
University of Innsbruck
Digital Enterprise Research Institute

# Thunderbird extension for Semantic eMail Addressing

Bachelor thesis

**Philipp Reisinger**
Botanikerstrasse 17/7
Innsbruck, Austria
{firstname.lastname}@student.uibk.ac.at
MatrNr.: 0315873

Supervised by:

Univ. Prof. Dr. Thomas Strang
University of Innsbruck
Innsbruck, Austria

Co-supervised by:

Uwe Keller
University of Innsbruck
Innsbruck, Austria

**Innsbruck, January 2007**

**Abstract**

Almost everyone is using eMail as a tool for communication and collaboration each day. Especially in a business setting eMail became a major means for information exchange. However, for efficient communication across highly dynamic groups of people (e.g. working groups, employees of a specific department, people with common interests) current tools are non-optimal. Today, mailing lists are used to email predefined groups. Since email addresses and mailing list often have no semantics, discovering 'the right' list can be a difficult task. As there are infinitely many ways to define a set of people one cannot in general rely on such predefined lists. Instead, one must have the ability to address emails to arbitrary groups of people. Semantic eMail Addressing may act as a possible solution to the outlined problem. Besides a general introduction of Semantic eMail this thesis mainly focuses on the development of an extension for Mozilla Thunderbird (TB) [10]. The extension enables TB to specify and resolve ontology based Semantic eMail Addresses.

# Contents

# 1 Introduction

Based on [17] this chapter provides a general introduction of Semantic eMail and its benefits.

## 1.1 Semantic eMail Addressing in general

Like telephone numbers, eMail addresses, are opaque identifiers. They are often hard for a person to remember, and worse still, people's email addresses and phone numbers change from time to time. Generally spoken, the idea of Semantic eMail Addressing is to address persons with descriptive attributes instead of eMail Addresses represented by strings. One could send a message to a person just by entering his name, his position, or some other descriptive attributes. If the email address of a person changes, then the email system should send to the new email address, automatically. If the person matching a description differs over time, the email system should always send to the person currently matching that description. One should also be able to send emails to groups of people matching a particular set of attributes e.g. all people currently working for the SAP rollout project, all people working for any project leaded by Mr. John, all people interested in accounting and do not work for any project yet. Today, mailing lists are used to email predefined groups of people. However, as there are infinitely many ways to define a set of people like the once mentioned above, one cannot in general rely on such predefined lists. Semantic email addressing enables emails to be addressed to a semantically defined set of entities. The recipients of a semantically addressed email are computed on the fly based on the semantic definition of the address.

The specification of Semantic eMail Addresses is based on ontologies that captures tasks and roles of people. Therefore, a sample ontology has been specified for the developed prototype. It describes the domain model of a simple project work scenario within a fictional company. The Web Service Modeling Language WSML [16] as a language for the specification of ontologies has been selected to specify the sample ontology. The resolution of Semantic eMail Addresses is based on an existing reasoner engine [15]. Both tools have been chosen because they are developed by the Digital Enterprise Research Institute (DERI)[2] and therefore it is possible for the author to get information concerning their proper usage.

### 1.1.1 Benefits of Semantic eMail Addressing

Think of a moderate size company with several ongoing projects. The company may have a centralized database containing the information about the personnel, its employment status and the projects the employees are assigned to. Without knowing the names, one may address all 'senior manager' who work for the 'SAP rollout project'. Due to the limitations of conventional mailing lists, addressing such a group of people is a demanding and error-prone task. Especially in a business setting users can benefit from Semantic eMail Addressing as follows.

- **No discovery required.** In the case of many addresses and mailing lists, the desired recipients can be difficult to discover, even for a human. Worse, since email addresses and mailing list names have no semantics, automatic discovery of email addresses and mailing lists by a computer is an extremely difficult task, if not impossible. With Semantic eMail Addressing, discovery is completely obviated.

- **No maintenance required.** Traditional mailing lists require manual labour to maintain. A mailing list administrator must go through the process of creating the list. The list must then be maintained by the administrator himself and the individuals who would like to subscribe to the list, unsubscribe to the list, and change their information with regards to the list. This can be particularly onerous when the user must deal with many lists, for example when his email address changes. Ideally, the user could update his personal information such as email address once, in a single place, and have email addressers automatically adapt. Semantic eMail Addressing makes this possible.

- **Ad hoc addressing** Existing mailing list might not precisely cover the needs the user has. Semantic eMail Addressing is very flexible and may be used just for one specific act. For such cases the creation of a new mailing list would lead to a technical overkill.

- **Sending eMail to people, not strings** Because of the declarative description of the recipients, it is clear who will get the eMail. The sender does not need to know the recipient's actual eMail addresses.

## 1.2 Goal of the project

The purpose of this project is to develop an extension of Thunferbird, that extends Thunderbird by the possibility to

- Specify Semantic eMail Addresses

- Resolve Semantic eMail Addresses during runtime

### 1.3 Structure of this document

After an outline of Mozilla this paper will give a overview of the principles of a Thunderbird extension. The introduction of necessary components to realize semantic eMail Addressing is followed by the concrete implementation of a Thunderbird extension, that extends Thunderbird by the possibility of semantic eMail Addressing.

