Diploma Thesis

July 7, 2006

Semantic Clipboard

Martin Morger
of Zürich, Schweiz (00-915-363)

supervised by
Prof. Dr. Harald C. Gall
Dr. Gerald Reif

University of Zurich
Department of Informatics

s.e.a.l.
software evolution & architecture lab

Diploma Thesis

Semantic Clipboard

Martin Morger

University of Zurich
Department of Informatics

Diploma Thesis

Author: Martin Morger, martin.morger at gmail.com

Project period: 08.03.2006 - 08.07.2006

Software Evolution & Architecture Lab
Department of Informatics, University of Zurich

Acknowledgements

I would like to thank all those who supported me during the programming and writing of this thesis: Gerald Reif, for his valuable input and support, and Sabine Locher for the proof reading. Special thanks go also to the numerous developers of the tools and frameworks used for creating this thesis, in particular Eclipse, Jena, iCal4j, Log4j, and TeXnicCenter.

Abstract

The *Semantic Web* provides a framework to share data across the boundaries of applications, enterprises, and communities. It uses the *Resource Description Framework* to provide metadata describing any resource accessible, or at least identifiable, on the Web. Current clipboard applications allow the exchange of data between applications running on the same platform while the semantics of the data are usually only retained if the source and target applications are part of a specific application suite. Expanding the range of data sources from desktop applications to Websites, the process of copying data from a Web resource into an application results in losing most of the semantics of the data, as the target application recognizes the pasted data as formatted or plain text only.

This thesis presents an implementation of the *Semantic Clipboard* concept using an extensible plugin architecture. The implemented Java application extracts RDF metadata describing a Web resource from accordingly annotated Websites and pastes them into a supported desktop application, retaining the semantics of the data.

By implementing a plugin architecture, the Semantic Clipboard uses individual plugin modules to extract ontology-specific data from the source location, to store this data temporarily in specific data containers, and to paste it into a suitable desktop application. To extend the range of supported source ontology vocabularies and target applications, additional plugins may be developed and registered at the Semantic Clipboard. The current implementation provides a number of plugins, supporting various ontology vocabularies as well as different desktop applications on the Mac and Windows platforms.

Zusammenfassung

Das *Semantische Web* ermöglicht es, Daten über die Grenzen von Applikationen, Firmen und Organisationen hinweg auszutauschen. Unter Verwendung des *Resource Description Framework* werden dabei Metadaten erstellt, mit denen beliebige Ressourcen beschrieben werden können, solange diese durch das Web erreichbar, oder zumindest identifizierbar sind. Gängige Clipboardapplikationen ermöglichen den Austausch von Daten zwischen Programmen des gleichen Betriebssystems, wobei die Semantik der Daten meist nur beim Austausch zwischen Programmen innerhalb von bestimmten Anwendungspaketen erhalten bleibt. Beim Kopieren von Daten aus einer in einem Webbrowser angezeigten Webseite geht die Semantik der kopierten Daten jedoch ganz oder teilweise verloren, da die Zielanwendung die eingefügten Daten meist nur als (un-)formatierten Text erkennt.

Diese Diplomarbeit beschreibt die Implementierung einer Plugin-Architektur des *Semantic Clipboard* Konzeptes. Die vorliegende Java Anwendung extrahiert RDF Metadaten, die eine Web Ressource beschreiben, aus entsprechend annotierten Webseiten und fügt diese Daten in eine unterstützte Desktop Anwendung ein, wobei die Semantik der Daten erhalten bleibt.

Die implementierte Plugin-Architektur verwendet einzelne Plugin-Module, um Ontologie-spezifische Daten aus der ausgewählten Quelle zu extrahieren, diese in spezifischen Datenbehältern zu speichern und sie schliesslich in eine passende Desktopanwendung einzufügen. Um die Menge der unterstützen Ontologie-Vokabulare und Applikationen zu erweitern, können einfach zusätzliche Plugin-Module entwickelt und zur bestehenden Konfiguration hinzugefügt werden. Die vorliegende Implementierung umfasst eine Reihe von Plugins, die das Lesen von Daten aus diversen Ontologien und das Einfügen dieser Daten in verschiedene Zielanwendungen der Mac und Windows Plattformen unterstützen.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 Motivation

The goal of a Semantic Clipboard, as it was first introduced in [Pai05] and is presented in this thesis, is to enable the exchange of data between applications without the semantics of the data being lost. Even within homogeneous environments such as the Windows or Mac OS operating systems, the operating system clipboard usually provides only a partial implementation of this functionality. Text being copied into the system clipboard keeps its formatting when it is pasted into another application. Its semantics however, e.g. the fact that the copied text represents a person entry in an address book, is lost as the data is treated only as a piece of formatted text by the target application.
Extending the range of possible data sources from local applications to any Website on the World Wide Web, the recognition of semantics is even worse, as current Web Browser applications represent most information found on Websites simply as formatted text. A human user assigns the applicable semantics of the text based on the context the information; i.e. the user knows how a postal address looks like, so he interprets a piece of text containing a name, street, zip code and city as an address. When the user copies this particular information from the Website, the system clipboard though threats the copied data as (formatted) text only. In order to paste the address information into an address book application, the user has to manually create a new address entry and paste the copied data into the respective address fields.

1.2 The Semantic Web and RDF

The *Semantic Web* is a collaborative effort led by the World Wide Web Consortium (W3C)[1] with participation from a large number of researchers and industrial partners. It provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. The Semantic Web is based on the *Resource Description Framework (RDF)* [Con06b], which integrates a variety of applications using the *Extensible Markup Language (XML)*[2] for syntax and *Uniform Resource Identifiers (URI)*[3] for naming [Con06c, AvH04]. RDF is intended to represent metadata describing World Wide Web resources, such as information about the author, title or modification date of a Website. Generalizing the concept of a *Web Resource*, RDF can

[1] `http://www.w3.org/Consortium/`
[2] `http://www.w3.org/XML/`
[3] `http://www.ietf.org/rfc/rfc1630.txt`

be used to represent descriptions of any resource that can be identified on the Web, whether the resource is accessible trough the Web or not. Such a resource may be a person, a product in a store or any other real or virtual object.
Using RDF, a resource can be described by *Statements*, which resemble statements of a natural language and consist of a subject (the resource to be described), a predicate, and an object, each identified by a specific URI. As an example, it is assumed that there is a person, identified by the URI `http://seal.ifi.unizh.ch/people/doe`, whose name is *John Doe* and whose email address is `john.doe@ifi.unizh.ch`. This group of statements might be represented as an RDF graph as shown in Figure 1.1

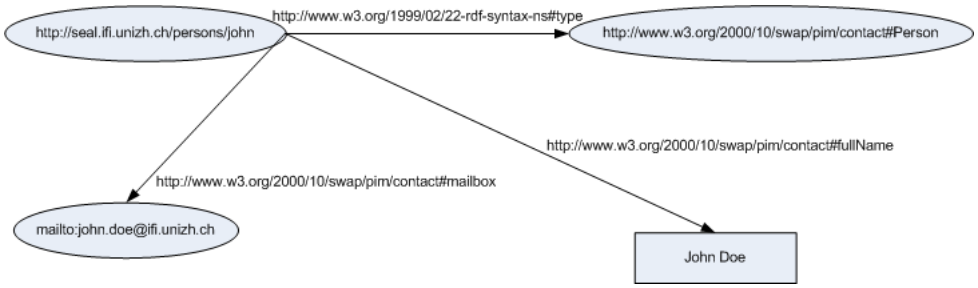

Figure 1.1: An RDF graph describing the person John Doe.

Using RDF to describe the data of a Website, the content of the World Wide Web can be "understood" by computers, i.e. programs can identify the particular semantics of the data and process it accordingly. This opens up a whole new field of possible Web applications: Instead of only indexing the content of the Web and letting the user search for keywords, Semantic Web applications can actively search for specific kind of data.
An example for such an application, as described in [BLHL01], is a program that searches the Web for a certain kind of medical specialist that has his practice within a defined range from the residence of the user. After looking up the published list of available appointment times and comparing it to the free entries in the user's agenda, the program schedules an appointment with the Semantic Web application of the medical specialist and informs the user about the appointment. Although the information about the doctor's field, address and available appointment times might also be available on a traditional text-based Website, a computer program, unlike a human Web-user, would not be able to recognize the semantics of the data.

1.3 Semantics of Data

The *semantics* of some piece of data are highly dependent on the target application the data will be processed with. A bank account statement shown on an electronic banking Website whose content can be exported into different target applications serves as an example to explain this idea as shown in Figure 1.2. The account statement is supposed to show a list of cash transfers into and out of the bank account, specifying its date and value, the name and address of the sender or receiver of the money, and a short description of the purpose of the transfer.
Exported to a home accounting application such as *Microsoft Money*, the semantics of the data will be a series of accounting records resulting from the cash transfers containing the date and value of the transfers. The same data may be represented in a sheet of a spreadsheet calculation program as *Microsoft Excel*. In a calendar application such as *Apple iCal*, the resulting semantics are a list of events which tell when the particular transfers took place. Using a contact management

application like *Microsoft Outlook*, the semantics may be a list of person entries which represent the persons that have sent or received the money transfers.

