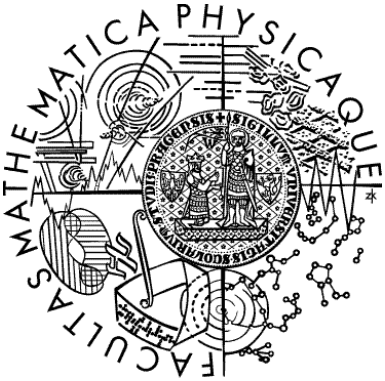
Charles University in Prague
Faculty of Mathematics and Physics

Doctoral Thesis



Jaroslav Gergič

Addressing On-Demand Assembly and Adaptation Using a Runtime Intentional Versioning Engine

Department of Software Engineering
Advisor: Doc. Ing. Petr Tůma, Dr.

Annotations

Title

Addressing On-Demand Assembly and Adaptation
Using a Runtime Intentional Versioning Engine

Author

Mgr. Jaroslav Gergič
e-mail: *gergic@dsrg.mff.cuni.cz*, phone: +420 606 376 218

Department

Distributed Systems Research Group, *http://dsrg.mff.cuni.cz*
Department of Software Engineering, Faculty of Mathematics and Physics
Charles University in Prague, Czech Republic

Advisor

Doc. Ing. Petr Tůma, Dr.
e-mail: *petr.tuma@dsrg.mff.cuni.cz*, phone: +420 221 914 267

Mailing Address

Charles University in Prague, Dept. of SW Engineering,
Malostranské nám. 25, 118 00 Prague, Czech Republic

Abstract

The World Wide Web has been changing rapidly in the past few years due to the emergence and fast adoption of large variety of new internet-enabled devices: starting with web-enabled phones through converged appliances, combining a PDA and a cell phone, to specialized internet tablets and business productivity tools. This change is bringing many challenges into the process of designing and developing both the thin-client (web-based) and thick-client (device-hosted) applications and related services. The application and service providers are facing a trade-off between the number of platforms and devices they are able to support, representing the size of the potential market, and mounting costs tied to developing and supporting multiple variants of their applications. There are several ongoing efforts taking place at various standardization organizations and industry associations to address these issues. Some of the essential standards for specifying and transporting device capabilities have been available for several years now, but so far they have had only a limited impact on the way the actual applications and services are being designed and developed. This work is trying to identify and explain the shortcomings of the existing approaches and as a reaction proposes an application-centric framework designed specifically to better manage the trade-off between the coverage and the cost. The main idea is describing device capabilities (requirements) and application artifacts (provisions) using semantically rich properties – mostly hierarchical classifications – and employing that semantical information for implementing a best-effort (approximate) matching algorithm.

Keywords

multi-variant intentional versioning, multi-criterial constraining, variation points, class factories, runtime composition and assembly, web applications, delivery context, provisioning, single authoring of multi-modal applications, Composite Capability/Preference Profiles (CC/PP), User Agent Profile (UAProf)

Revision 1160 (September 11, 2007)

Acknowledgments

This work builds on years of research and practical experience gained while working on both research and commercial projects. The component versioning baseline stemming from my work in the Distributed Systems Research Group at Charles University, Prague was subsequently influenced and enriched by the research I was involved in during the six years spent in the Human Language Technologies department of IBM T. J. Watson Research Center in Prague and Yorktown Heights, NY. Years of practical experience from various commercial projects taking place before and after my research period helped me to attain the right perspective when considering usability and real-world applicability of the information technology.

Before all I would like to thank to my advisor Petr Tůma for his relentless support and advice throughout the entire process of research, design and authoring the thesis. I would like to address special acknowledgment to Prof. František Plášil, who invited me as a member to the Distributed Systems Research Group and allowed me to learn the highest standards of academic research activity.

I would like to thank to all my former colleagues in IBM Research for wonderful teamwork experience, inspiration and many ideas which enriched my work by helping me to fully comprehend the problem domain I was trying to address. In particular, many thanks to Jan Kleindienst, Ladislav Seredi, Tomas Macek and Vladimir Bergl.

I must also thank to all my former and current colleagues I have met during my work at LCS International, Reuters and Ariba. Every project I have been working on so far has been a vastly enriching experience which has helped to form the way I think and approach problems.

Last, but definitely not least, I am deeply grateful to my parents and family for their support and encouragement. Especially during the last year when I have been finishing my work has had a disruptive impact on the family life and I am obliged to pay back to my wife Jolana and daughter Anna in the months and years to come for their patience and support.

