ESPRIT Project No. 25 338

Work package E

Personal Profiles

DE1: Survey

ID:	DE1Survey	Date:	07/01/98
Author(s):	Steve Battle	Status:	deliverable
Reviewer(s):		Distribution:	

Change History

Document Code	Change Description	Author	Date
DE1.1	First draft of document. No changes.	Steve Battle	19/11/97
DE1.2	Deliverable	Steve Battle	01/12/97
DE1.3	Added review of MONDO	Steve Battle	07/01/98
DE1.4	Added review of DOM	Steve Battle	19/01/98

PERSONAL PROFILES

1 Formats

2 Frameworks

Personal Profiles

The aim of this document is to survey the current state of the art in the storage and use of personal information. The quantity of data appearing in such a profile is generally small and highly irregular. Conventional database techniques, optimised as they are for large amounts of highly regular data, are therefore inappropriate. We look instead at an emerging body of standards designed for keeping track of so-called meta-data. Profiles act as a kind of catch-all for all the odd little pieces of data required in the followMe system. Meta-data gives us a systematic way of thinking about them.

Personal Information

Personal information is all about people and the resources they use. It may contain both public and private data. The personal profile would contain the following kinds of information:

User identification
Addressing properties
Security properties
Lifestyle properties
Service specifics
Agent specifics

We need a data format that can cope with this wide range of content.

1 Standards

Meta-Content Framework

MCF began life as a way of describing and cataloguing the content of web pages, hence the name meta-content; content *about* content. MCF emerged out of Apple's technology seeding programme as the HotSauce browser plug-in (aka project X) which allowed the user to view meta-data within a 3D navigation space. The ancestry of MCF includes knowledge representation languages such as CycL, KRL and KIF. KIF is the Knowledge Interchange Format usually associated with KQML, the Knowledge Query and Manipulation Language. KIF defined a core language with an extensible vocabulary based on frames. The goals of MCF are therefore:

- To establish a minimum *core* of properties.
- To make this core dynamically *extensible*.

Attempts at establishing this core include the 1995 Metadata Workshop, known as the `Dublin core'. This was intended to describe information about electronically stored documents, such as web pages. This kind of meta-data would assist the construction of indices for the internet, allowing greater precision than current key-word searches. Both MCF and the Dublin-core were designed to be independent of any particular syntax.

Subject	The subject area of the document.
Title	The name of the document.
Author	A person or agent responsible for the content of the document.
Publisher	An agency responsible for making the document available.
OtherAgent	Persons or organisations other than authors or publishers with some contribution.
Date	The date at which the document became available in its current form.
ObjectType	A category the document belongs to.
Form	The kind of machine readable format of the document.
Identifier	A unique reference for the document.
Relation	The document's relationship with another document.
Source	A document from which it is derived.
Language	The language in which the document is written.
Coverage	The spatial or temporal relevance of the document.

An MCF document defines a general purpose database, comprising:

- A set of nodes.
- A set of directed, labelled arcs.

Each arc is a directed relationship between a pair of nodes, having dinstinct source and target nodes. The arc is also labelled, but the label is just the name of another node. This node may belong to the core or it may be user-defined. The MCF core defines a set of *bootstrap nodes* which provide the primitives for constructing new nodes. In this example, these bootstrap nodes include the *domain* and *range* of a new relationship, and the *typeOf* relationship between nodes.