1.4 Functionality of the Semantic Clipboard

The Semantic Clipboard implemented in this thesis allows to extract information provided as RDF metadata annotations from a Website and to paste it into a local desktop application that can handle the semantics of the data.
To ensure a high degree of flexibility regarding the number of different source data ontologies and target applications the Semantic Clipboard can support, an extensible *plugin* architecture has been implemented. The functionality for extracting and processing the source data and pasting them into the target application is provided by plugin modules, which are developed individually for each ontology vocabulary and target application to be supported and are registered at the main clipboard application using an RDF manifest file.
The user interface of the application consists of a Wizard dialog window, which allows to display the different interface components required for user interaction in a consistent layout, following the work flow of the application, which includes the following steps: After the initialization process of the application, the user is requested to specify the *Uniform Resource Locator (URL)* referring to the RDF source data. The application then extracts the RDF source from the specified location and passes the data on to its plugin classes, which will try to process the data. If the processing has been successful, a list of applicable applications is presented to the user, who then can choose the target application the data should be pasted in. Finally the data is being pasted into the selected target application using platform and application specific methods like application or command line scripting.

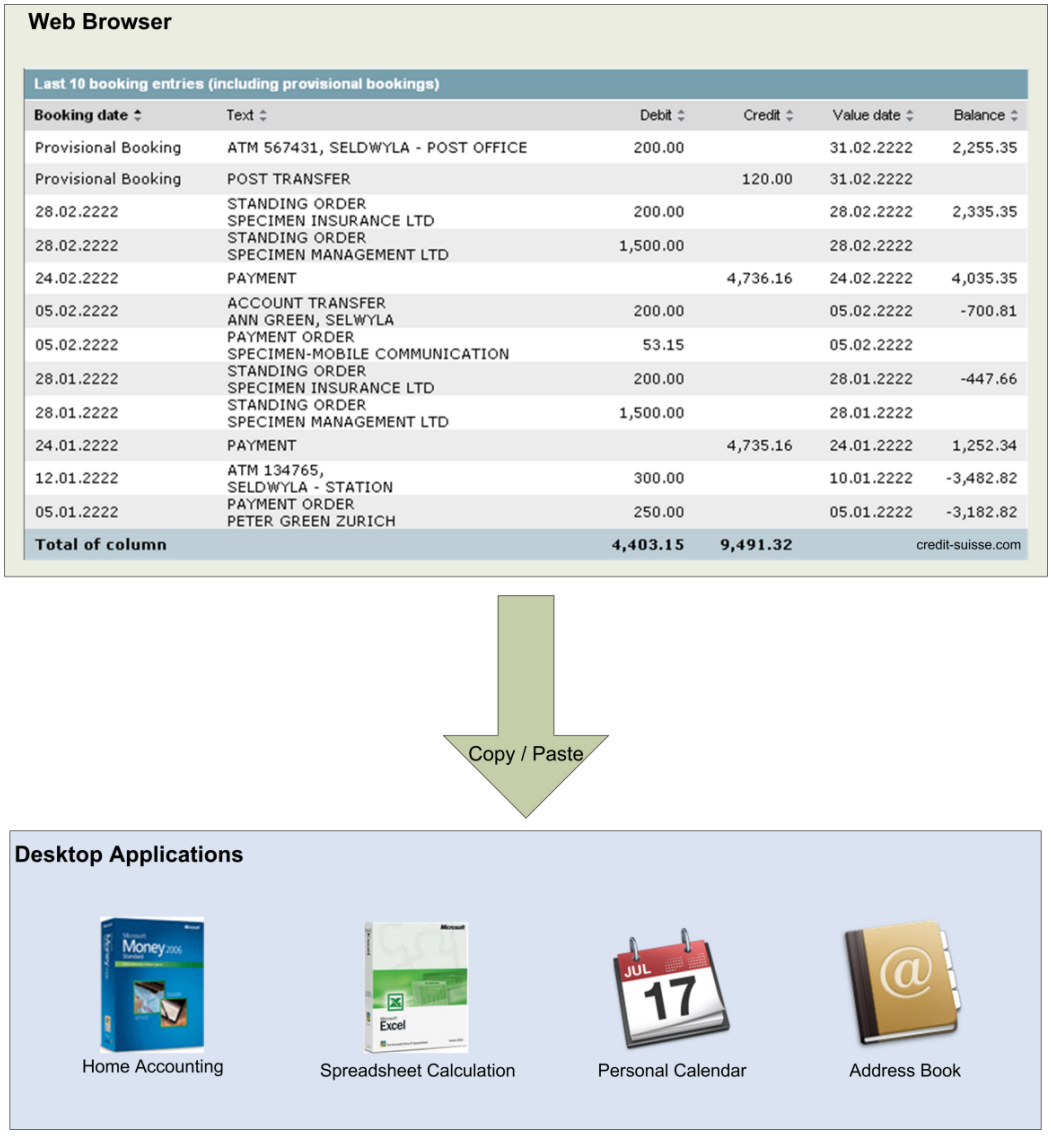

Figure 1.2: Exporting the data of a bank account statement into different applications.

Chapter 2

Implementation

2.1 Architecture

A main requirement imposed on the design of the Semantic Clipboard was the possibility to extend the number of supported ontologies and target applications by developing additional plugin modules. The current Semantic Clipboard application fulfills this requirement by implementing an extensible plugin-architecture, as shown in Figure 2.1. Functionality such as managing the configuration of available plugins, retrieving the source RDF data and displaying the graphical user interface is provided by commonly used core components. The tasks of extracting ontology-specific data from the source, storing this information in ontology-specific containers, and pasting it into a suitable target application are carried out by the respective plugin classes.
To allow all involved classes to access the references of shared objects, such as the user interface and dynamically instantiated plugin classes, the *Session* object provides functionality to store and retrieve references of objects.

There are three types of plugin modules designed to carry out their specific tasks:

- *Reader* plugins for each ontology vocabulary, see Section 2.3, such as information about a person stored in an electronic business card format, information about an event, or about a music title. Each reader plugin extracts the information of its specific ontology and creates a data container object using the extracted data.
- *Data Container* plugins which provide a storage container for a concept of the real world such as an address or a calendar event. The containers are filled with the data extracted from the RDF description by the reader plugin.
- *Application* plugins for each target desktop application such as *Microsoft Outlook* or the *Apple iCal* calendar application. Each application plugin implements the functionality to paste data contained in one ore more data containers into a specific target application. To paste the data into the target application, the plugin may implement platform and application specific means like application scripting such as *Apple Script*, or create a temporary file which is then being imported into the application using command line scripting.

This plugin-architecture is reflected in the Java package structure, consisting of the three main packages `core`, `plugins`, and `shared`, each containing a number of sub-packages, as shown in Figure 2.2.

Core Package The classes in this package provide the core functionality of the Semantic Clipboard, i.e. initializing and managing the configuration of the available plugins, retrieving and reading the RDF source data, and displaying a graphical user interface.

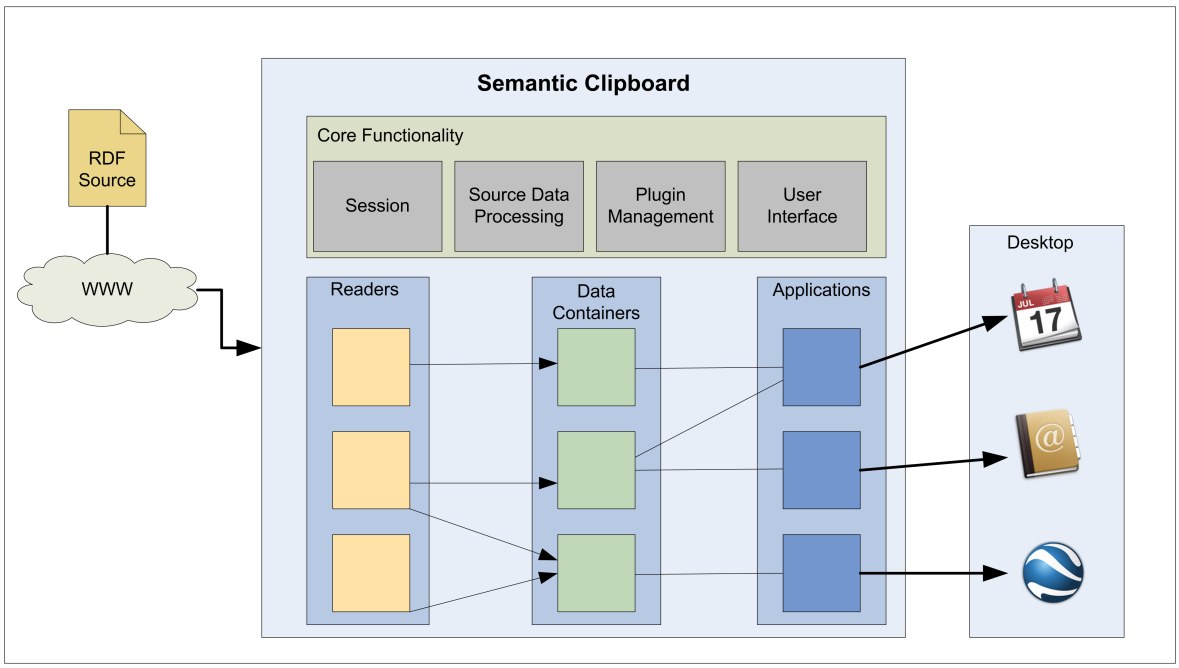

Figure 2.1: Data Flow and Plugin Component Structure.

Plugin Package Contains the reader, data container, and application plugin classes which extract its ontology-specific data from the RDF source, store it temporarily, and paste the data into target applications.