## 2 Mozilla Thunderbird in general

This chapter introduces the basic idea underlying Mozilla. It especially focuses on Mozilla Thunderbird [10] and how it can be extended.

### 2.1 About Mozilla

Historically, Mozilla had been used internally as a codename for the Netscape Navigator web browser from its beginning. Mozilla is sometimes used to refer to the free software/open source project that was founded in order to create the next-generation Internet suite for Netscape. The Mozilla Organization was founded in 1998 to create the new suite. On July 15, 2003, the organization was formally registered as a not-for-profit organization, and became Mozilla Foundation. The foundation now creates and maintains the Mozilla Firefox browser and Mozilla Thunderbird email application, among other products. The Mozilla trademark is held by the Mozilla Foundation as of 2006.

In March 1998, Netscape released most of the code base for its popular Netscape Communicator internet suite under a free software/open source license. The name of the application developed from this was named as Mozilla, as it was used as the codename of the original Netscape Navigator. After a series of lengthy pre-1.0 cycles, Mozilla 1.0 was released on June 5, 2002. The suite was well known as the free/open source base of the Netscape suite (versions 6 and 7), and its underlying code base (most notably the Gecko layout engine) became the base of many standalone applications, including the Mozilla Foundation's flagship products Firefox and Thunderbird. To distinguish the suite from the standalone products, the suite is often marketed as 'Mozilla Suite', or the more lengthy "Mozilla Application Suite". The Mozilla Foundation no longer maintains the suite, so that their developers can focus on Firefox and Thunderbird. The suite has been unofficially superseded by SeaMonkey, an Internet suite developed by the Mozilla community that is based on the source code of the Mozilla Suite. [6]

## 2.2 Principles of a Thunderbird Extension

A Mozilla extension is an installable enhancement to the Mozilla products (Firefox and Thunderbird) that provides additional functionality. From a technical point of view there is no difference between extensions for Firefox or Thunderbird. Therefore the following explanation can be applied to both.

Every Mozilla extension is divided into three layers: the structure, the style, and the behavior. The structure layer identifies the widgets (menus, buttons, etc.) and their position in the UI relative to each other, the style layer defines how the widgets look (size, color, style, etc.) and their overall position (alignment), the behavior layer specifies how the widgets behave and how users can use them to accomplish their goals. There are also some RDF [8] files which merge the layers. Finally the extension consists of just one archive file. For a better understanding what the layers are actually for, the detailed explanation below refers to the 'Progressmeters Example' from the XUL Periodic Table [13].

### 2.2.1 The structure layer

The basic Idea of Mozilla extensions is the overlay technique. The existing source code files should never be modified, instead one creates so called XUL (XML User Interface Language) [9] files which overlay the standard code. XUL is Mozilla's XML-based language and is used to describe extra content for the UI. XUL files are a general mechanism for adding UI for additional components, overriding small pieces of another XUL file without having to resupply the whole UI, and reusing particular pieces of the UI. Each window and dialog box in Mozilla is defined by a single XUL file (in some cases other XUL files called overlays contribute portions of another window's structure).

The best way to find a XUL file for a window is to use the DOM inspector. The DOM Inspector is a tool bundled with Mozilla that allows you to examine the DOM of web pages and XUL windows.

For a convenient extension development the Extension Developers's extension [3] is highly recommended. Testing JavaScript code, prototyping XUL layouts, and building XPI packages are all made easier by this extension.

Detailed information as well as quite good tutorials concerning XUL can be found on XULPlanet [14]. The XUL periodic table [13] shows some advanced XUL demos in Gecko-based browsers such as Firefox.

A basic example how a XUL file may look like is given below. Figure 1 shows the resulting window.

```
<?xml version="1.0"?>
```

```
<?xml-stylesheet href="chrome://global/skin" type="text/css"?>

<window title="XUL Progressmeters"
        xmlns:html="http://www.w3.org/1999/xhtml"
        xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<script>
<![CDATA[

function addProgress()
{
  var progmeter = document.getElementById("my-progressmeter");
  var progress = parseInt(progmeter.value) + 10;
  progmeter.value = progress;
}

]]>
</script>

<description><html:h1>XUL Progressmeters</html:h1></description>

<vbox flex="1" style="overflow: auto">
 <hbox >

  <groupbox>
  <caption label="determined" />
   <progressmeter id="my-progressmeter" mode="determined" value="10" />
   <button label="Hit me to advance" oncommand="addProgress();" />
  </groupbox>

  <groupbox>
  <caption label="undetermined" />
   <progressmeter mode="undetermined" />
  </groupbox>

 </hbox>
</vbox>
</window>
```

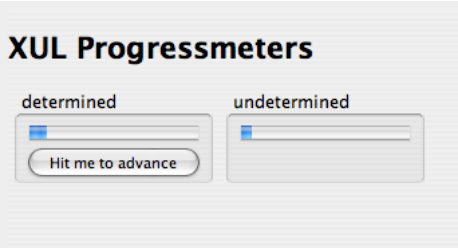



Figure 1: Basic XUL window

### 2.2.2 The style layer

Graphical elements defined in the XUL file are given class names. So they can be styled accordingly by the CSS stylesheet that defines these elements' appearance. Regarding to the 'Progressmeters' example, the stylesheet information is imported by the line:

```
<?xml-stylesheet href="chrome://global/skin" type="text/css"?>
```