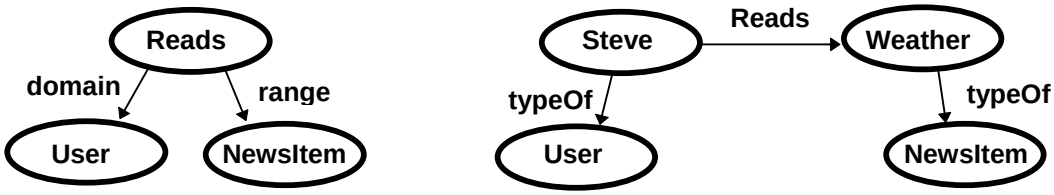


Figure 1: Defining and using a new MCF relationship

Versit's vCard and vCalendar

vCard

Where MCF focussed primarily on the description of documents, the VCARD personal data interchange (PDI) specification (1996) was designed as an electronic business card, containing information about the user and the organisation they work for. In the same way we hand out business cards to our colleagues, the vCard only contains data the user is happy to be made public, side-stepping the issue of security. A vCard contains properties relating to identification of the user, addressing for postal delivery, telecommunications including email, fax and phone, simple geographic data, properties relating to the organisation the user works for, and security. While the defined vCard properties overlap those required in followMe, it does not entirely cover the required functionality, particularly with respect to storing data about the services employed by the user. Because these service requirements are application specific there is a real need for extensibility. The following vCard fragment gives a flavour of vCard syntax.

```
BEGIN:VCARD
VERSION:2.1
FN: Steve Battle
N: Battle;Steven;Andrew;Dr.
END:VCARD
```

vCalendar

The vCalendar specification is a set of properties suitable for building diary objects. Like vCard, vCalendar is supported by multiple vendors. It is a container for scheduling and reminders. A schedule includes entities called *vEvents* which represent extended periods of time. They may have additional properties which define the details of a meeting, for example. Reminders include *vToDo* entities which aren't tied down to specific times, but might indicate a list of jobs that need doing within some stated interval of time.

Dates and times are represented as ISO 8601 strings, where the month is numeric and the time is 24 hour.

<YYYY><MM><DD>T<HH><MM><SS><type>

The recommended Universal time (UTC) is indicated by appending a 'Z' to the string. The start of the next millenium would be represented by "20000101T000000Z".

A period of time is represented according to the ISO 8601 standard as the following string, where square brackets indicate an optional item.

P[<years>Y][<months>M][<weeks>W][<days>D][T[<hours>H][<minutes>M][<seconds>S]]

vCalendar also supports ways of representing recurrent events using XAPIA's CSA specification.

<frequency><interval><frequency modifier>(<end date>|<duration>)

The frequency can be daily (D), weekly (W), monthly (M) or yearly (Y). The interval is a number indicating the number of days/weeks/months/years between recurrences. The modifer depends on the frequency used, and allows the rule to be more specific about the time of recurrence. For example you could specify that you get paid on the last Friday of every month. Finally we can specify either and end date or maximum number of occurrences that the rule should repeat for. The example below is a simple vCalendar entity.

```
BEGIN:EVENT
DTSTART:19971211T090000
DTEND:19971212T170000
ATTENDEE;ROLE=ATTENDEE:STATUS=CONFIRMED:Steve Battle
END:EVENT
```

Vcard and vCalendar entities can be linked together. We may want to associate a specific group of people (vCards) with a particular meeting, or designate an organisation as the host for that meeting. vCalendar provides a comprehensive set of calendar tags which look sufficiently powerful to support the kinds of diary activities envisaged in the followMe framework.

Appendices 1 and 2 contain a complete set of tags and attributes for vCard and vCalendar.

The eXtensible Markup Language: XML

The need for extensibility in MCF led its designer, R.V. Guha of Netscape Communications to propose that the MCF model is implemented in the eXtensible Markup Language, XML. XML is actually a subset of SGML, the Standard Generalised Markup Language in which more well known languages such as HTML are formally defined. However, while SGML is large and difficult to learn, the minimal features of XML were selected for ease of use. XML has the same <STRONG>look</STRONG> and <EM>feel</EM> as HTML but can be extended by providing a document type declaration (DTD), which provides a grammar for the body of the document. Both vCard and vCalendar are easily mapped onto XML. The vCard specification includes a suggested mapping from vCard to HTML forms. The attribute names in this mapping could in turn be mapped directly onto XML tags. The result of one possible mapping onto XML can be seen in appendix 3. A typical Document Type Declaration for appendix 3 is shown in appendix 4.