Shared Package Provides functionality used by modules from both the core and main packages, such as the `Session` class, `Exception` classes, ontology vocabularies, and some utility classes.

2.2 Technologies

The following technologies and libraries have been used to develop the Semantic Clipboard:

Java Software Development Kit (J2SE), Version 1.4.2 for Mac The Semantic Clipboard application is implemented in the *Java*™ programming language[1]. Being a relatively young language, Java has rapidly become one of the major languages for object oriented programming. Its virtual machine architecture allows programmers to *write once, run everywhere*, as the the Java compiler will generate a platform-independent byte-code, that can be run on any platform that provides a Java virtual machine. In addition to being a modern object oriented language, its community provides a large amount of freely available libraries.

Eclipse IDE The *Eclipse* Integrated Development Environment (IDE)[2] has been used to support the development process. The Eclipse IDE provides a large array of coding assisting, refactoring, and debugging functionality, and its license allows anyone to use the program free of

[1] `http://java.sun.com/`
[2] `http://www.eclipse.org/`

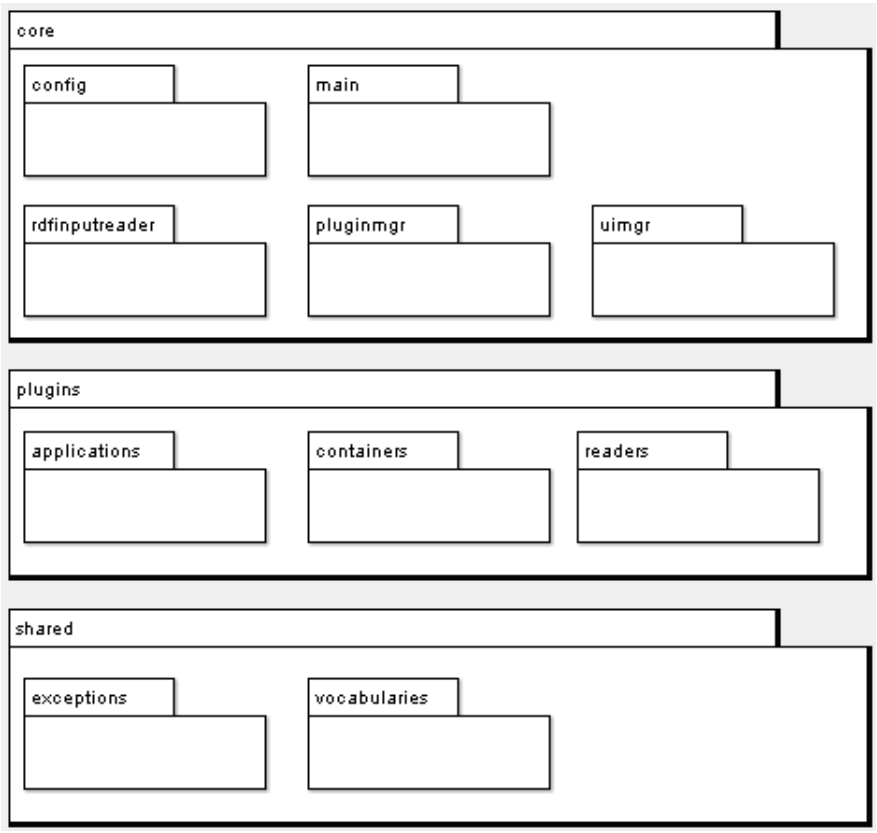

Figure 2.2: Package Structure of the Semantic Clipboard.

charge. Its plugin architecture allows the IDE to be extended with additional functionality, its community contributes also a wide range of freely available plugin modules.

Jena Semantic Web Framework *Jena*[3] is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS[4], OWL[5], and SPARQL, which is described in [Con06d], as well as a rule-based inference engine. The framework has originally been developed by HP Labs Semantic Web Research and is now available under an Open Source license. The Semantic Clipboard uses Jena RDF functionality to read the RDF source data and plugin manifest files, creating a Jena model from it, and applying queries on the model.

Apache log4j Logging Package The *log4j*[6] package provides efficient logging using hierarchical logging levels which allow to customize the logging output at runtime without modifying the application source code. In the Semantic Clipboard application, log4j is used to log warning and error messages as well as debugging information to the system console. The priority of logging messages to be displayed can be customized using command line parameters specified at the application startup.

[3] `http://jena.sourceforge.net/`
[4] RDF Schema
[5] Web Ontology Language
[6] `http://logging.apache.org/log4j/`

Xerces2 Java Parser The functionality to parse HTML[7] files containing linked or embedded RDF content is provided by the *Xerces2*[8] parser which is part of the Apache XML project.

iCal4j iCalendar API The *iCal4j* Java library[9] provides a Java API to read, process, and output iCalendar data. The `ICalContainer` data container plugin, see Section 2.6.2 on Page 16, uses iCal4j to internally store a representation of iCalendar objects and to create the contents of iCalendar files to be imported into target applications.

dom4j XML The *dom4j library*[10] provides an Open Source library for working with XML, XPath, and XSLT on the Java platform using the Java Collections Framework and with full support for DOM, SAX, and JAXP. The `GeoContainer` data container plugin, which is described in Section 2.6.2 on page 17, creates its XML output to be imported into the target application using the XML functionality provided by dom4j.

2.3 Implemented Ontologies

Each RDF graph adheres to an *ontology*, which is a data model that describes a domain. Using an ontology allows reasonings about objects in a domain and the relationships between them. RDF supports various ontology languages, such as *RDF Schema*[11] and the *Web Ontology Language (OWL)*[12].
In order to extract the ontology-specific data provided in an RDF source, each reader plugin implements a specific ontology vocabulary.

2.3.1 vCard

vCard is a standard for the exchange of personal data, similar to the data to be found on a paper business card. The standard was defined by the Internet Society in RFC 2426 [Soc98b] and is widely supported by mail client applications, mobile phones, and personal digital assistants.
A vCard object contains information about a person such as his name, birthday, and nicknames, organizational data containing the name of the organization and of the business unit, postal addresses including geographical coordinates, and contact information, like email addresses and telephone numbers. vCard data is usually stored as unformatted ASCII data. The W3C however has developed a recommendation for a representation of vCard data using RDF [Con01], making vCard data available to Semantic Web applications. Listing 2.1 shows a simple vCard containing information about the person *John Doe*, specifying his name, telephone number, email address, and the organization he is working for. The same information is represented as a vCard RDF graph in Listing 2.2.

```
1 BEGIN:VCARD
2 VERSION:3.0
3 N:Doe;John
4 FN:John Doe
5 TEL;WORK;VOICE:+41 99 555 5555
6 ORG:Example Corp.
```

[7] http://www.w3.org/TR/REC-html40/
[8] http://xerces.apache.org/xerces2-j/
[9] http://ical4j.sourceforge.net/
[10] http://www.dom4j.org/
[11] http://www.w3.org/TR/rdf-schema/
[12] http://www.w3.org/TR/owl-features/

```
EMAIL;INTERNET:john.doe@example.com
END:VCARD
```

Listing 2.1: A simple vCard.

```
<?xml version="1.0"?>
 <rdf:RDF xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:vCard = "http://www.w3.org/2001/vcard-rdf/3.0#">
  <rdf:Description rdf:about = "http://seal.ifi.unizh.ch/people/doe">
    <vCard:FN>John Doe</vCard:FN>
    <vCard:N rdf:parseType="Resource">
      <vCard:Family>John</vCard:Family>
      <vCard:Given>Doe</vCard:Given>
    </vCard:N>
    <vCard:ORG>Example Corp.</vCard:ORG>
    <vCard:TEL rdf:parseType="Resource">
      <rdf:value>+41 99 555 5555</rdf:value>
      <rdf:type rdf:resource="http://www.w3.org/2001/vcard-rdf/3.0#work"/>
      <rdf:type rdf:resource="http://www.w3.org/2001/vcard-rdf/3.0#voice"/>
    </vCard:TEL>
    <vCard:EMAIL rdf:parseType="Resource">
      <rdf:value>john.doe@example.com</rdf:value>
      <rdf:type rdf:resource="http://www.w3.org/2001/vcard-rdf/3.0#internet"/>
    </vCard:EMAIL>
  </rdf:Description>
</rdf:RDF>
```

Listing 2.2: vCard data represented in RDF.

2.3.2 FOAF

The *Friend of a Friend (FOAF)* project[13] is about creating a web of machine-readable Websites describing people, the links between them and the things they create and do, using an RDF-based data format [BM05]. Using the FOAF vocabulary, a person can provide information about herself, such as her name, interests and Website, and link to other people's FOAF data using the `<foaf:knows>` property.

2.3.3 iCal

iCalendar is a standard for calendar data exchange as defined in RFC 2445 [Soc98a]. The format of iCalendar data resembles the vCard format, and iCalendar data is also stored as unformatted ASCII data.
The W3C created a specification for an RDF representation of the iCalendar format, the *iCal* ontology [Con05]. The work on this was still in progress at the time of writing, as there was no final version of the iCal ontology published.