Table of Contents

1 Introduction

The World Wide Web has been changing rapidly in the past few years due to the emergence and fast adoption of large variety of new internet-enabled devices: while the *.com*[1] era of the Internet was clearly dominated by the personal computers and the *browser wars*[2], the early years of the 21st century were signified by the emergence of the mobile Internet which allows people to connect to the Internet using a wide range of strikingly different devices. Nowadays, the users are accessing the same information and services while in the office, at home or on the go and are expecting the service providers to provide multiple variants of their applications specifically tailored to different modes of operation.

The application and service providers are facing a trade-off between the number of platforms and devices they are able to support, representing the size of the potential market, and mounting costs tied to developing and supporting multiple variants of their applications. There are several ongoing efforts taking place at various standardization organizations and industry associations to address these issues. Some of the essential standards for specifying and transporting device capabilities have been available for several years now, but so far they have had only a limited impact on the way the actual applications and services are being designed and developed.

This work is trying to identify the reasons, why the existing approaches have not achieved a broader adoption and proposes an application-centric framework, aimed specifically to better manage the trade-off between the coverage and the cost.

1.1 Application Domain

While most of the ideas discussed in the following chapters can be easily generalized to cover a broader spectrum of applications, we keep the core chapters of this thesis focused on a single application domain: the domain of *multimodal web applications*. We believe that such an approach improves comprehensibility and by using a series of related examples from the same application domain, the reader can more easily assume end-to-end real life scenarios and gain better understanding on how the proposed technologies can be applied in practice.

Multimodal web applications can be seen as a natural extension of ordinary web applications. From the end user perspective, the applications are consumed using a browser application accessing a particular web site; the only exception are voice applications, where the end user is interacting via voice over the phone and the voice browser is a part of the technical infrastructure hosted by the application provider [VXML03]. From the technological perspective, multimodal applications share the high-level architecture with regular web applications: a web browser on the client side, a web server as a front-end to the server-side infrastructure, an application server to implement the application logic and usually a relational database server for persistent data storage.

1 The second half of 90's of the 20th century (approximately 1995 – 2000) when the Internet emerged from the academia and became a mainstream, commercialized communication medium.

2 The head to head competition between Netscape and Microsoft for the dominance of the web browser market.

What makes the multimodal applications different, is that there are multiple variants of the application, allowing for different *modes* of interaction, depending on the client device. For example a cultural/sports events program guide such as one described in [ICSM2001] accessible using a web browser on a personal computer (PC), a web-enabled cell phone and a phone using voice interaction (usually referred to as Interactive Voice Response - IVR). Another example can be a *unified messaging/productivity* application allowing to access e-mail, voice mail, calendar, contacts and virtual fax inbox using a PC, smart phone or a PDA and via phone using an automated *voice assistant*. In both cases, each of the incarnations of the application is accessing the same data, implementing the same business logic and providing similar end-user functions (pending the limitations of a particular *modality*), while the requirements for the user interface design are strikingly different. Not only there are technological distinctions in terms of markup languages consumed by each platform: xHTML in case of PC versus xHTML Basic or WML in case of a web phone and VoiceXML in case of an IVR; but there are also many other aspects like screen size and resolution (or a presence of the screen at all), input capabilities: qwerty keyboard versus a numeric phone key-pad or a presence of a pointing device. All these distinctions require the application designer to consider alternative approaches to mapping the functional requirements to the user interface artifacts and often lead to implementing several different variants of the user-interface layer, or even multiple variants of the entire application – the only common denominator being the database layer.

Such a multiplication of efforts is increasing the overall cost of providing multimodal services and the lack of suitable methodology and tools may prohibit the service providers from expanding the coverage beyond the mainstream devices or even from entering the multimodal[3] services market.

1.2 Versioning Domain

While author's former work ([JG99]) in the domain of software configuration management has been trying to cover all the aspects of software component versioning including version identification, revision and variant support, more recent effort started to emphasize the important of using semantically rich properties for software versioning [JG03]. This thesis evolves the idea further and while focusing primarily on supporting variants and variation points in the application architecture and design. Following the approach and using the framework described in the following chapters, an application effectively becomes a template with pre-defined variation points and placeholders. The template serves as a skeleton for the actual application which gets instantiated by substituting the most appropriate components and resources in particular placeholders.