The eXtensible Markup Language (XML) is a simple language designed for the storage and transmission of information on the internet, controlled by the WorldWide Web Consortium (W3C). XML is actually a subset of SGML, the Standard Generalised Markup Language in which more generally known markup languages such as HTML are formally defined. Unlike SGML, XML is small and easy to learn. While HTML has generated a new level of consistency between web documents, it's function is confined to markup; describing the way things look. HTML does not do a good job at representing what is increasingly known as meta-data. This is additional information that sits alongside existing data, defining it's structure and emphasising important content for a variety of purposes. A well formed XML description comprises a hierarchical structure of paired tags, using the <tag>...</tag> syntax familiar from HTML. The structure is hierarchical in that the structure can be nested to any depth. In addition to this nesting, the opening tag may posess simple attributes. e.g. <TEL TYPE="FAX">...</TEL>. The tags one can use with HTML are fixed by big players like Netscape and Microsoft, whereas XML is extensible; putting power back into the hands of the users, allowing more creative use of tags with customised functionality. This freedom is balanced by moves to add optional extensions on top of basic XML. These additional layers don't change the language but provide standard sets of tags for common uses. Proposed extensions currently include:

- XML-TYPE Commonly used data types.
- XML-LINK (XLL) Hyper-links for XML.
- XML-STYLE (XSL) Style sheets for markup.
- CML Chemical Markup Language (an example of a domain specific extension).
- RDF Resource Description Framework.

Also, new W3C standards for privacy and profiling on the web are based on XML.

XML is well supported by Microsoft who provide a Java based XML interpreter. Microsoft is already using XML in anger with their Channel Definition Format which is used to specify details for server push technologies.

Resource Description Framework: RDF

RDF is the product of merging MCF with XML. It is a formalisation of the way that XML could be used to describe meta-data. Where XML follows an essentially grammatical structure, the RDF extensions are more suited to the description of graphical structures. Because a graph can be flattened into a grammatical structure in so many ways, this additional level of interpretation provides a consistency which may not be immediately apparent in the one dimensional XML.

The example below uses the graphical structure of RDF to describe an interface to a service acting as a middle tier between a database application and the database itself. It allows the application to construct a new query, adding columns to the projection and asserting appropriate selection conditions. When the query is complete, it may be executed, the results can be fetched one at a time and the appropriate columns can be read out. As you can see from the Interface definition below (described in CORBA IDL), the interface typing is represented but it fails to capture this behavioural aspect.

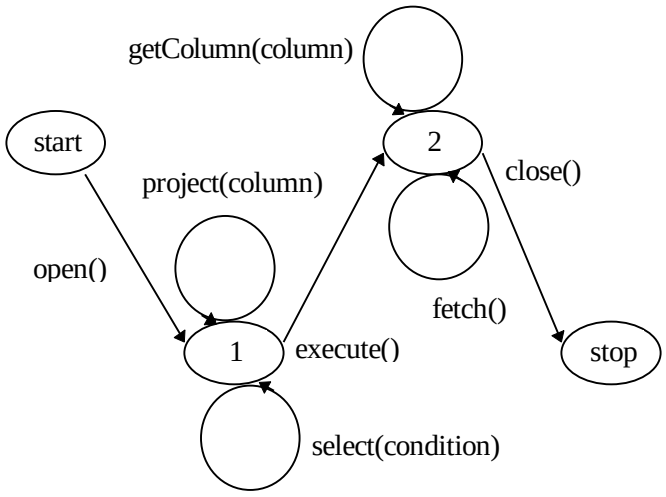
```
interface QueryBuilder {
  long open() ; // open a new query

  void project(in long handle, in string exp) ; // add column to the query
  void select(in long handle, in string exp) ; // add a condition to the query
  void execute(in long handle) ; // perform the query

  void fetch(in long handle) ; // fetch the next row
  string getColumn(in long handle, in string exp) ; // return column value
  void close(in long handle) ; // close an existing query
} ;
```

The required behaviour can be described as a finite state graph, where each node represents a state the service may be in at any time, and the arcs drawn between them represent method invocations on the service. There are many ways this graph could be flattened into XML. Each node, and the arcs attached them could be defined separately. The XML below uses the idea of embedded assertions so that we can encode the graph moving from left to right. Note that it would be equally valid to encode the graph going from right to left. Because of the huge variety of such encodings, RDF must be parsed internally into this graph structure, otherwise operations like graph matching would fail. The RDF is deliberately untyped as the finite state graph could be used in conjunction with the typed interface definition above.

```
<RDF:assertions>
  <RDF:resource id="start">
    <open>
      <RDF:resource id="1">
        <project:column>1</project>
        <select:condition>1</select>
        <execute>
          <RDF:resource id="2">
            <fetch>2</fetch>
            <getColumn:column>2
            </getColumn>
            <close>
              <RDF:resource
               id="stop"/>
            </close>
          </RDF>
        </execute>
      </RDF>
    </open>
  </RDF>
</RDF>
```



As a concrete representation of MCF, RDF provides a general way of representing graph based information, including logical structures which could support formal deductive reasoning.