[13] `http://www.foaf-project.org/`

2.4 RDF Source Reader

In order to process the RDF data by the respective reader plugin, at first a Jena model has to be built from the data. After the user has specified a local RDF file or the URL of an RDF-annotated Website as source, the `RDFInputReader` class checks if the specified RDF source can be assumed to be a plain RDF file, by trying to match the file extension of the source with any extension defined in the `Config.MANIFEST_EXTENSION` constant. If no matching extension is found, `RDFInputReader` assumes the source being an HTML file having RDF content linked or embedded in it. It subsequently hands the URL of the source to the `HtmlReader` class which then tries to extract the RDF content embedded in the HTML file, or retrieving the linked RDF source.
`HtmlReader` uses *Xerces*, see Section 2.2, to parse the HTML file, supporting two different methods of annotating a Website using RDF data:

- Embedding the RDF data using the `<script>` tag, as the example in Listing 2.3 shows.
  As there was no published standard document on the subject of embedding RDF in HTML available from the W3C at the time of writing, the parser was implemented to support the method using the `<script>` tag as described in [Pal02]. Although RDF data embedded in HTML using the `<script>` tag is ignored by a Web Browser, the HTML document containing the RDF data no longer adheres completely to the XHTML standard, and as such the validation of the document using the W3C Markup Validation Service[14] fails. See Section 3.2.2 for proposals on how to extend the capabilities of the Semantic Clipboard to process embedded RDF data in the future.

- Linking to an external RDF source using the `<link>` tag, as the following line of code shows:
  `<link rel="meta"type="application/rdf+xml"href="content.rdf"/>`

`RDFInputReader` will then create a Jena model object from the retrieved RDF content, which will be used as session model.

```
<script type="application/rdf+xml">
<rdf:RDF xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:vCard = "http://www.w3.org/2001/vcard-rdf/3.0#">
  <rdf:Description rdf:about = "http://seal.ifi.unizh.ch/people/doe">
    <vCard:FN>John Doe</vCard:FN>
    <vCard:N rdf:parseType="Resource">
      <vCard:Family>John</vCard:Family>
      <vCard:Given>Doe</vCard:Given>
    </vCard:N>
  </rdf:Description>
</rdf:RDF>
</script>
```

Listing 2.3: RDF embedded in HTML.

[14] `http://validator.w3.org/`

2.5 Plugin Manager

The plugin architecture of the Semantic Clipboard requires the application to discover the plug-in classes that are currently present at the local configuration and to dynamically instantiate those classes needed to process the source data and paste it into target applications. Therefore, the `PluginMgr` class initializes and manages the configuration of the currently available plugin classes using the manifest files provided by the plugins, storing this information using an instance of the `PluginConfig` class.

2.5.1 Manifest Files

To register itself at the Semantic Clipboard, each plugin class has to provide a *manifest* file containing information about the plugin and its capabilities as RDF data. The information provided by all plugin types include the Java classname and the type of the plugin, being either a reader, data container or application plugin. Furthermore, data container and application plugins have to provide additional information as described in the following:

Reader Plugin A reader plugin implements the functionality to extract the data stored in the RDF source using a specific ontology vocabulary, e.g. vCard or iCal, and to create a data container object using the data. Therefore, the supported data container creates a dependency on a reader plugin. However, it is assumed that a reader plugin and the corresponding data container plugin are being jointly developed; thus, the compatibility of a reader plugin and its data container (and vice versa) is implicitly assumed. Consequently, a reader plugin has to provide the information about its classname and its plugin type in its manifest file.

Data Container Plugin A data container provides a Java object to store the data described by a specific ontology. E.g. the vCard data container provides means to store the personal data contained in a vCard file, such as name, addresses and telephone numbers of a specific person. This data will then be pasted into a target application by an application plugin, which has to be implemented according to the specification of the current data container.
To ensure compatibility between the application plugin and its supported data containers, the application plugins specify the name and accepted version number of each of its supported data containers in their manifest files. The data containers, on the other hand, specify its name and version number in their manifest files. For the sake of simplicity, a version number is represented by a single integer number, and the test whether an application plugin accepts the version number of a data container or not is done by simply testing the numbers specified by the application plugin and by the container plugin on equality. However, using the Semantic Clipboard in a real world environment, that would provide a larger number of plugins, more sophisticated methods for version controlling would have to be implemented.

Application Plugin An Application plugin acts as an interface between the Semantic Clipboard and desktop applications, implementing functionality to paste the contents of one or more data containers into a specific target application.
In addition to the version information of the supported data container plugins, which has to be provided as described in the paragraph above, application plugins specify also the name and a short description of the target application they are built for. This information is provided to the user when he has to choose an application to paste the extracted data into. As the target applications require, other than the Semantic Clipboard application itself, a specific operating system environment such as *Windows XP* or *Mac OS X*, the application

plugin manifests also specify the identifier of the platform required for its target application. Only application plugins specifying the same platform identification string as the one returned by the method `System.getProperty("os.name")` are being instantiated by the plugin manager.

As an example for an application plugin manifest file, the manifest of the Microsoft Outlook application plugin, providing functionality to paste vCard and iCal data into the Outlook application on Windows, as described in Section 2.6.3, is shown in Listing 2.4:

- The XML namespace `semclip` used by all plugin manifest files is associated with the URI `http://seal.ifi.unizh.ch/semclip-rdf/`, as defined in line 2.
- Lines 6 to 10 specify the classname and type of the plugin, the name and description of the target application, and the required platform for the target application.
- The supported data container plugins are each specified as individual resources in lines 16 et seq. and 21 et seq., defined as anonymous nodes carrying node ID "A0" and "A1" respectively, which are referenced in lines 12 and 13.

```
1  <rdf:RDF
2    xmlns:semclip="http://seal.ifi.unizh.ch/semclip-rdf/"
3    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
4    <rdf:Description rdf:about="http://seal.ifi.unizh.ch/semclip-rdf/
       plugins#OutlookAppPlugin">
5
6     <semclip:classname>
        ch.unizh.ifi.seal.semclip.plugins.applications.OutlookAppPlugin
        </semclip:classname>
7     <semclip:plugintype>AppPlugin</semclip:plugintype>
8     <semclip:appname>MS Outlook</semclip:appname>
9     <semclip:appdesc>Microsoft Office Outlook</semclip:appdesc>
10    <semclip:platform>Windows XP</semclip:platform>
11
12    <semclip:containers rdf:nodeID="A0"/>
13    <semclip:containers rdf:nodeID="A1"/>
14   </rdf:Description>
15
16   <rdf:Description rdf:nodeID="A0">
17    <semclip:containername>VCardContainer</semclip:containername>
18    <semclip:containerversion>1</semclip:containerversion>
19   </rdf:Description>
20
21   <rdf:Description rdf:nodeID="A1">
22    <semclip:containername>ICalContainer</semclip:containername>
23    <semclip:containerversion>1</semclip:containerversion>
24   </rdf:Description>
25 </rdf:RDF>
```

Listing 2.4: RDF manifest for the iCal application plugin.

2.5.2 Plugin Configuration

During the initialization process which is performed at the startup of the Semantic Clipboard, the `PluginMgr` class searches in the directory defined in the `Config.MANIFEST_PATH` constant for files having the extension defined in the `Config.MANIFEST_EXTENSIONS` constant using the `discoverPlugins()` method. The list of matching files is then being processed by the `ManifestReader` class using its `readManifest()` method, which creates a `ContainerInfo` object for each data container, and an `AppPluginInfo` object for each application plugin manifest found. These plugin info objects store and provide access to the plugin information specified in the RDF manifest files, allowing to parse the manifest file only once and retrieving the information about a plugin afterward efficiently from a Java object instead.
The `PluginConfig` Singleton class stores the information contained in the plugin manifest files using references to all existing `ContainerInfo`, `AppPluginInfo` and reader plugin objects. References to the latter are being stored directly in a Vector instead of a dedicated plugin object, as the reader plugins provide only a single value, its classname, in the manifest.

2.5.3 Plugin Instantiation

Depending on the type of the plugins classes, the plugin manager instantiates the different plugin types as follows:

Reader Plugin All reader plugins whose classnames are available in the plugin configuration are instantiated during the initialization process at the startup of the application. After the user has specified the RDF source, the plugin manager iterates over all available reader plugins to let them process the source model using the `processModel()` method, implemented by every reader plugin class.

Container Plugin For every recognized `rdf:about` subject URI in the source data, a container plugin object is created by the `processModel()` method of the respective reader plugin. Using the `Session.getContainersBySubject()` method[15], the reader receives a list of all existing data containers for the specified URI. If the list contains an instance of a data container the reader supports, it reuses this container, although existing data, contained in fields the reader writes into, might be overwritten. Otherwise, a new instance of the data container is created and the key-value pair of its URI and object reference is stored in the previously explained Hashtable in the `Session` object.

Application Plugin After the source data has been processed by the reader plugins, the list of applicable application plugins is created, containing application plugins matching all of the following criteria:

- The operating system required by the target application of the plugin matches the actual platform the Semantic Clipboard is currently running on. Determining the current platform version is done using the method described in Section 2.5.1 on page 12, where the application manifest file is explained.
- The list of existing data containers contains at least one instance of a container which is supported by the respective application plugin, regarding name and version of the container.

[15] Which is simply an accessor method for a Hashtable in the Session object, that stores Vectors containing references to the existing data container objects, using their subject URI as key.

2.6 Plugin Modules

A number of reader, data container, and application plugins have been implemented in order to demonstrate the use of the Semantic Clipboard to process data from various source ontologies into different desktop applications. The features of these plugins are explained in the following subsections. Figure 2.3 shows the inheritance structure of these plugin classes.