Another important aspect is that our former work was mostly concerned about application design-time and build-time [JG99], more recently also deployment-time assembly process [JG03]. This work is aiming solely at employing a versioning engine at the application runtime, or better to say, to address the needs of those applications which defer portions of their final assembly process until the application runtime. The multimodal applications described in the section above are a prime example: they need

3 The multimodal applications as defined above are sometimes referred to as *multi-channel* applications to emphasize the fact that each modality is used exclusively in a given point in time. This is to contrast these applications to *simultaneous* multimodal applications which combine two or more different modalities during a single user interaction – one of those modalities typically being the voice modality.

to adapt to different client devices and user preferences (for example locale settings) and it is hardly possible to deploy all the possible combinations of an application. Moreover, there is the timig aspect adding to the complexity as well: It is only upon arrival of the first HTTP request, which usually triggers the user session creation, that the application can be tailored specifically for that particular user session, prior to the session creation, it is impossible to determine the device capabilities and the user preferences. In addition to that, certain properties can change even during the user session: the user can choose a different user locale (language), change the screen orientation (portrait/landscape) and/or switch the device audio on/off. All these changes can apply at any time during the user session, which represents yet additional challenge which needs to be addressed by the runtime versioning engine responsible for (re)configuring the application as necessary, by choosing the most appropriate components and resources.

1.3 Usage Domain

We are aiming at the versioning domain from the application development perspective. The focus point is how to facilitate the design and development of an end-to-end application featuring a large number of variation points using the *single authoring* approach [SA02], while stressing the ability to delay the final assembly until the application runtime. This represents a requirement to provide a version-aware library of software components and other artifacts together with the apparatus which takes the actual runtime context of the application into account and retrieves the most suitable variant of each artifact and substitutes it at the corresponding variation point. We are trying to address the process of multi-variant application design applying the recursive top-down approach: introducing a skeleton together with the core functionality tied to the skeleton and a set of placeholders ready to pull-in the plug-in components and thus instantiate many different variants of the application. This can be contrasted to the bottom-up approach of a more traditional software component versioning perspective, where we would deal with versioning of individual components and then the possibilities to combine them together in a compatible way so that we end-up with a consistent application.

1.4 Thesis Contributions

This thesis makes the following contributions to the research within the domain of Computer Science:

- analysis and critical reflection of the related work and standards in the area of multimodal web applications and web content adaptation
- definition of a generic object-oriented framework targeting the application runtime assembly and adaptation tailored the application domain stated above, while ensuring the following properties of the framework:
  - flexibility in terms of metadata sources used for assembly and adaptation
  - comprehensibility and steep learning curve for the application designers and developers
  - enforcing best practices in terms of separation-of-concerns, modularity and reusability
  - scalability from both the logical complexity as well as runtime performance perspectives
- the framework has been designed with immediate practical usability mindset, based on the author's comprehensive experience in the domain of interest

1.5 Structure of the Thesis

In the three subsections above, we stated the context this work by defining the *class of applications* we are targeting (1.1 Application Domain), the *versioning aspects* this work is addressing (1.2 Versioning Domain) and the *intended usage mode* we are trying to support (1.3 Usage Domain).

In the following chapter ( ) we prepare the stage for a deeper discussion by going over a set of case studies and motivational examples (2.1 Case Studies), followed by the relevant standards established in domain of interest (2.2 Related Standards). Then, based on the evaluation of the existing standards and technologies, we summarize our observations and map the landscape of the problem area in section 2.4 (Important Observations). In section 2.3 (Related Work) we look at how the shortcomings of the standards-based technology stack are being addressed and how the gap remaining between the standards stack and the application developer is being overcome. The evaluation of alternative initiatives leads us to setting the goals of our own work (3 Setting the Goals). The chapter is concluding by discussion of the possible approaches to fulfilling our goals (4 Addressing the Goals).

The chapter 5 (The Versatile Framework) presents the key deliverables of this work. We start by describing the underlying concepts of the framework in section 5.2 (Conceptual Overview). The abstract part is followed by the technical part describing the mapping of the framework's abstract concepts to a concrete representations in the object-oriented programming language (5.3 Technical Overview). All the major elements of the technical representation are described in subsequent sections of the chapter; however, due to readability reasons, the work itself presents only important aspects of each element and the very technical details of the framework are separated into an API reference manual *Versatile 1.0 API Reference* [VERSAPI], which is an integral part of the work.

The last chapter (6 Conclusion) evaluates the framework presented in chapter 5 against the goals set in section 3 and compares it to the alternative approaches presented in chapter 2.3. We also briefly re-iterate the important aspects of the framework and hind possible application beyond the scope defined the first chapter.