2 Frameworks

MONDO

The MONDO framework does not promote any particular standard for the storage of meta-data but proposes a framework within which this information can be used to build object-oriented systems. Reference versions of MONDO in fact use a combination of SGML/XML and OML (Object Modelling Language).

MONDO emphasises the difference between what it calls *recipes* and *domain objects*. The recipe is crucially a simple text file which encodes the information necessary to build a domain object. However, the recipe doesn't completely determine the resulting domain object. An analogy is drawn with the way a chef uses a real recipe. The recipe certainly defines what the dish will turn out to be, but the skill of the chef in interpreting the recipe is at least, if not more important in determining the quality of the result. The chef corresponds to a procedure called the object builder which is a particular class of object that knows how to interpret recipe data.

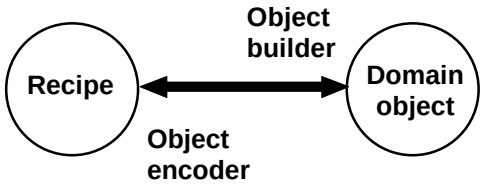
Each instruction, or element, of a recipe codes for a single domain object. Recipes may be hierarchically organised so that instructions can be nested within each other. The building process returns a single object corresponding to the top-level recipe, containing nested objects corresponding to the output of the builders for each sub-instruction.

```
Recipe
      Instruction
      Instruction
            Instruction
            Instruction
      Instruction
```

The building process can be assisted by meta-data containing explicit links to implementation classes for distinct types of entity in the *domain model*. The example below associates a particular class *MyDate* with the *Date* type. Note that the syntax, though similar to XML is actually OML.

```
<Implementation
      type = "Date"
      class = <Class
            name = "MyDate" version = 1.0
            bytecodes = <Binary encoding= "hex" data= "cafebabe…">
>>
```

The building process can be reversed if the domain object supports introspective processes which can be used to generate a recipe which can later be used to reconstruct the object.



The concepts found in MONDO are remarkably close to the ideas in currency for the representation of profiles using some meta-data language, and their relationship to profile objects within an object oriented system. While the MONDO framework may not be used explicitly, it can still be seen in spirit.

Document Object Model

The Document Object Model, designed by the W3C DOM working group, is a programmatic interface to the contents of a document. The API is introduced on two levels.

Level 0

This level provides a basic interface for HTML and XML documents as might be generated by a non-validating parser. A document defines a tree of objects corresponding to the nesting of markup tags within the document text. The nodes which can be found in this tree are as follows.

Document
The entire HTML or XML document. Contains a root element.
eg. For an HTML document this is the html tag:- <html>...</html>

Element
A tagged element containing a number of attributes and sub-nodes.
eg. The head element contains a title sub-node:- <head> <title>Document Object Model</title> </head>

Attribute
A named value specified in the opening tag.
eg. A white bgcolor is an attribute of the body element:- <body bgcolor=#ffffff> ... </bgcolor>

Text
Plain text containing no markup.
eg. The text "Document Object Model" of <title>Document Object Model</title>

Comment
plain text comment.
eg. <!-- Gort, Klaatu Birada Niktoh -->

PI
This is a processing instruction intended for the document parser.
eg. <?XML version=1.0 ?>

In addition to methods for accessing data specific to each class of node, each of these inherits from an abstract Node class which provides basic functionality to navigate up and down the tree.

Level 1

HTML

This level corresponds to the access a scripting language such as JavaScript or VBScript would have to the HTML document it resides within. The root Document node procides the functionality of JavaScript's Document object. This includes short-cuts to forms and images which are held in arrays as immediate properties of the Document. Among the attributes defined by the generic HTMLElement are event handler strings such as 'onClick' and 'onKeyDown', which define an action to be taken (a script) on a given user event. A separate get and set style interface (ie. similar to the Java Beans pattern) exists for access to documents from Java.