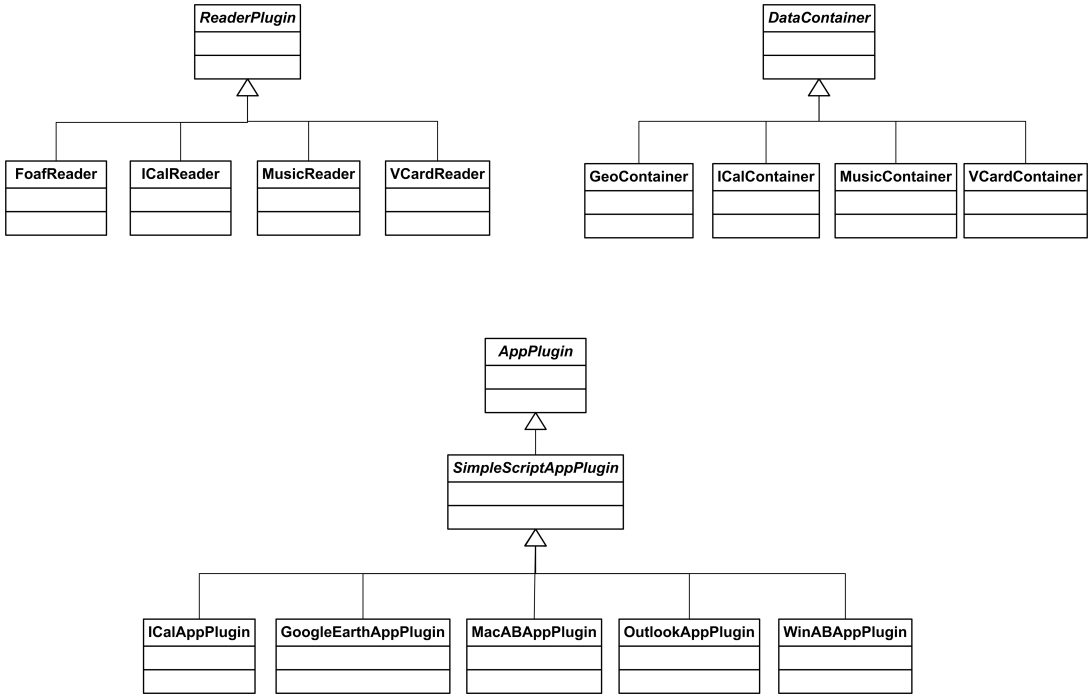

Figure 2.3: Inheritance structure of the reader, data container, and application plugins.

2.6.1 Reader Plugins

The class `ReaderPlugin` represents the superclass for all reader plugin implementations. It provides methods for frequently used Jena model processing tasks, such as retrieving the value of a literal or the content of an RDF Sequence, or determining if at least one of the namespaces used by the model, each representing a specific ontology, is supported by the reader. Its abstract method `processModel()` has to be implemented by any reader plugin subclass, containing the functionality to extract the ontology-specific data from the Jena model and pasting it into the matching data container objects.

iCal Reader As described in Section 2.3.3, iCal provides an ontology to describe events and related information. The `ICalReader` class extracts iCal data contained in the RDF source using the *SPARQL* [Con06d] query language, provided by the Jena framework. For each `VEvent` component found in the source, an `ICalContainer` object, see Section 2.6.2, is created to store the information about the event and any associated `VAlarm` components.
The W3C provides two slightly different RDF schemata for iCal, which differ only in the representation of timezone information. As the current ICalReader plugin ignores any

timezone-specific data, both of these schemata, represented by the namespaces as defined in the `ICAL_NS` and `ICAL_TZD_NS` constants respectively, are supported by the plugin. Values representing date and time in the source code have to conform to the *ISO 8601* date format, as specified in [Con98].

VCard Reader The `VCardReader` class implements functionality to extract data that adheres to the vCard ontology; see Section 2.3.1 for a description of this ontology. It looks for RDF subjects having the `vCard:FN` property and extracts the vCard data for this subject. The vCard data may be directly associated with the subject, or linked from another property resource, such as `DC:creator`, representing a *Dublin Core* RDF [Ini02] property, as Listing 2.5 shows. For each subject associated with a `vCard:FN` property, a `VCardContainer` container object, as described in Section 2.6.2, is created to store the vCard information of the subject. If a `vCard:GEO` property is found, containing the geographical coordinates of the described person, the coordinates are stored in a separate `GeoContainer` container object, as described in Section 2.6.2 on page 17.

```
1 <DC:creator rdf:parseType="Resource">
2   <vCard:FN>John Doe</vCard:FN>
3   <vCard:N rdf:parseType="Resource">
4     <vCard:Family>Doe</vCard:Family>
5     <vCard:Given>John</vCard:Given>
6   </vCard:N>
7 </DC:creator>
```

Listing 2.5: Dublin Core Metadata containing vCard data.

FOAF Reader The `FoafReader` plugin supports parts of the FOAF ontology, as described in Section 2.3.2, whereas some of the properties describing a person, but no references to other persons, are recognized by the plugin. Due to the close resemblance of these FOAF properties to some properties of the vCard ontology, a `VCardContainer` object is used to store the extracted FOAF data. In doing so, each value of a FOAF property is mapped onto its respective vCard property, as Table 2.1 shows.

FOAF	**vCard**
firstName	Given
surname	Family
nick	NICKNAME
mbox	EMAIL

Table 2.1: Mapping of FOAF to vCard properties.

Music Reader Developed as a first prototype of a reader plugin, the `MusicReader` implements just a simple vocabulary to describe the name of the artist and the title of a song, rather than an existing standard ontology vocabulary. To store the data extracted from the RDF source, an instance of the `MusicContainer` object, as described in Section 2.6.2 on page 17, will be used.

2.6.2 Data Container Plugins

Any data container plugin has to be a subclass of the `DataContainer` superclass. The only requirement to its subclasses is to implement the abstract method `toString()`, which itself is

inherited from the `Object` class. As return value of this method, every data container object should return a String serialization of its contents that follows the specific syntactic and formatting requirements of the implemented ontology, e.g. the `VCardContainer` class described in the following section returns a String representing the contents of a `.vcf` vCard file.
Using this String representation, an application plugin may paste the data contained in a data container object into a target application simply by creating a file containing the return value of the `toString()` method of the particular data container, which is then passed on to the application. If there is no specific serialization standard for the ontology implemented by the data container, the method may also return an empty or any other descriptive String value.
Besides this requirement, a data container class may implement any other functionality needed to store and retrieve the specific values of its ontology vocabulary.

iCal Container The `ICalContainer` plugin provides functionality to store and retrieve iCal event data. It uses an *iCal4j* `Calendar` object to store and process the iCal data internally. Although the iCal specification allows more than one `VEvent` component within an iCal RDF file, a separate container object has to be used for every VEvent component, as an event has no individual identification within a Calendar object, and therefore could not be accessed directly.
Table 2.2 lists the subset of iCal event properties that are supported by the plugin. In addition, alarm components may be added to an event, allowing to notify the user about an upcoming event.

Property	Description
`dtstart, dtend, dtduration`	Values defining the beginning and end date, and duration time of the event. The start and either the end or the duration of an event have to be specified.
`uid`	A unique identifier String for the event. If not specified by the source, a random hexadecimal UID String will be generated using the `generateUID()` method.
`summary, description`	Text values to describe the event. While the summary provides a title for the event, the description may describe the event more complete.

Table 2.2: VEvent properties implemented by ICalContainer.

To allow the import of the contents of the container into *Microsoft Outlook*, some changes to the contents of the container may be necessary, as Outlook imposes some restrictions on iCal data in addition to the requirements defined in [Soc98a].
The method `makeMSOutlookReady()` modifies the contents of the container according to these restrictions, if needed by an application plugin such as `OutlookAppPlugin`, which is described in Section 2.6.3 on page 18. As Outlook refuses importing any iCal file that does not fulfill its requirements, this method makes the following changes to the contents of the data container:

- Alarm components having a negative value for its trigger property are removed from the event, as Outlook does only support positive values for the alarm trigger property.
- Alarm components having the value for its trigger property formatted as a `DateTime` value, representing an absolute date, rather than a `Duration` as required by Outlook, are removed from the event.

The other requirements, namely inserting a `METHOD:PUBLISH` property into the vCalendar object, inserting a `UID` for every event component, and adding a `DTSTAMP` property for each

event object, are taken care of by the `fillICalContainer()` method and the constructor of iCal4j's `VEvent` class, respectively.

vCard Container The `VCardContainer` plugin implements a representation of a vCard object, supporting a large subset of the vCard properties specified in [Soc98b]. As at the time of writing no Java framework to store and process vCard data, similar to the iCal4j framework for iCalendar data, was available, the properties are stored internally using inner classes, each of them representing a vCard property. Table 2.3 lists the implemented vCard properties and its meaning.

Property	Description
`UID`	Unique identifier of the vCard.
`FN`	Formatted Name of the person.
`N`	Name components (Family Name, Given Name, Additional Names, Honorific Pre- and Suffixes).
`NICKNAME`	Nickname of the person.
`BDAY`	Birthday of the person.
`ADR`	Postal addresses.
`LABEL`	Formatted text corresponding to delivery address.
`EMAIL`	Email addresses.
`TEL`	Telephone numbers.
`GEO`	Geographical coordinates.
`ORG`	Organizational name and units.
`ROLE`	Occupation, or business category.
`TITLE`	Job title, functional position or function.