### 2.2.3 The behaviour layer

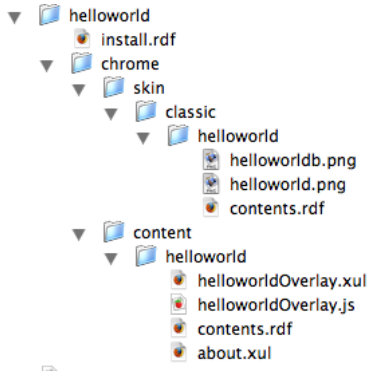
This layer specifies how the widgets behave and how users can use them to accomplish their goals. JavaScript functions are linked with the UI and called when specific events occur (e.g. a button is pressed or any other user interaction) In our example the behaviour layer is represented by the follwing JavaScript function, which is called by the 'oncommand' event from the structure layer.

```
function addProgress()
{
  var progmeter = document.getElementById("my-progressmeter");
  var progress = parseInt(progmeter.value) + 10;
  progmeter.value = progress;
}
```

### 2.2.4 The installation file

All the separate files mentioned above are packed in one single ZIP file which will be renamed as a Cross-Platform Installer Module (XPI) file. A XPI contains installation instructions (install.js or install.rdf) as well as the actual software to install, which is often itself packaged as a JAR file. When downloaded or dropped into an extension manager, XPInstall automatically interacts with the installation instructions contained in the XPI, and installs the contained software.[1]

The structure of a very basic Mozilla extension may look like this: [1]




| Filename | File extension | Corresponding layer | Comments |
|---|---|---|---|
| install | rdf | — | used by the extension manager for installation and keeps information about extension (its name, authors, web site, etc...), and also information about target application e.g. Thunderbird |
| helloworld | png | style | Icon used by about.xul |
| helloworldb | png | style | Icon used by about.xul |
| contents | rdf | — | This is the file that tells Firefox/Thunderbird where to store the overlay information |
| helloWorldOverlay | xul | structure | Adds a helloWorld menue item to the standard file menue |
| helloWorldOverlay | js | behaviour | JavaScript defines a function which is executed when one clicks the elements defined in the corresponding XUL file |
| about | xul | structure | Definition of the popup you see when you click 'About Hello, world!...' in the extensions menu |

[1] To get the files above simply rename the Installation file from helloworld.xpi to helloworld.zip and unpack it. The file can be downloaded from [4]

# 3 The Semantic eMail Extension for Thunderbird (SMAIL) in detail

The following introduces a concrete implementation that allows to use Semantic eMail Addressing.

## 3.1 Cooperate example

As Semantic eMail Addressing is particularly suitable for the usage within companies a fictional business setting has been developed. Think of a company with several employees who are involved in one or more projects the company runs. Employees as well as projects have one or more Subjects they are related to. Subjects are the field of interest an employee has or the topic a project is related to. Each project has a project leader. The extension makes it possible to address an employee or even groups of employees without knowing their eMail addresses. Just by entering the desired descriptive parameter the extension determines the right person and the corresponding eMail Addresses on the fly.

In order to achieve the mentioned functionality, three basic components are required:

- Some type of knowledge base which keeps the actual data (e.g. Persons, Projects, Interests) and describes its relations. The knowledge base used in the SMAIL extension is represented by an Ontology Language. [7]
- A mechanism to extract data from the knowledge base. This is done by an Ontology Reasoner .
- A client application which refers to the Ontology and to the Reasoner service. The client is represented by the SMAIL extension.

### 3.1.1 Ontology Language

To represent and describe the data model the Web Service Modeling Language (WSML) [16] has been chosen. WSML is a frame based language with an intuitive human readable syntax and XML and RDF exchange syntaxes. This Ontology Language allows objectoriented-style conceptual data modelling. Among the five variants of WSML, WSML-Flight has been chosen. WSML-Flight allows efficient implementations for query tasks.

### 3.1.2 Ontology Reasoner

In order to retrieve the desired information from the WSML ontology described above the WSML 2 Reasoner Framework [15] has been chosen. The WSML 2 Reasoner Framework is a highly modular architecture which combines various validation, normalization, and transformation functionalities essential to the translation of ontology descriptions in WSML to the appropriate syntax of several underlying reasoning engines. WSML 2 Reasoner Framework is implemented in Java and can be freely downloaded. The Frameworks basic functionality can also be accessed via a Web Client [11] or as Web Service [12]. The SMAIL extension consumes the reasoning service via Web Client. For that purpose the query string and a reference to the ontology location is passed to the service. The result coming back from the service is a HTML Document which includes the desired information.