XML

While the HTML DOM specification can be defined for a fixed set of markup tags, the XML interface must be extensible. XML documents may be defined with respect to a Document Type Definition (DTD) which specifies additional rules for determining if an XML document is well formed. This information is accessible via the DocumentType interface.

A standard interface at this level is essential for the commercial adoption of XML parsers, and will be invaluable in

the drive towards a cross-language scripting architecture.

Conclusions

The XML language is emerging as a standard for representing profile data. By conforming to these standards we increase the chances of re-using and exploiting individual components of the followMe system. The aim is to design a core set of properties appropriate to the followMe pilot domains, and to implement the appropriate profile classes.

References

ChiMu Corporation, MONDO, <http://www.chimu.com/projects/mondo>

The DARPA knowledge Sharing Effort, <http://logic.stanford.edu/sharing/knowledge.html>

Guha R.V., Meta-Content Framework Using XML, <http://www.w3.org/TR/NOTE-MCF-XML-970624/>, June 97

Microsoft, Specs & Standards, <http://www.microsoft.com/standards/xml/>

Versit Consortium, vCard: The Electronic Business Card, <http://www.imc.org/pdi>, January 1997

Stuart Weibel et al, OCLC/NCSA Metadata Workshop Report,
<http://www.oclc.org:5046/oclc/research/publications/weibel/metadata/dublin_core_report.html>, March 1995

W3C, Document Object Model, <http://www.w3.org/DOM>, December 1997

W3C, Extensible Markup Language (XML), <http://www.w3.org/TR/WD-xml-970807>, August 1997

Appendices

The conventions used in these appendices are as follows.

For property listings (appendices 1 and 2):
Property tags are listed in the leftmost column.
Tag Attributes are listed below the tag in the second column in UPPER-CASE.
Tag Sub-elements are listed below the tag in the second column.
Descriptions of tags and attributes appear in the third column.
Values between braces {} indicate possible attribute assignments.

Appendix 1: vCard properties

Identification properties:

FN		Formatted name formatted for display
N		Structured representation of name
	Family	Family name
	First	Given name
	Middle	Additional names
	Prefix	
	Suffix	
PHOTO		Photograph of an individual
	TYPE	One of {GIF, TIFF, JPEG, ..}
BDAY		Birthdate (ISO 8601)
SOUND		Audio or phonetic representation of name
	TYPE	One of {WAVE, PCM, AIFF, ...}
URL		Source of additional personal information
UID		Globally unique identifier

Delivery addressing properties:

ADR		Address (based on X.520 PO Box attribute)
	TYPE	One or more of {DOM, INTL, POSTAL, PARCEL, HOME, WORK}
	Street	Street address
	Locality	City
	Region	State/province
	Postal Code	
	CountryName	
LABEL		Delivery address (based on X.520 Postal address)
	TYPE	One or more of {DOM, INTL, POSTAL, PARCEL, HOME, WORK}

Telecommunications addressing properties:

TEL		Telephone number
	TYPE	One or more of {PREF, WORK, HOME, VOICE, FAX, …}
EMAIL		Electronic mail
	TYPE	One of {AOL, CIS, SMTP, ...}
MAILER		email software

Geographical properties:

TZ	Time Zone (ISO 8601), offset from UTC
GEO	Geographic position (latitude, longitude)

Organisational properties:

TITLE		Job title (based on X.520 title)
ROLE		(based on X.520 Busines Category explanatory)
LOGO		Logo image
	TYPE	One of {GIF, TIFF, JPEG, ..}
AGENT		separately addressable agent
ORG		(based on X.520 Organisation name attribute)
	Name	Organisation name
	Unit	Organisation unit

Explanatory properties:

NOTE	Comment (based on X.520 description)
REV	Last revision (ISO 8601)
VERSION	vCard version number

Security properties:

KEY		Public encryption key
	TYPE	One of {X509, PGP, ...}

Appendix 2: vCalendar properties

LANGUAGE	String consistent with RFC 1766
DAYLIGHT	Daylight savings rule
GEO	Geographic position (latitude, longitude)
PRODID	Product identifier, calendar creator
TZ	Time Zone (ISO 8601), offset from UTC
VERSION	vCalendar version number
ATTACH	object reference for an event
ATTENDEE	an individual associated with an event
Role	One of {ATTENDEE, ORGANISER, OWNER, DELEGATE}
Status	One of {ACCEPTED, NEEDS ACTION, SENT, TENTATIVE, CONFIRMED, DECLINED, COMPLETED, DELEGATED}
RSVP	One of {YES, NO}
Expect	One of {FYI (for your info), REQUIRE, REQUEST, IMMEDIATE}
AALARM	Audio alarm
TYPE	One of {PCM, WAVE, AIFF}
CATEGORIES	Define a vCalendar category
CLASS	Classification of access rights
TYPE	One of {PUBLIC, PRIVATE, CONFIDENTIAL}
DCREATED	Creation date of a given calendar entry
COMPLETED	The time a scheduled `to do' was actually completed
DESCRIPTION	Long comments
DALARM	Pop up alarm
Run time	The time to execute the alarm
Snooze time	Time the alarm is dormant before repeat alarm
Repeat count	Number of times the alarm is to be repeated
Display string	The text to be displayed when the alarm is executed
DUE	The date/time a `to do' is due to be completed
DTEND	The date/time an event will end
EXDATE	Specific date/time exceptions for a recurring action
EXRULE	General recurrence rule for exceptions
LAST-MODIFIED	The date/time a calendar entry was last modified
LOCATION	The location of a calendar entry e.g. a meeting room
MALARM	email alarm
Run time	The time to execute the alarm
Snooze time	Time the alarm is dormant before repeat alarm
Repeat count	Number of times the alarm is to be repeated
Email address	Destination of the alarm
Note	The text to be sent when the alarm is executed
RNUM	The number of times a calendar entry will reoccur
PRIORITY	Numeric priority for a calendar entry
PALARM	Invoke procedure on alarm
Run time	The time to execute the alarm
Snooze time	Time the alarm is dormant before repeat alarm
Repeat count	Number of times the alarm is to be repeated
Procedure name	destination of the alarm
RELATED-TO	Reference to another calendar entry
RDATE	Specifically recurring date/times for a calendar entry
RRULE	General recurrence rule for a calendar entry
RESOURCES	Resources required in a calendar entry
SEQUENCE	The instance of a calendar entry in a sequence of revisions
DTSTART	The date/time an event will start
SUMMARY	Short comment
TRANSP	Transparent to free time searches
URL	Uniform Resource Locator
UID	Globally unique identifier

Appendix 3: Example XML profile based on vCard properties

```
<?XML version= “1.0”>
<!DOCTYPE profile SYSTEM “Profile.dtd”>

<!-- Profile data based on vCard 2.1 properties -->
<Profile>
  <!-- Identification -->
  <!-- Formatted Name: for display -->
  <FN>Dr. Steve Battle</FN>
  <!-- Name: structured representation -->
  <N>
    <Family>Battle</Family>
    <First>Steven</First>
    <Middle>Andrew</Middle>
    < Prefix>Dr.</ Prefix>
  </N>
  <!-- Birthdate --><BDAY>1963-06-06</BDAY>

  <!-- Delivery Addressing -->
  <ADR TYPE ="WORK">
    <!-- note optional way to specify empty tags -->
    <POAddr/>
    <ExtAddr>UWE</ExtAddr>
    <Street>Coldharbour Lane, Frenchay</Street>
    <Locality>Bristol</Locality>
    <Region>South Gloucestershire</Region>
    <PostalCode>BS16 1QY</PostalCode>
    <CountryName>United Kingdom</CountryName>
  </ADR>

  <!-- Telecommunications Addressing -->
  <TEL TYPE ="WORK">+44-117-965-6261x3177</TEL>
  <TEL TYPE ="FAX">+44-117-975-0416</TEL>
  <EMAIL>sab@ics.uwe.ac.uk</EMAIL>

  <!-- Geographical -->
  <!-- Time Zone in Universal Coordinated Time (GMT) -->
  <TZ>0000</TZ>
  <!-- Geographic Position (degrees.minutes N/S,E/W as -/+) -->
  <GEO>51.47,-2.58</GEO>

  <!-- Organisational -->
  <TITLE>Researcher</TITLE>
 <LOGO TYPE=“GIF” HREF=“http://zen.btc.uwe.ac.uk/icons/hilogo2.gif”/>
  <ORG>
    < Name>The University of the West of England</ Name>
    < Unit>The Intelligent Computer Systems Centre</ Unit>
  </ORG>

  <!-- Explanatory -->
  <REV>1997-11-28T12:00:00Z</REV> <!-- Last revision -->
  <URL>http://www.ics.uwe.ac.uk/staff/Steve.html</URL>

</Profile>
```