Table 2.3: vCard properties implemented by VCardContainer.

Geo Container A `GeoContainer` object stores information about a geographical coordinate, represented by a latitude, longitude, and a summary and description text. The `toString()` method returns the String serialization of its content, using the *Keyhole Markup Language*[16] XML grammar. To create the XML output, the functionality provided by the dom4j library is used.

Music Container The `MusicContainer` class provides a data container to store information about a music track, namely the name of the artist and the title of the song.

2.6.3 Application Plugins

Application plugins, implemented as subclasses of the `AppPlugin` class, are designed to act as interfaces between the contents of a data container and a desktop application. Therefore, they are developed to support a specific version of one ore more data container plugins and a specific desktop application.
The `AppPlugin` superclass implements utility methods usable by any application plugin, such as retrieving creating an `ImageIcon` object from an icon file, representing the icon of the target application, and providing a common used `TitledBorder` object. The following abstract methods have to be implemented by any application plugin subclass:

[16] `http://earth.google.com/kml/kml_intro.html`

- `showUI()` has to implement the functionality to show the application-specific user interface inside a Wizard panel component.
- `pasteData()` Has to provide the means to paste the content of the data container chosen by the user into the target application.

For application plugins that paste the data into the target application using a temporary file and command line scripting, the abstract `SimpleScriptAppPlugin` subclass of the `AppPlugin` class provides functionality used in this case. Its `createFileName()` method returns a filename to be used for a temporary file, using a specified prefix and suffix. `execSimpleScript()` pastes the contents of the specified data container using the container's `toString()` method into a temporary file and executes the provided command line script, which calls the target application to open the temporary file. Any errors occuring during this process are displayed on the user interface.
If a target application is available for different platforms and the process to import data into the application differs only slightly between the platforms, a single application plugin may be implemented to support multiple platforms, as the `GoogleEarthAppPlugin`, decribed on page 19, does for example.

ICal Application The Apple *iCal* personal calendar application[17] comes bundled with the Mac OS X operating system and is, therefore, available to all users of this platform.
The `ICalAppPlugin` class shows a list of the iCalendar events contained in the existing `ICalContainer` objects to the user. The user can display the start and end time, and the description of the event, as well as a boolean value indicating whether any alarms are associated with the event, by selecting an event from the list and clicking on the *Preview* button provided by the plugin user interface.
The data of the selected event is pasted into the iCal application using a temporary `.ics` file that is imported into the application using a command line script.

Mac Address Book Being the integrated address book application for Mac OS X, the *Address Book* application[18] is the standard address managing tool on the Mac platform.
The user interface of the `MacABAppPlugin` class is similar to the one of the iCal plugin, showing a list of the vCard components contained in the existing `VCardContainer` objects to the user. A preview of the selected vCard, showing the UID, formatted name, name components, and the first specified email address of the person, is displayed to the user after clicking the *Preview* button.
The data of the selected vCard is pasted into the address book application using a temporary `.vcf` file that is imported into the application using a command line script.

Windows Address Book The Windows address book[19] is a simple application to create and manage contact entries and is integrated in the current Windows distribution.
Its corresponding application plugin, the `WinABAppPlugin` class, implements a basic user interface, showing buttons with the names of the persons represented by the currently existing vCard container objects as its titles. After the user has clicked on any of the buttons, the vCard data is pasted into the address book application using a temporary `.vcf` file that is imported into the application by executing a command line script.

Microsoft Outlook *Microsoft Outlook*[20] is one of the most widely used personal information manager application and is part of the *Microsoft Office* suite. Besides an email client and other

[17] `http://www.apple.com/macosx/features/ical/`
[18] `http://www.apple.com/macosx/features/addressbook/`
[19] `http://msdn.microsoft.com/workshop/wab/overviews/wabovw.asp`
[20] `http://www.microsoft.com/outlook/`

features, it also provides contact management and calendaring functionality.
The application plugin for Outlook, the `OutlookAppPlugin` class, implements a user interface similar to the one of the Windows address book plugin. It displays the vCard and iCal data contained in the current `VCardContainer` and `ICalContainer` objects as buttons, showing the name of the vCard object or the summary of the iCal event, respectively, as button titles. Clicking on one of the buttons pastes the selected data as address book entry or calendar event into the Outlook application, using a temporary `.vcf` file for vCard data and a temporary `.ics` file for events, respectively.
Before being pasted into the target application, iCal data is processed using the `ICalContainer.makeMSOutlookReady()` method, which renders the content of an `ICalContainer` object into an Outlook compatible form, as described in Section 2.6.2 on page 16.

Google Earth *Google Earth*[21] is a virtual-globe application, allowing the user to display a satellite image of any point on the earth surface.
The user interface of the `GoogleEarthAppPlugin` application plugin consists of buttons, each labeled with the summary text of the existing `GeoContainer` data containers. After the user has clicked on one of the buttons, the coordinates of the selected location is pasted into the Google Earth application, which displays the location on its virtual globe. A temporary `.kml` file, containing the coordinates and its description in the Keyhole Markup Language XML format, is used to paste the data into the application by executing a command line script.

iTunes The Apple *iTunes* application[22] is a music jukebox program with an integrated online music store.
Using the `ITunesAppPlugin` class, a music track whose artist's name and song title are contained in a `MusicContainer` data container, can be played in the iTunes application. The application plugin user interface displays buttons, each labeled with the name of the artist and the title of the song. When the user clicks on one of the buttons, iTunes starts to play the specified song, if a corresponding music file is available in the local iTunes library. As the plugin uses the *Apple Script*[23] script language to communicate with the application, only iTunes on the Mac platform is supported by the plugin.

2.7 User Interface

The user interface of the Semantic Clipboard is made of Java Swing components, showing the main window which contains the Wizard interface.

2.7.1 Main Window

The main application window of the Semantic Clipboard is a single `JFrame`, which acts as parent frame for the Wizard dialog window and as container for the menu bar. Using the items provided by the menu bar, the user may restart and display the Wizard dialog, display the about dialog, and exit the application. Any other program interaction is done within the Wizard dialog, as described in the following section.

[21] `http://earth.google.com/`
[22] `http://www.apple.com/itunes/overview/`
[23] `http://developer.apple.com/applescript/`

2.7.2 Wizard Interface

The Wizard user interface guides the user through the steps of the work flow of the Semantic Clipboard. The Wizard consists of a single dialog window that displays the respective user interface for the current stage. Using the *Next* and *Back* buttons, the user navigates through the work flow, which may be exited at any time by clicking the *Cancel* button. Figure 2.4 shows the main window containing the Wizard dialog, as it is displayed after the initialization process of the application is completed.
An existing Wizard framework [Eck05], implemented as a Java Beans component, has been reused to realize the Wizard interface, which reduced the total effort required to develop the interface, compared to the alternative of developing an new Wizard framework from scratch. The development work to be carried out included the modification and improvement of the Wizard functionality, and the creation of customized Panel components to suit the specific needs of the Semantic Clipboard application.
The Model and Controller components of the *Model-View-Controller* pattern implemented by the Wizard framework are provided by the `WizardModel` and `WizardController` classes, respectively, the Controller being completed by individual `WizardPanelDescriptor` classes for each panel to be displayed. The View of the Wizard, represented by three Wizard panels is implemented using subclasses of the `WizardPanel` class.

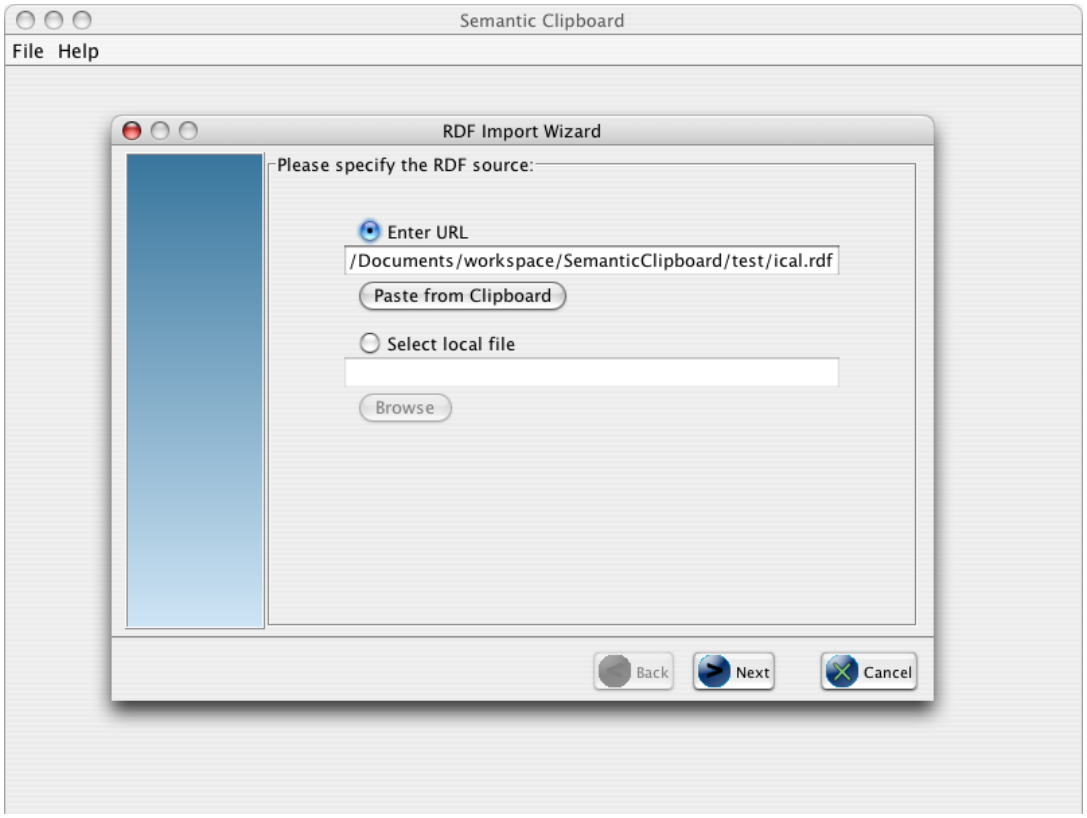

Figure 2.4: Main application window showing the Wizard interface.