### 3.1.3 Graphical data representation

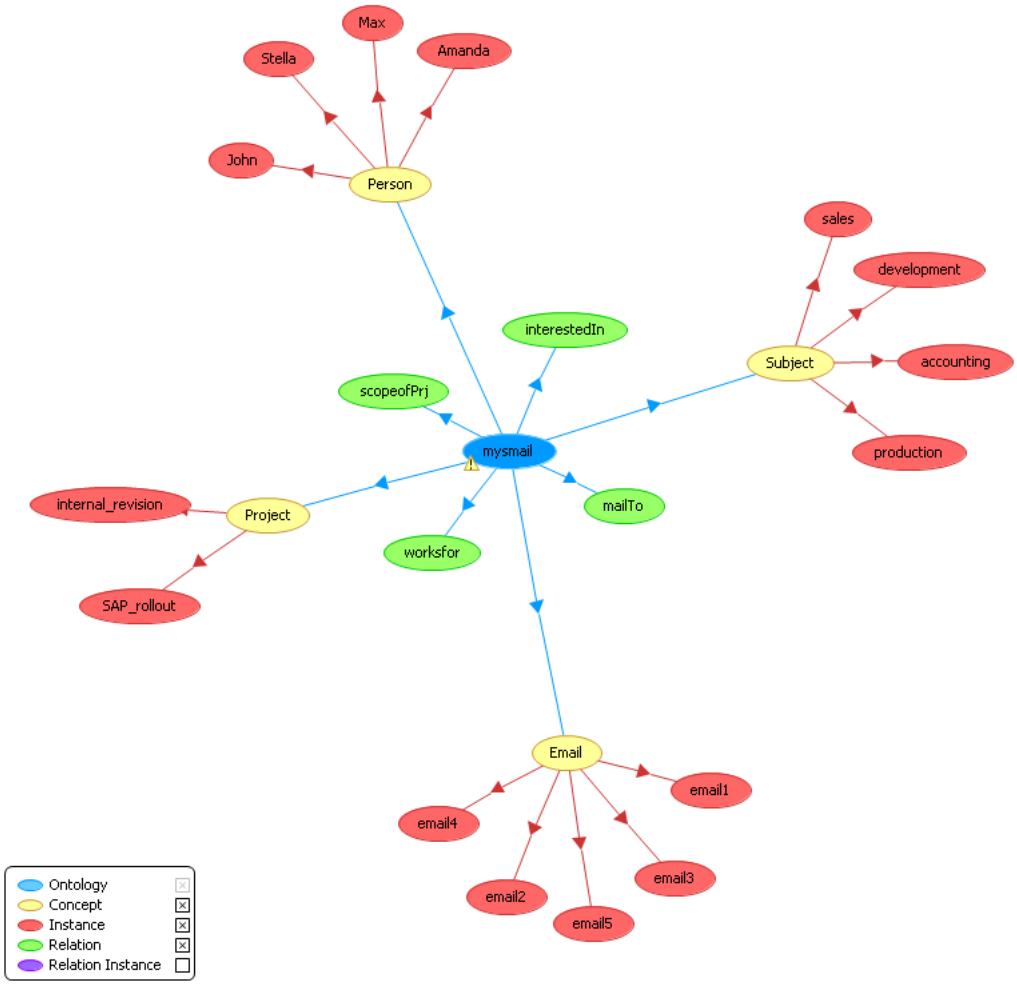



Figure 2: The WSML ontology the semantic eMail extension is based on

### 3.1.4 Data representation as a text file

mysmail.wsml[2]

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
  namespace {_"http://www.mysmail.com/#",
    dc _"http://purl.org/dc/elements/1.1#",
    foaf _"http://xmlns.com/foaf/0.1/",
    wsml _"http://www.wsmo.org/wsml-syntax#"}

  /*
   * ONTOLOGY - WSML test-ontology for mysmail - SemanticEmail
   */

  ontology mysmail

  nonFunctionalProperties
    dc#title hasValue "about Persons an Projects"
    dc#subject hasValue "WSML test-ontology for mysmail - SemanticEmail"
    dc#description hasValue "fragments of a company/project ontology to provide WSML examples"
    dc#contributor hasValue  _"http://mysmail.com"
    dc#date hasValue _date(2006,12,11)
    dc#format hasValue "text/html"
    dc#language hasValue "en-US"
    dc#rights hasValue _"http://mysmail.com"
    wsml#version hasValue "$Revision: 0.1 $"
  endNonFunctionalProperties

  importsOntology { _"http://www.wsmo.org/ontologies/location",
_"http://xmlns.com/foaf/0.1" }
```

[2]The WSML file is located at [5]

```
concept Person
hasfirstName impliesType foaf#firstName
hassurname impliesType foaf#surname
hasWebSite impliesType foaf#homepage

concept Project
hasprojectName impliesType _string
hasprojectDescription impliesType _string
leader impliesType Person

concept Subject
nonFunctionalProperties
dc#description hasValue "describes a apecific theme or topic a Project or Person is related to"
endNonFunctionalProperties
hasName    impliesType _string
hasDescription impliesType _string

concept Email
hasStringRepresentation impliesType _string

relation mailTo (ofType Person, ofType Email)
relation worksfor (ofType Person, ofType Project)
relation interestedIn (ofType Person, ofType Subject)
relation scopeofPrj (ofType Project, ofType Subject)

/*
 *  Instances
 */

//Persons
instance Max memberOf Person
hasfirstName hasValue Max
hassurname hasValue Mustermann
hasWebSite hasValue "max.mysmail.com"

instance Amanda memberOf Person
hasfirstName hasValue Amanda
hassurname hasValue Musterfrau
hasWebSite hasValue "amanda.mysmail.com"

instance John memberOf Person
hasfirstName hasValue John
hassurname hasValue Jankins
hasWebSite hasValue "john.mysmail.com"

instance Stella memberOf Person
hasfirstName hasValue Stella
hassurname hasValue Steel
hasWebSite hasValue "stella.mysmail.com"

