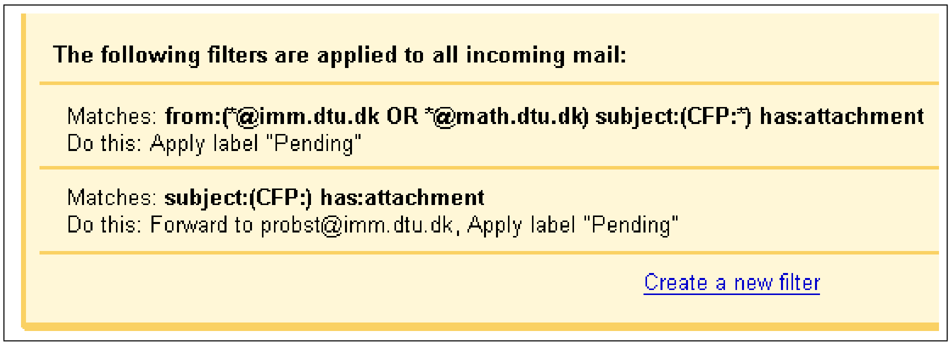
Email address:	**probst@imm.dtu.dk**
Username:	probst
Password:	**********
POP Server:	pop.imm.dtu.dk Port: 110

✓ Leave a copy of retrieved message on the server. Learn more

✓ Always use a secure connection (SSL) when retrieving mail. Learn more

✓ Label incoming messages: Pending

Figure B.15: Setting up POP3 to retrieve emails from a remote server



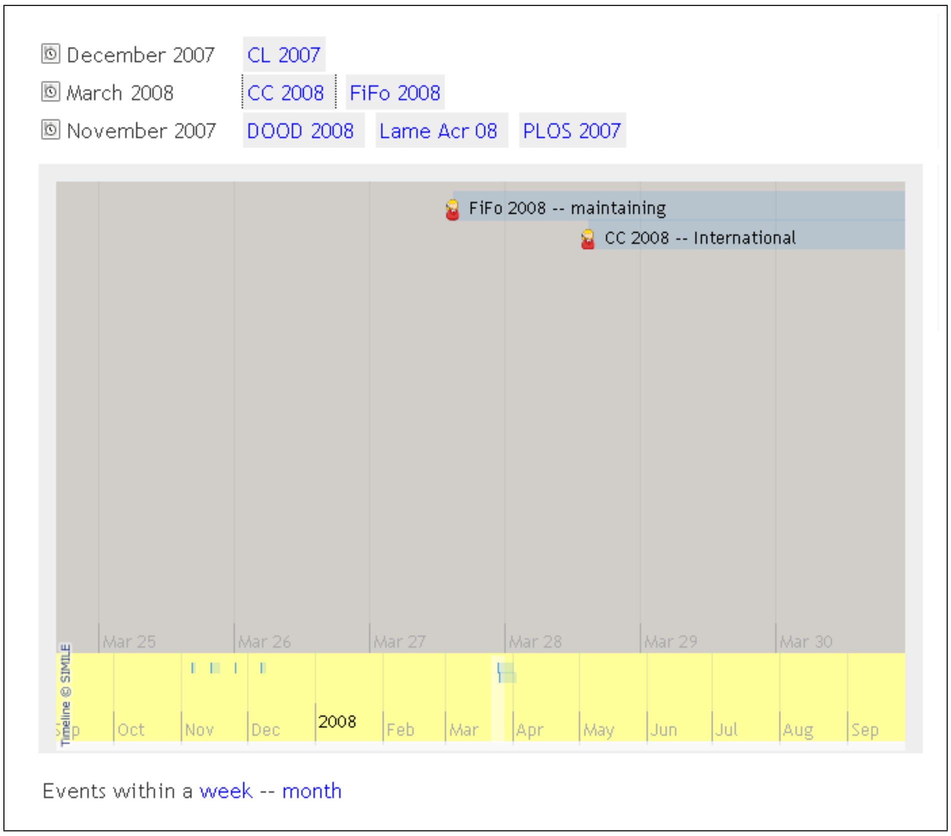
The following filters are applied to all incoming mail:

Matches: **from:(*@imm.dtu.dk OR *@math.dtu.dk) subject:(CFP:*) has:attachment**
Do this: Apply label "Pending"

Matches: **subject:(CFP:) has:attachment**
Do this: Forward to probst@imm.dtu.dk, Apply label "Pending"

Create a new filter

Figure B.16: Setting up filters to organize and prepare email for harvesting



December 2007 CL 2007
March 2008 CC 2008 FiFo 2008
November 2007 DOOD 2008 Lame Acr 08 PLOS 2007