2 Background

In this chapter, we go over a set of case studies and motivational examples. We also discuss in more detail the standards and technologies relevant to the focus domain of this thesis. Based on this research, we note some important observations and draw conclusions which lead us to setting the goals and requirements, thus forming the design guidelines for our own work.

2.1 Case Studies

2.1.1 Content Adaptation Legacy and Reality

Even though the Internet and the World Wide Web has always been a prime example of applying open standards in practice, still there has been a need to adapt content according to the capabilities of a particular end-user terminal – the *web browser*. There have always been differences between the features implemented by different web browser vendors and across the hardware and operating system platforms. The HTTP protocol, the engine of the World Wide Web, allows the client devices to issue meta data attached to every HTTP request in the form of HTTP headers. Some headers are standardized by the HTTP protocol, however, vendor-specific extensions are allowed as well.

The Examples 1 and 2 show the HTTP headers issued by Mozilla Firefox 2.0 and Microsoft Internet Explorer 6.0 web browsers respectively. Not all HTTP headers are used for content adaptation, some of them like *Keep-Alive*, or *Cache-Control* are used for other purposes. Let us have a closer look a the HTTP headers which are used for content adaptation most often:

- Accept – a list of MIME media types[4] consumable by the web browser
- Accept-Charset – a list of character encodings consumable by the web browser
- Accept-Language – a list of preferred user locales[5] (this is rather a user's preference than browser capability)
- User-Agent – information about the web browser, it usually contains a name and version, it is quite common to include operating system and relevant runtime libraries information

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
     rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
Accept: text/xml, application/xml, application/xhtml+xml,
    text/html;q=0.9, text/plain;q=0.8, image/png, */*;q=0.5
Accept-Language: cs,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cache-Control: max-age=0
```

Example 1: Mozilla Firefox 2.0 HTTP Headers Example

4 MIME (Multipurpose Internet Mail Extensions) media types are managed by IANA (Internet Assigned Numbers Authority) [MIMEMT]

5 *Locale* – typically a *language* and optionally also *country* ISO codes, see also [LC142]

Out of the list above, the *Accept* header was originally meant to be the primary factor to be taken into account for the content adaptation, however, as you can see from the real world samples, the actual web browsers use this field to list some sort of examples terminated by the magic "*/*" content type which means: *I accept just anything*.

```
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
    .NET CLR 1.1.4322; .NET CLR 2.0.50727)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
    application/vnd.ms-excel, application/vnd.ms-powerpoint,
    application/msword, application/x-shockwave-flash, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Example 2: Microsoft Internet Explorer 6.0 HTTP Headers Example

Due to the above, another HTTP header has established itself as the primary content adaptation factor: the *User-Agent* header. According to the standard [HTTP11]: *"The field can contain multiple product tokens and comments identifying the agent and any subproducts which form a significant part of the user agent "* Product tokens take the form of `name/version` and should be listed in order of their significance. Example 3 below Shows an example taken from the HTTP specification. Looking at the real world browsers (Example 1 and Example 2), it is clear that the standard is not being followed very strictly and the *User-Agent* header is beging formatted in many different ways, which makes parsing and interpreting the header values quite difficult.

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

Example 3: User-Agent header according to the HTTP 1.1 standard

In addition to the non-standard format, there are other serious drawbacks linked to the fact that the *User-Agent* header has become the major (and many times the only) factor used for content adaptation: the content providers need to know all its clients in advance in order to build the feature matrix (web-browser -> feature set mapping). They also need to keep updating this matrix as new devices appear on the market and and the web browsers evolve. This was not so much an issue during the 90's as there were only two major browsers on the market (Netscape and Microsoft Internet Explorer), however, with the growing number of browsers (Opera, Safari) and especially, with all the variety of web-enabled phones and other mobile devices, this simple mechanism becomes a serious bottle-neck and in fact creates a barrier-of-entry for all the newcomers to the browser market as the web-servers are unable to serve content properly to *unknown* user agents.

In fact, the issues with this approach dates back to the late 90's: when Netscape introduced *frames*[6], the content providers started using the *User-Agent* header to determine whether a framed or non-framed version can be served to the client. The keyword they were looking for was *"Mozilla"*. Due to this fact, even today most of the web browsers on the market claim they are Mozilla, even though they are not, for compatibility reasons. (this technique is known as *cloaking*).

6 At time a unique extension to HTML markup language allowing to split a web page into several independent regions

Usage of the *User-Agent* header and other HTTP headers mentioned above still remains prevalent form of content adaptation on the World Wide Web today. There are several ongoing efforts to improve the situation, yet none of them has got sufficient traction in the community so far. We briefly introduce the most promising standards-based attempt to address the content adaptation needs in section 2.2.1 of this chapter.