//Projects
instance internal_revision memberOf Project
hasprojectName hasValue internal_revision
hasprojectDescription hasValue "annual internal revision"
leader hasValue Max

instance SAP_rollout memberOf Project
hasprojectName hasValue SAP_rollout
hasprojectDescription hasValue "internaltional SAP introduction"
leader hasValue John


//Subjects
instance sales memberOf Subject
hasName hasValue sales
hasDescription hasValue "Deals with customers"

instance accounting memberOf Subject
hasName hasValue accounting
hasDescription hasValue "internal accounting administration"

instance production memberOf Subject
hasName hasValue production
hasDescription hasValue "has to do with the manufactoring process"

instance development memberOf Subject
hasName hasValue development
hasDescription hasValue "Product development"

//Email
instance email1 memberOf Email
hasStringRepresentation hasValue "max@mysmail.com"
instance email2 memberOf Email
hasStringRepresentation hasValue "max1@mysmail.com"
instance email3 memberOf Email
hasStringRepresentation hasValue "stella@mysmail.com"
instance email4 memberOf Email
hasStringRepresentation hasValue "john@mysmail.com"
instance email5 memberOf Email
hasStringRepresentation hasValue "amanda@mysmail.com"

//Relations
relationInstance mailTo(Max, email1)
relationInstance mailTo(Max, email2)
relationInstance mailTo(Stella, email3)
relationInstance mailTo(John, email4)
relationInstance mailTo(Amanda, email5)

relationInstance interestedIn (Stella, accounting)
relationInstance interestedIn (Stella, sales)
relationInstance interestedIn (Amanda, accounting)
relationInstance interestedIn (Max, accounting)
relationInstance interestedIn (John, development)
relationInstance interestedIn (John, production)

relationInstance scopeofPrj (SAP_rollout, sales)
relationInstance scopeofPrj (SAP_rollout, development)
relationInstance scopeofPrj (internal_revision, accounting)

relationInstance worksfor(Stella, SAP_rollout)
relationInstance worksfor(John, SAP_rollout)
relationInstance worksfor(Stella, internal_revision)
relationInstance worksfor(Amanda, internal_revision)
relationInstance worksfor(Max, internal_revision)
```

### 3.1.5 Data extraction with WSML 2 Reasoner Framework

To get an overview of the query syntax one may consider the following examples which refers to the ontology mentioned above. For a detailed specification of the WSML 2 Reasoner Framework please refer to the official documentation [16].

- *Get a list of all projects:*

```
?project memberOf Project
?project[hasprojectName hasValue ?name]
and ?project[hasprojectDescription hasValue ?text]
```

- *Get the string representations of all eMail addresses which belong to a person with firstmane 'Stella':*

```
?person memberOf Person
?person[hasfirstName hasValue Stella] and mailTo(?person,?email)
and ?email[ hasStringRepresentation hasValue ?mailstring]
```

- *Get the string representations of all eMail addresses which belong to everyone who works for the 'SAP_rollout' project:*