Events within a week -- month

Figure B.17: Timeline: Linear representation of conference dates

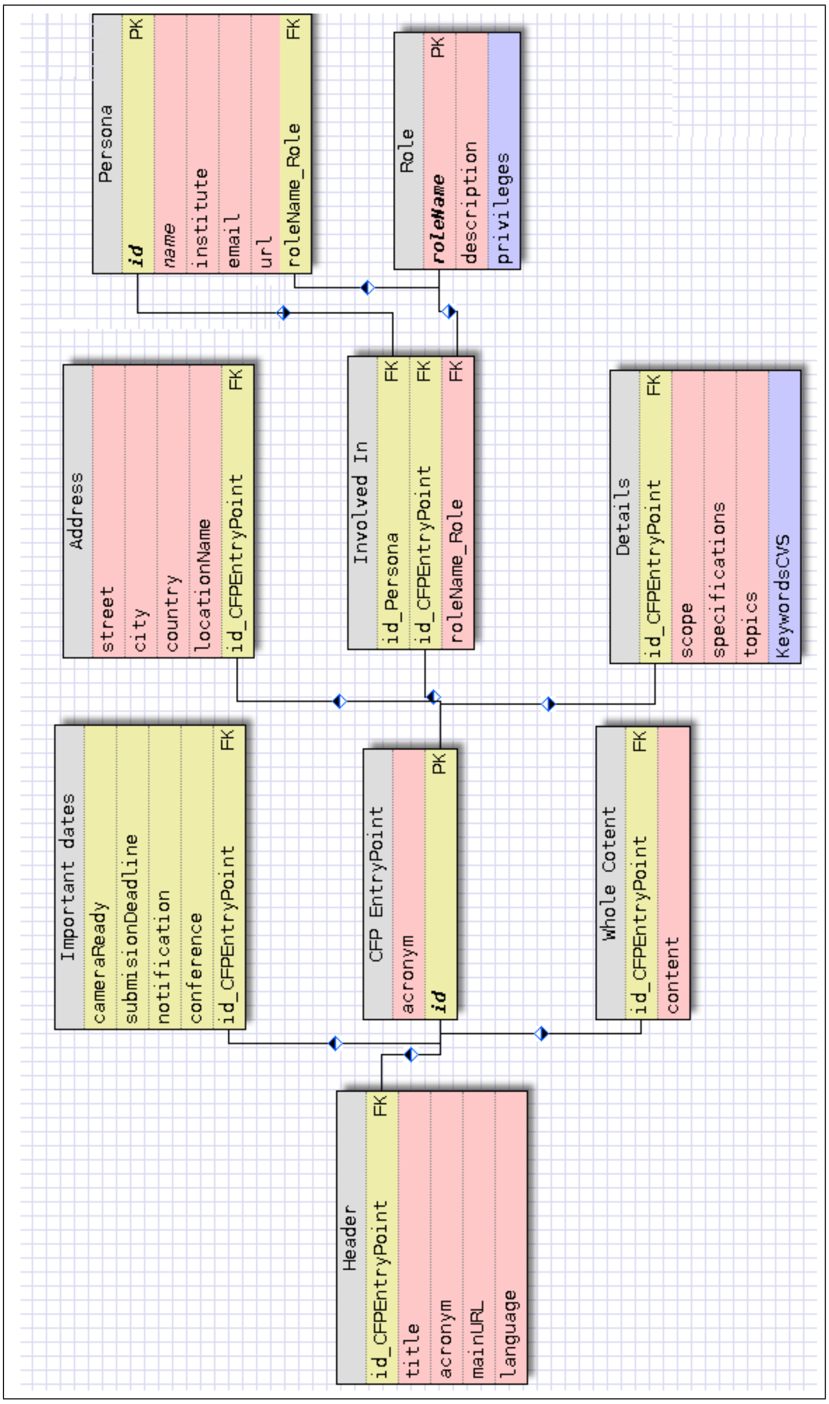


Figure B.18: Database tables and their relations, FK=Foreing Key, PK=Primary Key (table names simplified)

Bibliography

[Bjø05] Dines Bjørner. *vol. 3: Domains, Requirements and Software Design of Texts in Theoretical Computer Science, the EATCS Series*. Springer-Verlag, 2005.

[Bra97] S. Bradner. Key words for use in rfcs to indicate requirement levels. Internet, 1997. http://www.ietf.org/rfc/rfc2119.txt.

[Com07] Gilleron Jacquemard Lugiez Tison Comon, Dauchet. Tree automata techniques and applications. Internet, 2007. http://tata.gforge.inria.fr/.

[ea01] Sharon Adler et al. Extensible stylesheet language (xsl) version 1.0. Internet, 2001. http://www.w3.org/TR/2001/REC-xsl-20011015/.

[HSTM04] M. Maloney H. S. Thompson, D. Beech and N. Mendelsohn. Xml schema part 1: Structures. Internet, 2004. http://www.w3.org/XML/Schema.

[IEE98] IEEE. Ieee recommended practice for software requirements specifications. Internet, 1998. http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel4/5841/15571/0072057

[IH07] Inc. Ian Hickson, Google. Xml binding language (xbl) 2.0. Internet, 2007. http://www.w3.org/TR/xbl/.

[Kep04] Stephan Kepser. A simple proof for the turing-completeness of xslt and xquery. Internet, 2004. http://www.idealliance.org/papers/extreme/proceedings//html/2004/Kepser0

[McL03] Brett McLaughlin. Output large xml documents. Internet, 2003. http://www.ibm.com/developerworks/xml/library/x-tipbigdoc.html.

[Pro] A tool for web-based management of call-for-papers. Internet. http://www2.imm.dtu.dk/courses/02125/0019/.

[W3P] World wide web consortium process document. Internet. http://www.w3.org/Consortium/Process/.

[XML06] Extensible markup language (xml) 1.0 (fourth edition). Internet, 2006. http://www.w3.org/TR/xml/.