Appendix 4: Document Type Declaration (DTD)

The file **Profile.dtd** is defined as:

```
<!ELEMENT Profile
  (FN, N, PHOTO?, BDAY?, SOUND?, URL?, UID?, ADR?, LABEL?, TEL?, EMAIL?, MAILER?, TZ?, GEO?,
    TITLE?, ROLE?, LOGO?, AGENT?, ORG?, NOTE?, REV?, VERSION?, KEY?)>

<!ELEMENT FN (#PCDATA) >
<!ELEMENT N (Family, First?, Middle?, Prefix, Suffix?)>
<!ELEMENT PHOTO (#PCDATA) >
<!ATTLIST PHOTO
  TYPE (GIF|CGM|WMF|BMP|MET|PMB|DIB|PICT|TIFF|PS|PDF|JPEG|MPEG|MPEG2|AVI|QTIME) #IMPLIED
  HREF CDATA #IMPLIED>
<!ELEMENT BDAY (#PCDATA) >
<!ELEMENT SOUND (#PCDATA) >
<!ATTLIST SOUND Type (WAVE|PCM|AIFF) #IMPLIED>
<!ELEMENT URL (#PCDATA) >
<!ELEMENT UID (#PCDATA) >
<!ELEMENT ADR (Street, Locality, Region?, PostalCode?, CountryName?)>
<!ATTLIST ADR Type CDATA #IMPLIED>
<!ELEMENT LABEL (#PCDATA) >
<!ATTLIST LABEL Type CDATA #IMPLIED>
<!ELEMENT TEL (#PCDATA) >
<!ATTLIST TEL Type CDATA #IMPLIED>
<!ELEMENT EMAIL TYPE
  (AOL|AppleLink|ATTMail|CIS|eWorld|INTERNET|IBMMail|MCIMail|POWERSHARE|PRODIGY|TLX|X400)
  #IMPLIED>
<!ELEMENT MAILER (#PCDATA) >
<!ELEMENT TZ (#PCDATA) >
<!ELEMENT GEO (#PCDATA) >
<!ELEMENT TITLE (#PCDATA) >
<!ELEMENT ROLE (#PCDATA) >
<!ELEMENT LOGO (#PCDATA) >
<!ATTLIST LOGO
  TYPE (GIF|CGM|WMF|BMP|MET|PMB|DIB|PICT|TIFF|PS|PDF|JPEG|MPEG|MPEG2|AVI|QTIME) #IMPLIED>
<!ELEMENT AGENT (Profile)>
<!ELEMENT ORG (Name, Unit?)>
<!ELEMENT NOTE (#PCDATA) >
<!ELEMENT REV (#PCDATA) >
<!ELEMENT VERSION (#PCDATA) >
<!ELEMENT KEY (#PCDATA) >
<!ATTLIST TYPE (X509|PGP) #IMPLIED>

<!ELEMENT Family (#PCDATA) >
<!ELEMENT First (#PCDATA) >
<!ELEMENT Middle (#PCDATA) >
<!ELEMENT Prefix (#PCDATA) >
<!ELEMENT Suffix (#PCDATA) >
<!ELEMENT Street (#PCDATA) >
<!ELEMENT Locality (#PCDATA) >
<!ELEMENT Region (#PCDATA) >
<!ELEMENT PostalCode (#PCDATA) >
<!ELEMENT CountryName (#PCDATA) >
<!ELEMENT Name (#PCDATA) >
<!ELEMENT Unit (#PCDATA) >
```