2.8 Workflow

To give an impression of the workflow of the Semantic Clipboard user interface, the following subsections explain the steps necessary to select an RDF source, choose the target application, and finally paste the data into the application.
The RDF source used in this example consists of an HTML document containing embedded RDF data, as shown in Listing 2.6. The RDF data in the document describes the name and geographical position of a person in lines 10 to 17, and the artist and title of a music track in lines 19 to 22.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
4  <head>
5  <title>Embedded RDF</title>
6  <script type="application/rdf+xml">
7  <rdf:RDF xmlns:semclip="http://seal.ifi.unizh.ch/semclip-rdf/"
8         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
9         xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#">
10   <rdf:Description rdf:about="http://seal.ifi.unizh.ch/people/doe">
11     <vCard:FN>John Doe</vCard:FN>
12     <vCard:N rdf:parseType="Resource">
13       <vCard:Family>John</vCard:Family>
14       <vCard:Given>Doe</vCard:Given>
15     </vCard:N>
16     <vCard:GEO>8.477225834570831,47.41419165048976</vCard:GEO>
17   </rdf:Description>
18   <rdf:Description
19    rdf:about="http://seal.ifi.unizh.ch/semclip-rdf/examples/music#keane">
20      <semclip:trackartist>keane</semclip:trackartist>
21      <semclip:trackname>somewhere only we know</semclip:trackname>
22   </rdf:Description>
23 </rdf:RDF>
24 </script>
25 </head>
26 <body>
27 <p>This HTML document contains embedded RDF data.</p>
28 </body>
29 </html>
```

Listing 2.6: An HTML document containing embedded RDF data.

2.8.1 Specifying the RDF Source

As the current implementation of the Semantic Clipboard does not yet support an integration into a Web Browser application, see Section 3.2.1 suggesting how this could be done in the future, the user has to specify the RDF source manually.

Using the first panel of the Wizard user interface, as shown in Figure 2.5, the user specifies the RDF source, represented by a plain RDF file or an HTML file with embedded or linked RDF content. The location of the source is specified either as a URL, or by choosing a local file using a `FileChooser` dialog. If the current content of the system clipboard is a text representing a URL, the user may paste this URL into the corresponding text field by clicking the *Paste from Clipboard* button.
After specifying the location of the RDF source, the user clicks the *Next* button, which causes the Semantic Clipboard to retrieve the specified RDF source, iterate through the reader plugins to process the data, and create data container objects from it, as described in Sections 2.4 and 2.5.3.

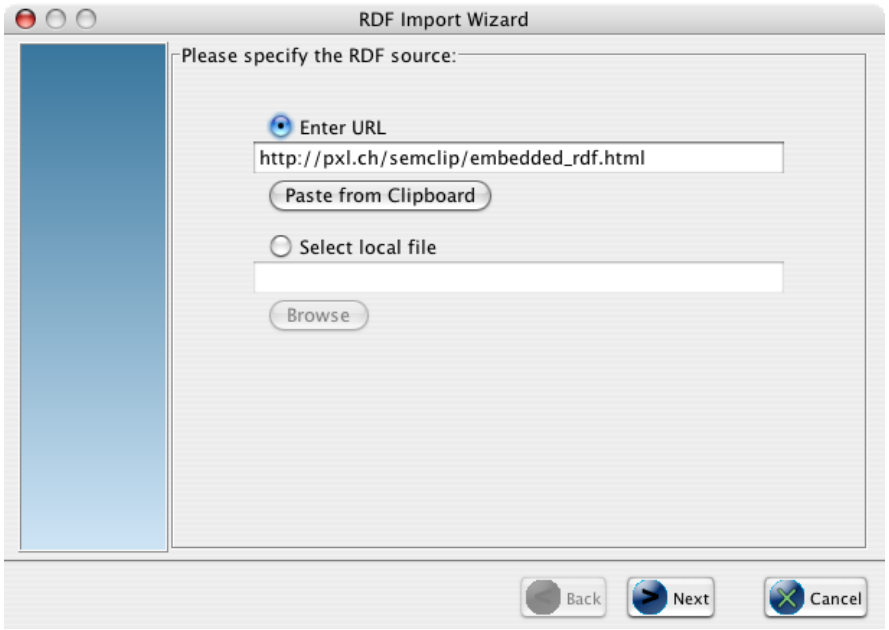

Figure 2.5: Specifying the RDF source in the first Wizard panel.

2.8.2 Selecting the Target Application

The second Wizard panel displays a progress bar, showing the progress of retrieving and processing the RDF source to the user. Below the progress bar, the list of application plugins that are able to handle the current source data is displayed, as Figure 2.6 shows. The description of a plugin, as defined in its manifest file, see Section 2.5.1, is displayed for the selected plugin upon clicking the *Get Plugin Description* button.
In this example, the Google Earth, Mac Address Book, and iTunes applications would be possible target applications for the data contained in the RDF source and are, therefore, shown in the plugin list.
After the user has selected the application plugin to handle the data and has clicked on the *Next* button, the Semantic Clipboard instantiates the chosen application plugin and lets the plugin display its user interface.

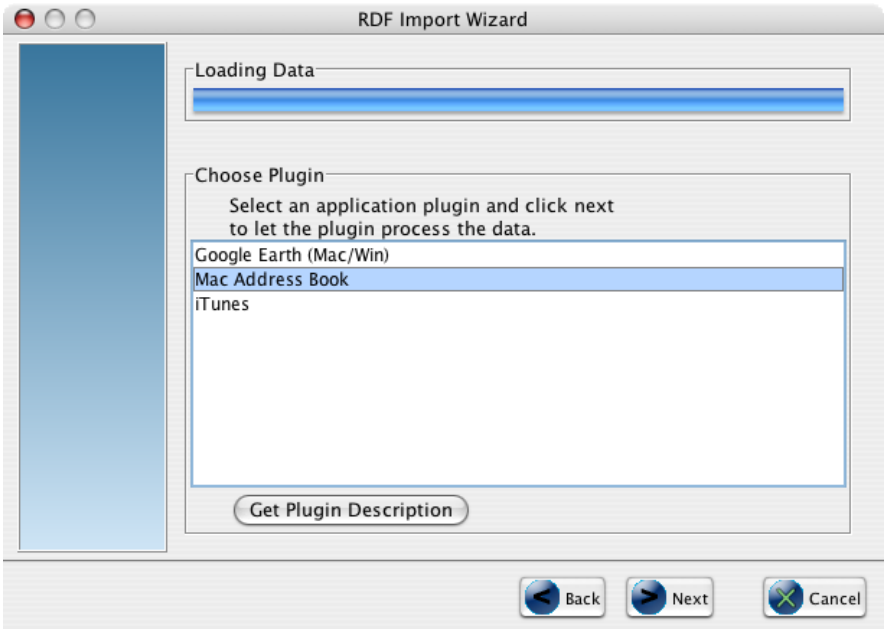

Figure 2.6: Selecting the target application in the second Wizard panel.

2.8.3 Selecting the Data to be pasted

In the third Wizard panel, the specific user interface provided by the application plugin, as selected in the previous step, is displayed. As each application plugin implements its specific user interface, the layout and contents of this Wizard panel varies, depending on the chosen application plugin. Therefore, the screenshots shown in Figures 2.7 and 2.8 are only one possible example of such a plugin-specific user interface.
Figure 2.7 shows the Wizard displaying the user interface of the Mac Address Book application plugin. The plugin displays a list containing the vCard components extracted from the source, in this case a vCard about the person *John Doe*. Clicking on the *Preview* button displays a message box showing the summary of the contents of the selected vCard component, as Figure 2.8 shows.

Clicking on the *Paste Data* button results in the application plugin pasting the data into its target application. In the case of this example, the application plugin pastes the selected vCard data into the Mac Address Book application, resulting in the creation of a new address entry.

After the data is successfully pasted into the target application, the user may select and paste other data into the same application. He may also click on the *Back* button to return to the second or first Wizard panel to choose another target application or specify a different RDF source. Clicking on the *Finish* button closes the Wizard user interface, which may be displayed again by selecting the *Show Wizard* command from the *File* menu bar.

2.9 Installation

The enclosed CD-ROM provides a ZIP file (`semclip_bin.zip`) that contains a JAR archive of the Semantic Clipboard classes, as well as the plugin classes and manifest files, and all required external Java libraries. The contents of the ZIP file should be extracted into an empty folder, in the following referred to as `$SEMCLIP`. In order to run the Semantic Clipboard application, the

applicable method for the respective platform, as described below, should be used.

Mac OS X Using the console, the `run.sh` is executed by changing the current working directory to `$SEMCLIP` and typing the commands as shown in Listing 2.7.