```
?Project[hasprojectName hasValue SAP_rollout]
and worksfor(?Person,?Project)
and mailTo(?Person,?Email)
and ?Email[hasStringRepresentation hasValue ?Mail]
```

## 3.2 Technical infrastructure

To achieve the maximum possible flexibility the WSML ontology, the WSML 2 Reasoner and the client application (the actual extension) are assumed to be on distinct places. This architecture allows maintaining the WSML ontology instances at a central spot. Therefore all connected clients can be sure to access the same up-to-date data instances. The data is retrieved on the fly upon user interaction with the extension. This requires network access since the SAMIL client needs to communicate with the ontology reasoner component. In general this is not a problem since most of people write their eMails with working network access. Figure 3 shows the how the components interact.

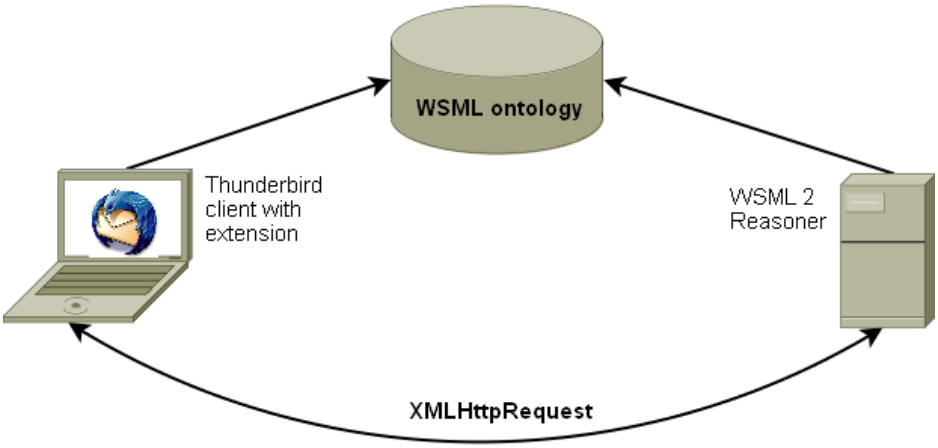
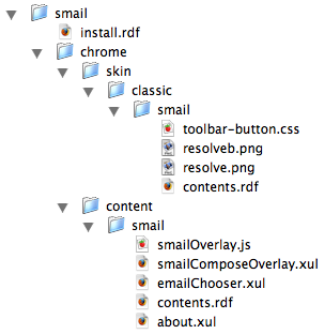



Figure 3: Communication between client and reasoner is based on the XMLHttpRequest[18]

### 3.3 The SMAIL extensions file structure

SMAIL extension consists of the following files:



| File | Comment |
|---|---|
| smail/install.rdf | Used by the extension manager for installation and keeps meta information about the extension (its name, authors, web site, etc...) as well as information about the target application in this case: Thunderbird. |
| smail/chrome/content/smail/contents.rdf | Tells Thunderbird where to overlay the custom XUL files. In this case we overlay the messenger compose window. |
| smail/chrome/content/smail/smailComposeOverlay.xul | This XUL is overlaid to the messenger compose window. It adds the SMAIL Button. (to access to SMAIL chooser window) |
| smail/chrome/content/smail/emailChooser.xul | Defines the modal SMAIL chooser window which is called upon the SMAIL Button is pressed. |
| smail/chrome/content/smail/smailOverlay.js | This JavaScript file keeps all the program logic. (e.g. implementation of the client – server communication, creation of the query strings based on the user's input ,DOM parsing and further handling of the query results, form validation) |
| smail/chrome/content/smail/about.xul | Simple popup window which is shown upon click on 'About' in the extensions menu. |
| smail/chrome/skin/classic/smail/contents.rdf | Tells Thunderbird where to overlay the custom style sheet files. In this case we overlay the messenger compose window. |
| smail/chrome/skin/classic/smail/toolbar-button.css | Keeps the style sheet information. (e.g. which button is related to which icon) |
| smail/chrome/skin/classic/smail/resolve.png | SMAIL Button icon small size |
| smail/chrome/skin/classic/smail/resolveb.png | SMAIL Button icon regular size |

Figure 4: SMAIL extension's files and their usage