2.1.2 CATCH 2004

The Converse in Athens, Cologne and Helsinki (CATCH) 2004 was a research project[7] whose objective was to *develop a multilingual, multimodal, conversational system with a novel unifying architecture across devices and services* [CATCH2004]. The author of this thesis was actively involved in the project regarding the design and development of its server-side infrastructure (publications [ICSM2001], [IIWAS2001] and [SEKE02]; see also [ICMI02] and [IWANLIS01]); some design aspects of the CATCH 2004 multimodal framework have inspired the versioning engine described in this thesis.

In CATCH 2004, we were designing and developing a multimodal framework validated by a series of proof-of-concept applications used to demonstrate its capabilities. Its earlier phase included a cultural events information service for cities of Athens, Helsinki and Cologne, the later phase was mostly focused on to the Olympic games information service using real data gathered in the course of preparations for the 2004 summer Olympics in Athens. The modalities supported by the framework and subsequently by the pilot applications were three single-modal channels: PC-based web browser (HTML), web-enabled phone (WML[8]) and voice interaction over phone using *natural language understanding* (NLU) capabilities. There were also two dual-mode channels involving combinations of HTML or WML with VoiceXML[9] to allow for simultaneous interaction using both visual and voice web browsers.

Given the requirement to support *multilingual* as well as *multimodal* variants of each application we needed to organize components and resources alongside these two orthogonal axes. As a research project, we did not need to deal with the real-world variety of the client devices, we only needed one representant of its class for each modality, but still there was a need to identify the devices and map them on the modality axis in a generic and an extensible way. To actually implement the modality dimension, we took an inspiration in the other axis: the language/locale axis and its implementation in the Java programming language.

Java uses a triplet[10] (*language*, *country*, *variant*) to represent information about user's locale. Not all attributes of the triplet are mandatory, it is for example quite common to specify only the user's language, but if needed, more details can be provided to properly render user interface according to the user's regional settings. In addition to the standards-based locale descriptor, there is a clever best-effort algorithm implemented as a part of the Java Standard Library to locate the most appropriate resources given

7 Work partially funded by the European Union, through the research project IST-1999-11103 CATCH2004 as a part of the IST (Information Society Technologies) research program.

8 [WAP20] - an Open Mobile Alliance (OMA) standard for web-enabling cell-phones and other mobile terminals

9 [VXML03] - a W3C standard markup language for voice interaction

10 [LC142] - java.util.Local serves as a locale descriptor (language, country, variant), languages and countries must adhere to their respective ISO language and country codes, variants are implementation specific.

a particular locale information[11]. Given the *resource name*, the user's locale and the system default locale information, the algorithm searches in the following sequence:

1. *resource name* + *language* + *country* + *variant*
2. *resource name* + *language* + *country*
3. *resource name* + *language*
4. *resource name* + *default language* + *default country* + *default variant*
5. *resource name* + *default language* + *default country*
6. *resource name* + *default language*
7. *resource name*

The above fall-back strategy combines relaxing a given set of constraints (going from very specific to more generic locale) with almost assured availability of the resources for system default locale. This allows for sharing locale-dependent resources where appropriate (for example on the language level) while providing more specific data where needed (e.g. country-dependent date or currency formatting). The algorithm also allows to implement fail-over in case the required resources are missing for a particular locale. In fact, the algorithm defines a classification hierarchy (*taxonomy*) of the Java locale triplets. When relaxing a constraint (assuming a fully specified triplet given as the constraint) it bubbles up the taxonomy from a bottom leaf node up to its root, when searching for a partially-specified locale, it may start from one of the inner-nodes of the taxonomy. Figure 1 below demonstrates the hierarchical classification idea drawing a taxonomy tree using a small subset of potential Java locale triplets.