```
1 chmod u+x run.sh
2 ./run.sh
```

Listing 2.7: Shell commands to start the Semantic Clipboard on Mac OS X.

Windows In an Explorer window showing the `$SEMCLIP` directory, the application is started by simply double-clicking on the `run.bat` batch file.

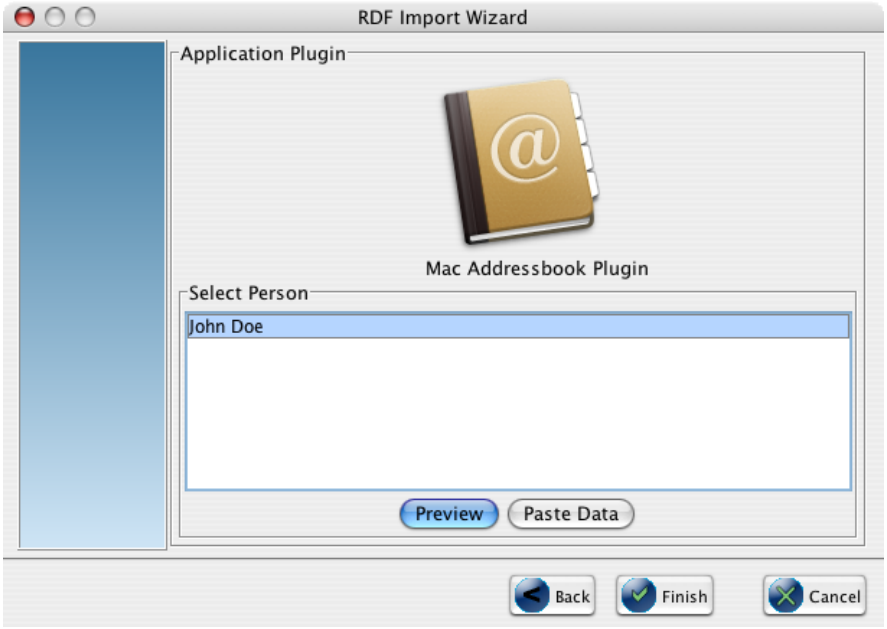

Figure 2.7: The third Wizard panel showing the data to be pasted into the target application.

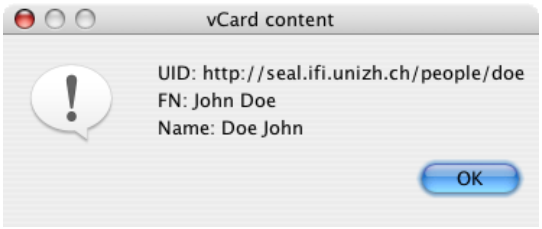

Figure 2.8: A message box displaying the contents of the selected vCard component.

Chapter 3

Conclusion and Future Work

3.1 Conclusion

In this thesis, the concept of the Semantic Clipboard, using an extensible plugin architecture and a Java implementation thereof, is presented.
The Semantic Web, a collaborative effort of a large number of researchers and the industry, initiated and led by the World Wide Web Consortium, provides a common framework to share data across application, enterprise, and community boundaries. The Resource Description Framework RDF, using XML syntax and the URI schema for naming, enables Semantic Web applications to compose metadata describing Web resources, or, by extending the concept of a resource, any other object of the real or virtual world.
Current clipboard applications, as the built-in clipboards of the Windows or Mac OS operating systems, allow the exchange of data between applications running on the same platform. Retaining the semantics of the data copied, however, is only supported within the boundaries of some application frameworks, as for data exchanged within the applications of the Microsoft Office suite. Expanding the range of data sources from desktop applications to any Website of the World Wide Web, the process of copying data from a Web resource and pasting it into a desktop application results in losing most of the semantics of the data, as the target application usually recognizes the pasted data as formatted or plain text only.
The Semantic Clipboard application extracts RDF data from accordingly annotated Websites, presents the user a list of target applications which are able to handle the data, and pastes the data chosen by the user into a target application, retaining the semantics of the data. The implemented plugin architecture uses three different types of plugin modules, which each of them can be developed individually to extend the range of supported source ontologies and target applications:

- Reader plugins, each implementing a specific ontology vocabulary, extract the supported ontology-specific data from the RDF source.
- Data container plugins provide functionality to store the extracted data, while keeping the specific syntactic and formatting standards of the particular ontology.
- Application plugins, acting as interface between the Semantic Clipboard and desktop applications, paste the data contained in the supported data container objects into a specific target application.

Using the Semantic Clipboard, metadata contained in an RDF annotation of a Website, for example describing the time, date, and description of an event, and the name and address of

the author, can be extracted, processed and pasted as an event entry into a calendaring desktop application, and as an address entry into a contact management application, respectively.

3.2 Future Work

Due to the extensible plugin architecture implemented in the Semantic Clipboard, additional plugin modules to support further source ontologies and target applications may easily being added to the current configuration. Besides this possible extensions, the integration of the Semantic Clipboard into a Web Browser application, and the extension of the range of supported methods to embed RDF data in HTML, are proposed.

3.2.1 Browser Integration

The current implementation of the Semantic Clipboard requires the user to manually paste the URL of an RDF annotated Website into a text field provided by the application user interface. Integrating the Semantic Clipboard into a Web Browser application would allow the Semantic Clipboard to be able to recognize Websites containing RDF metadata, which then could be automatically extracted and presented to the user, creating a "true" clipboard functionality. After extracting the source data, the user may chose the data he wants to paste into a specific target application, as it is the case in the workflow of the current implementation. A possible way to integrate the Semantic Clipboard into a Web Browser might be the *Mozilla Extensions* technology[1], allowing to develop add-on functionality for the Mozilla *Firefox* and *SeaMonkey* Web Browsers.

3.2.2 Embedded RDF Data

As the development work in this thesis focused on creating a set of plugins to support a number of source ontologies and target applications, the implemented functionality to extract RDF data embedded in HTML includes only a basic set of embedding methods, as described in Section 2.4. However, to expand the range of Websites whose RDF annotations are supported by the Semantic Clipboard, the `HTMLReader` class could be extended by implementing the ability to extract RDF content that is embedded in HTML documents using one of the following methods:

- Embedding RDF data in the HTML body, using the *RDFa* syntax as proposed by the W3C in [Con06a]. RDFa allows to define RDF properties directly in the HTML content, e.g. the code `<p>My name is <meta property="contact:fn">Jo Smith</meta></p>` displays the name *Jo Smith* to the visitors of the Website, and, at the same time, defines this name as the value of the `contact:fn` property, allowing Semantic Web applications like the Semantic Clipboard to extract this data and process it as contact information, rather than plain text.
- RDF data pasted into the `<head>` of a HTML document[2], which completes the embedding using the `<script>` tag as described in Section 2.4.

[1] `http://developer.mozilla.org/en/docs/Extensions`
[2] see "How do I put some RDF into my HTML pages?" at `http://www.w3.org/RDF/FAQ#How`

References

[AvH04] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, Cambridge, Massachusetts, USA, 2004.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001. http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[BM05] Dan Brickley and Libby Miller. FOAF Vocabulary Specification. Namespace Document, July 2005. http://xmlns.com/foaf/0.1/.

[Con98] The World Wide Web Consortium. Date and Time Formats. W3C Note, August 1998. http://www.w3.org/TR/NOTE-datetime.

[Con01] The World Wide Web Consortium. Representing vCard Objects in RDF/XML. W3C Note, February 2001. http://www.w3.org/TR/vcard-rdf.

[Con05] The World Wide Web Consortium. RDF Calendar Workspace. Website, April 2005. http://www.w3.org/2002/12/cal/.

[Con06a] The World Wide Web Consortium. RDFa Primer 1.0: Embedding RDF in XHTML. W3C Working Draft, May 2006. http://www.w3.org/TR/xhtml-rdfa-primer/.

[Con06b] The World Wide Web Consortium. Resource Description Framework (RDF). Website, June 2006. http://www.w3.org/RDF/.

[Con06c] The World Wide Web Consortium. Semantic Web. Website, June 2006. http://www.w3.org/2001/sw/.

[Con06d] The World Wide Web Consortium. SPARQL Query Language for RDF. W3C Candidate Recommendation, April 2006. http://www.w3.org/TR/rdf-sparql-query/.

[Eck05] Robert Eckstein. Creating Wizard Dialogs with Java Swing. Sun Developer Network, February 2005. http://java.sun.com/developer/technicalArticles/GUI/swing/wizard/index.html.

[Ini02] The Dublin Core Metadata Initiative. Expressing Qualified Dublin Core in RDF / XML. DCMI Proposed Recommendation, May 2002. http://dublincore.org/documents/dcq-rdf-xml/.

[Pai05] Reinhard Paizoni. The Semantic Clipboard. Master's Thesis, Technical University of Vienna, March 2005.

[Pal02] Sean B. Palmer. RDF in HTML: Approaches. Website, June 2002. http://infomesh.net/2002/rdfinhtml/.

[Soc98a] The Internet Society. Internet Calendaring and Scheduling Core Object Specification (iCalendar). Request for Comments, November 1998. http://www.ietf.org/rfc/rfc2445.txt.

[Soc98b] The Internet Society. vCard MIME Directory Profile. Request for Comments, September 1998. http://www.ietf.org/rfc/rfc2426.txt.