## 3.4 Installation

Installing the SMAIL extension is a two-step process. First, save the extension to your computer (right-click and choose "Save Link As...") in a convenient location. Second, in Thunderbird go to the "Tools" menu, select "Extensions", click the "Install" button, browse to the extension that you just downloaded, and click the Open/OK button.

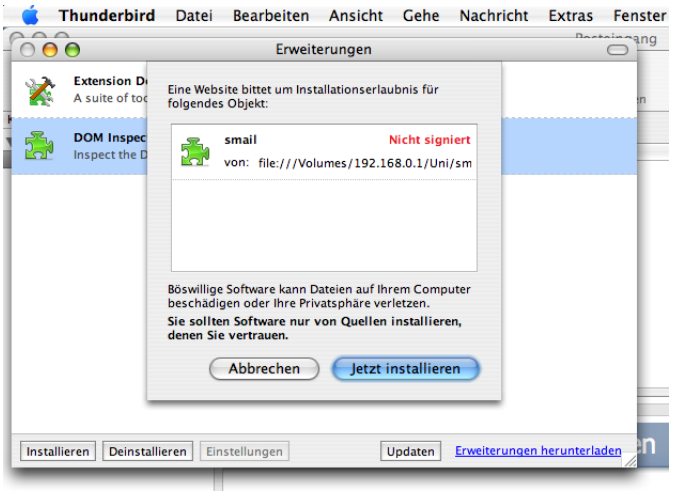



Figure 5: Extension manager

The warning message can be ignored. Click the 'Install now' button. After restarting Thunderbird, the extension manager window should look like this:

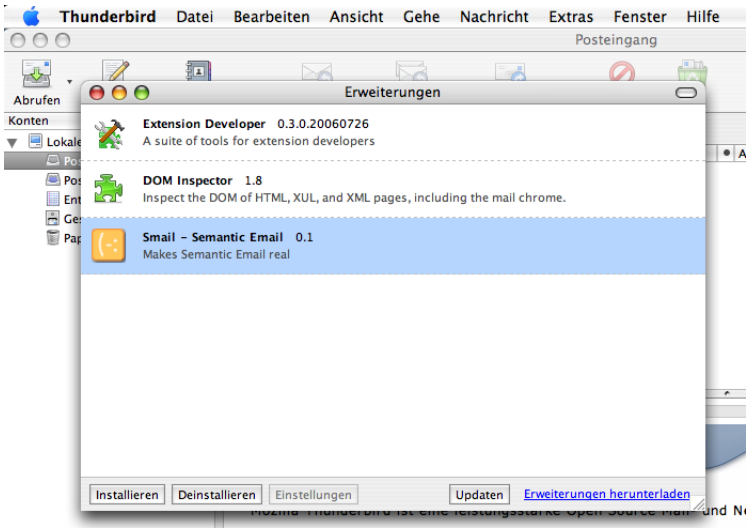



Figure 6: Installation succeeded

As the 'Semantic Email' button does not show up automatically in the message compose window, it has to be added manually.After opening a new compose window, go to the 'View' menu, select 'Toolbar', 'Customize…' and simply drag and drop the button on the toolbar of the message compose window.

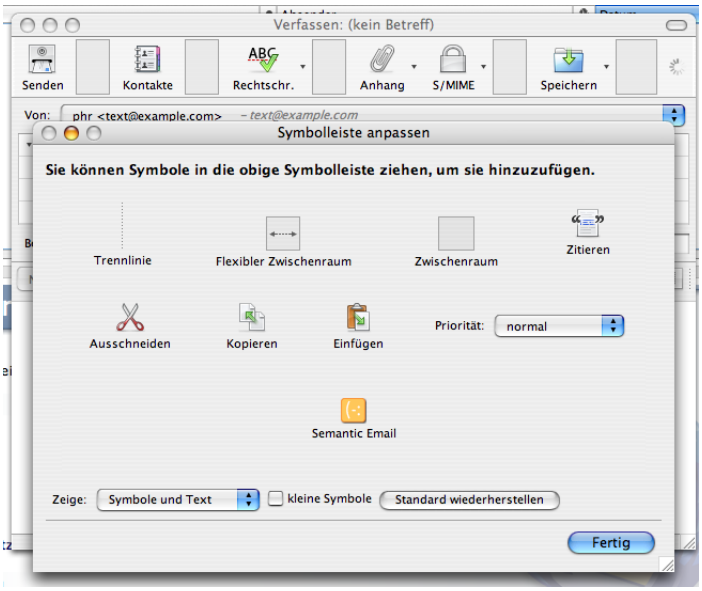



Figure 7: Drag and drop the orange button to the toolbar

Finally, the message compose window should look like this:

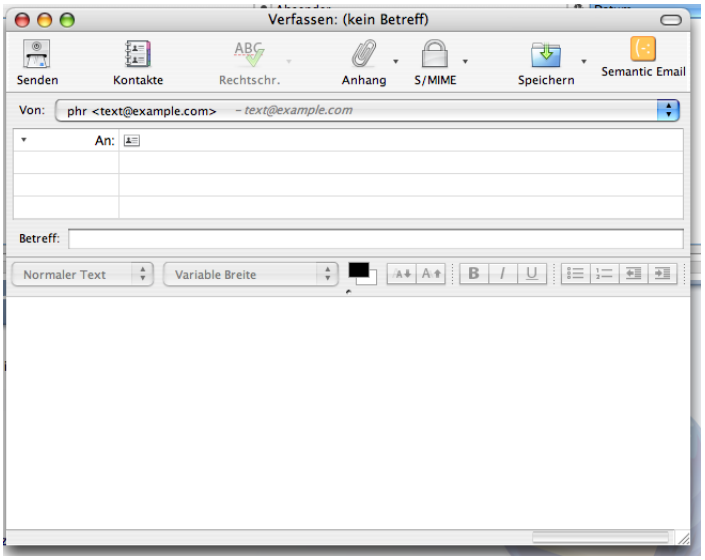



Figure 8: Message compose window after installation

## 3.5 Usage

The usage of the SMAIL extension is pretty simple. Click the 'Semantic Email' button and the mail chooser window shows up.

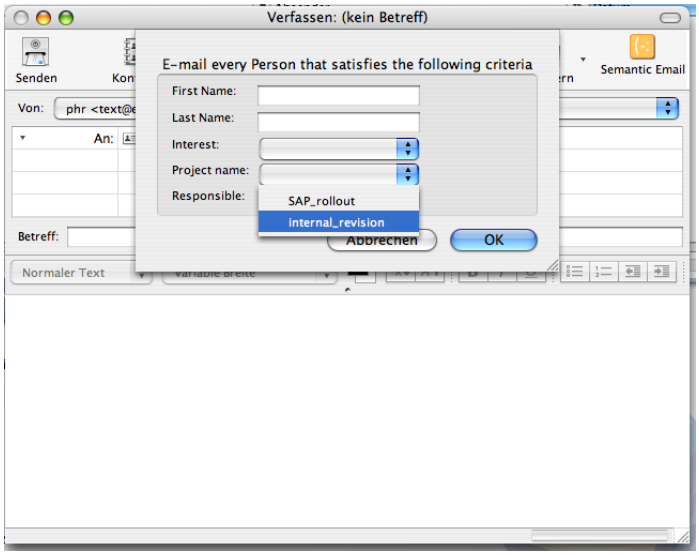



Figure 9: Email chooser window

Make your choice and press OK. Please note, that the values of the drop down fields are retrieved on the fly upon pressing the arrow button. Depending on your network connection this may take a second. According to the example above, the email addresses of every person who is assigned to the project 'internal_revision' will automatically inserted in the recipients field of the message compose window. You may check the plausibility of the generated list of recipients and go ahead writing the eMail as usual.

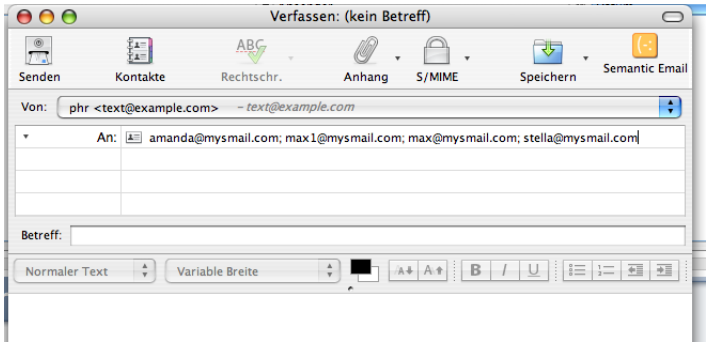



Figure 10: Check the plausibility of the generated list of recipients

You may add any additional person of you choice to additional recipient fields on the message compose window.

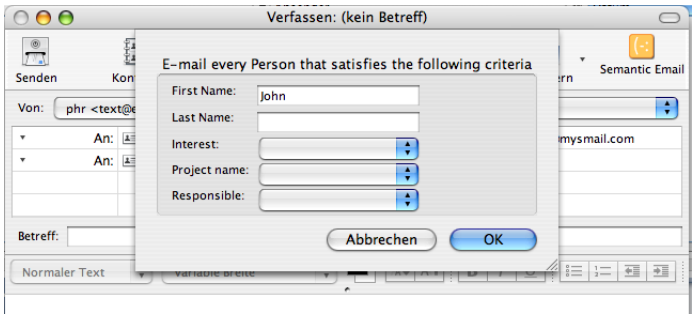



Figure 11: Additional recipient

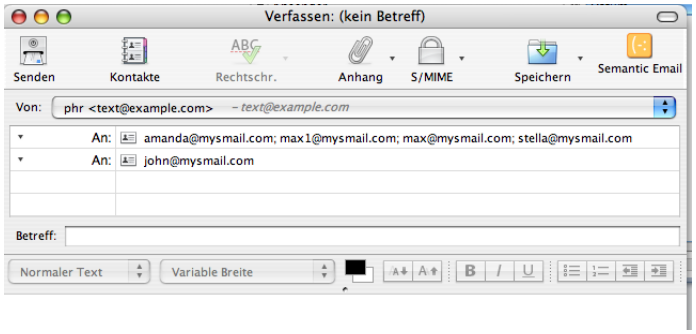



Figure 12: Write and send the eMail as usual

# 4 Conclusion

This paper overviews the basic idea of Semantic eMail Addressing and describes a prototype implementation of a Semantic eMail Extension to Mozilla Thunderbird. Semantic eMail Addressing is a powerful tool which can facilitate the contacting of individuals based on their characteristics. Therefore, Semantic eMail Addressing may improve the efficiency of eMail communication with in dynamic groups. There are still open issues concerning Semantic eMail Addressing which emerged during working on this paper:

Replying on Semantic eMail Addresses: As the set of people satisfying a condition may change over time, replying can result in new people receiving the reply and old people not receiving the reply, which may or may not be the desired behaviour.

As Semantic eMail Addressing could easily be used to email large numbers of people, safeguards must be put in place to make sure that user errors do not result in mass spamming. On the public Internet Semantic eMail Addressing is hard to control.

A sophisticated Semantic eMail client should also support the possibility of using various ontologies. The client would need to dynamically generate the chooser interfaces based on the ontology.

Because of the extensibility of Mozilla Thunderbird it is suitable eMail client for prototyping and testing purposes. Unfortunately, it is not very widespread among companies. But the implementation of the SMAIL extension discussed in this paper may be adapted to other clients as soon as the client offers an application program interface (API).

# References


[1] Creating XPI Installer Modules. http://developer.mozilla.org/en/docs/Creating_XPI_Installer_Modules.

[2] Digital Enterprise Research Institute (DERI). http://deri.org/.

[3] Extension Developer's Extension. http://ted.mielczarek.org/code/mozilla/extensiondev/.

[4] helloworld extension. http://www.phr24.info/helloworld.xpi.

[5] Location of the SMAIL ontology. http://www.phr24.info/mysmail.wsml.

[6] Mozilla. http://en.wikipedia.org/wiki/Mozilla.

[7] Ontology. http://en.wikipedia.org/wiki/Ontology_Language.

[8] Resource Description Framework (RDF). http://www.w3.org/RDF/.

[9] The Joy of XUL. http://developer.mozilla.org/en/docs/The_Joy_of_XUL.

[10] Thunderbird 1.5 - Reclaim you inbox. http://www.mozilla.com/en-US/thunderbird/.

[11] WSML Reasoner – Web Client. http://tools.deri.org/wsml/rule-reasoner/v0.1/.

[12] WSML Reasoner – Web Service, WSDL location. http://tools.deri.org/wsml/rule-reasoner/v0.1/services/reasoner?wsdl.

[13] XUL Periodic Table. http://www.hevanet.com/acorbin/xul/top.xul.

[14] XUL Planet. http://www.xulplanet.com/.

[15] Uwe Keller Nathalie Steinmetz Gabor Nagypal Stephan Grimm Darko Anicic, Holger Lausen. WSML 2 Reasoner Framework - Overview. http://tools.deri.org/wsml2reasoner/index.html.

[16] Reto Krummenacher Axel Polleres Livia Predoiu Michael Kifer Dieter Fensel Jos de Bruijn, Holger Lausen. The Web Service Modeling Language WSML D16.1v0.21. http://www.wsmo.org/TR/d16/d16.1/v0.21/.

[17] Lee-Ming Zen Michael Kassoff, Charles Petrie and Michael Genesereth. Semantic Email Addressing: Sending Email to People, Not Strings. http://logic.stanford.edu/ mkassoff/papers/sea-fss06.pdf.

[18] Anne van Kesteren. The XMLHttpRequest Object. http://www.w3.org/TR/XMLHttpRequest/.