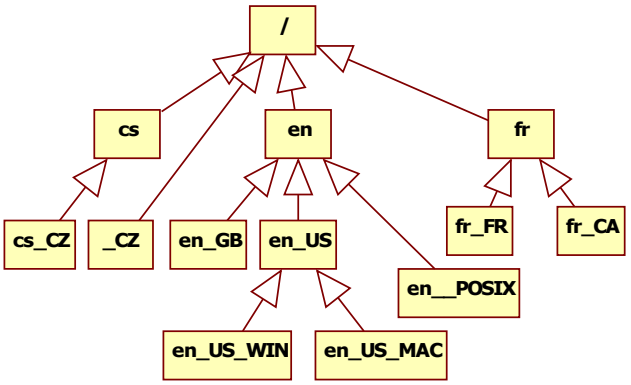


Figure 1: A snippet of the Java locale taxonomy

As mentioned earlier, we took the way Java treats the locale-specific resources as a template to implement the second axis: the modality-specific resources. We defined the *modality* triplet:

1. *modality family* (`GUI` versus `SPEECH`)
2. *modality name* (a markup name like `HTML`, `WML` or `VXML`)
3. *modality flavor* (version of a particular markup)

Examples of modalities in their textual form are: `GUI/HTML/4.0` or `SPEECH/VXML/2.0`. Figure 2 on page 17 shows the modality taxonomy as employed throughout the CATCH 2004 project. We were using HTTP header attributes (*User-Agent* and *Accept,* described in section 2.1.1 above) to derive modality triplet from incoming HTTP requests. We implemented an algorithm, similar to the one described above for the Java locales, to search for the most appropriate resources and components, while being able to share as

11 [RB142] - java.util.ResourceBundle implements the lookup algorithm for localized resources.

much as possible between the different variants of the application. When configuring class factories, we tagged the implementation classes by modality information and the modality-aware class factory then picked-up the nearest best-match to the modality information provided in the incoming request.

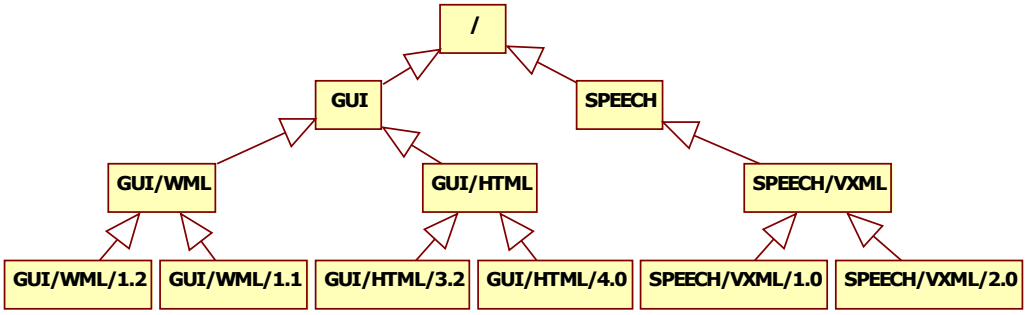


Figure 2: The modality taxonomy as used in CATCH 2004

When storing and retrieving locale-**and**-modality sensitive resources, we had to have them organized using both dimensions. We chose the locale dimension to be the primary one (in order to avoid mixing different languages on the same "*screen"*) and the modality dimension as the secondary. Static resources were stored in locale-specific XML files and each resource was tagged by the most-generic modality it can serve. It allowed us to share a large number of resources between all the modalities while accommodating the user interface to a particular modality where needed. Similar approach was applied to the resources dynamically pulled out of the database[12].

```
<message name="eventType">
  <!-- the default content (untagged) -->
  Select the event type:

  <!-- following sub-entry is content for all the GUI/WML/* -->
  <alt modality="GUI/WML">Event Type:</alt>

  <!-- any speech modality uses the following sub-entry -->
  <alt modality="SPEECH">Please say the event type.</alt>
</message>
```

Example 4: An example of modality-tagged resources

The Example 4 shows a snippet of the static resource file. Please note the differences between the modalities: a PC web browser (assuming `GUI/HTML/*` modality) falls back to the default variant, a web phone retrieves a special shortened label to better fit a small screen, while the speech modality requires a whole sentence to prompt the user for the data entry.

The CATCH 2004 multimodal framework featuring just two attributes (properties) in its *delivery context* , both being shallow/fixed-depth taxonomies, seems to be very simple; yet it turns out to be very powerful and efficient in applying the *single-authoring*[13] approach to developing a family of variants of the web application – a multimodal

12 The data model had to be extended to accommodate both the locale and modality information and the cultural/sport events data providers had to extend their listings accordingly.

13 Developing all the application variants together within one framework as opposed to developing each variant separately as a separate project, possibly using different tools and having different teams of people involved.

application. It is a well-known fact, that to take a successful research prototype to the streets means over eighty percent of work still needs to be done. While keeping the ideas introduced in this section in mind, let us move on to see how these can be generalized to reach the point of practical applicability in the real-world environment.

2.2 Related Standards

2.2.1 CC/PP and UAProf

In this section, we briefly introduce the existing standards-based technology stack, built partly in a response to the issues with using the *User-Agent* and *Accept* HTTP headers for the purpose of content and application adaptation on the World Wide Web, as discussed in section 2.1.1. The majority of the issues related to using the *User-Agent* header stems from the fact that the content adaptation is *implicit*: given a device identifier, the server implicitly assumes the device properties. The only piece of information which needs to be attached to a request is the device identifier, on the other hand, the recipient of the request must maintain the implicit knowledge. The idea underlining the standards-based technology stack is to describe the device capabilities explicitly, by describing the device capabilities feature by feature so that the content providers and application authors do not need to care about a particular device but rather focus on the set of capabilities of the device.

The technologies introduced by *World Wide Web Consortium* (W3C) and subsequently by *Open Mobile Alliance* (OMA) are aiming to solve the content adaptation problem by employing the very same metadata technologies which lay the foundations of the *Semantic Web* activity [SEMWEB], [SEMWEBVIS]. The heart of the semantic web is built around the *Resource Description Framework* (RDF) [RDF04], an XML-based system for annotating existing (web) resources with external metadata. RDF uses statements in the form of triplets (*subject*, *predicate*, *object*) to express facts about resources. Resources are uniquely identified using URIs[14] and correspond to the *subject* in the triplet above. Predicates are named properties, these also must have a URI to prevent ambiguity. Objects are property values, these can be other resources (to express relationships between resources) or literal values (strings, numbers and other XML primitive data types). RDF also allows properties to have multiple values, this is achieved using two special RDF types: *Bag* (an unordered set of values) and *Seq* (an ordered list of values).

W3C has developed *Composite Capabilities/Preference Profiles* (CC/PP) framework [CCPP04], which builds on RDF and introduces concept of device *profiles* composed of several *components* each component being a logical grouping of related *attributes*. CC/PP framework also introduces the *defaults* concept – the ability to partially override a subset of component attributes by specifying their values while referring to the existing "*default*" attribute set via URI for retrieval of non-overridden values. The *defaults* concept is important with the respect to network utilization and caching: the aim is to prevent the entire device profile being transferred over and over with every HTTP request.

14 URI – Uniform Resource Identifier (please refer to http://www.w3.org/Addressing/)

```
<!-- Start of Hardware component -->
- <prf:component>
  - <rdf:Description rdf:ID="HardwarePlatform">
      <rdf:type rdf:resource="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-20021212#HardwarePlatform"/>
    - <prf:BluetoothProfile>
      + <rdf:Bag></rdf:Bag>
      </prf:BluetoothProfile>
      <prf:BitsPerPixel>18</prf:BitsPerPixel>
      <prf:ColorCapable>Yes</prf:ColorCapable>
      <prf:CPU>ARM</prf:CPU>
      <prf:ImageCapable>Yes</prf:ImageCapable>
    - <prf:InputCharSet>
      - <rdf:Bag>
          <rdf:li>ISO-8859-1</rdf:li>
          <rdf:li>ISO-10646-UCS-2</rdf:li>
          <rdf:li>US-ASCII</rdf:li>
          <rdf:li>UTF-8</rdf:li>
        </rdf:Bag>
      </prf:InputCharSet>
      <prf:Keyboard>PhoneKeyPad</prf:Keyboard>
      <prf:Model>N95-3</prf:Model>
      <prf:NumberOfSoftKeys>2</prf:NumberOfSoftKeys>
    - <prf:OutputCharSet>
      - <rdf:Bag>
          <rdf:li>ISO-8859-1</rdf:li>
          <rdf:li>ISO-10646-UCS-2</rdf:li>
          <rdf:li>US-ASCII</rdf:li>
          <rdf:li>UTF-8</rdf:li>
        </rdf:Bag>
      </prf:OutputCharSet>
      <prf:PixelAspectRatio>1x1</prf:PixelAspectRatio>
      <prf:PointingResolution>Pixel</prf:PointingResolution>
      <prf:ScreenSize>240x320</prf:ScreenSize>
      <prf:ScreenSizeChar>15x6</prf:ScreenSizeChar>
      <prf:StandardFontProportional>Yes</prf:StandardFontProportional>
      <prf:SoundOutputCapable>Yes</prf:SoundOutputCapable>
      <prf:TextInputCapable>Yes</prf:TextInputCapable>
      <prf:Vendor>Nokia</prf:Vendor>
      <prf:VoiceInputCapable>Yes</prf:VoiceInputCapable>
    </rdf:Description>
    <!-- End of Hardware component -->
  </prf:component>
```

Example 5: Hardware Platform component of Nokia N95 phone UAProf profile

While CC/PP defines the framework for working with the device and preference profiles, it does not define any particular vocabulary (a set of components and their attributes) nor does it define a particular protocol for transporting a CC/PP profiles and profile references. This is where the OMA[15] comes in with its *User Agent Profile* (UAProf) specification. [UAP06] UAProf builds on CC/PP and provides a standardized vocabulary of components and their attributes expressed in terms of an RDF Schema. Please refer to on page for an extract of the *HardwarePlatform* component from the Nokia N95-3 smart phone[16] UAProf profile. Other standard UAProf components include *SoftwarePlatform*, *NetworkCharacteristics*, *BrowserUA*, *WapCharacteristics*, *PushCharacteristics* and *MmsCharacteristics*. They are expressed in a similar way as the *HardwarePlatform* in . A UAProf profile can also include vendor-specific extension components in addition to the standard components defined by the UAProf RDF Schema governed by Open Mobile Alliance.

15 OMA – Open Mobile Alliance – a consortium of mostly mobile operators, device and network suppliers

16 For a complete UAProf profile cited in this work please refer to http://nds.nokia.com/uaprof/NN95-3r100.xml

UAProf standard defines two protocols for exchanging the profile information: one based on WSP[17] and more importantly the new WAP 2.0-compliant protocol called *Wireless Profiled HTTP* (W-HTTP). W-HTTP is a CC/PP-aware extension of HTTP 1.1 protocol. The protocol defines several extension HTTP headers (*x-wap-profile*, *x-wap-profile-diff*, *x-wap-profile-warning*) and their semantics required for transporting CC/PP profiles and their fragments over HTTP and also the resolution rules needed to merge the profile and optional profile fragments (overrides) into a single *consolidated profile* on the server side.

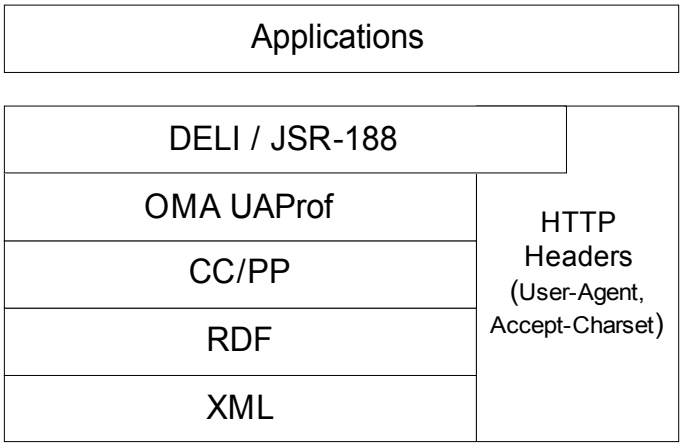


Figure 3: Device Capabilities / User Preferences Technology Stack

2.2.2 DELI and *CC/PP Processing Specification*

While CC/PP together with UAProf provides sufficient and complete resources for device manufacturers and network infrastructure providers, it comes short when targeting the application providers and developers. As described above, the standard defines the data structures, protocols and semantics (resolution rules), but does not provide any application programmer's interface (API). This gap has been filled by *DELI – Delivery Context Library* [DELI], an open source library developed originally by HP Labs[18], and later on by a Java extension API developed as a part of the *Java Community Process* under *Java Specification Request* (JSR) #188 called *Composite Capability/Preference Profiles (CC/PP) Processing Specification* [JSR188]. It is interesting to note that some people were involved in both activities, for example Mark H. Butler of HP Labs who was originally leading the development of DELI also co-authored JSR188, so that the later has been significantly influenced by DELI.

It is important to note, that there is a new RDF query language [SPARQL] currently under development in W3C, which, once approved as a W3C Recommendation, can potentially replace the simple *getter* APIs like DELI and JSR188. The SPARQL language allows quite advanced RDF queries (it is in some sense comparable to SQL) and also uses XQuery [XQUERY] functions and other functions like *regular expressions* to express constraints and implement low-level data manipulation of individual RDF literal values.

17 WSP – Wireless Session Protocol – a part of the original WAP 1.0 protocol stack which required a special WAP gateway to mediate all communication between the WAP devices and eventual content providers. This requirement has been removed in WAP 2.0 where the WAP gateway (if present) serves as a standard HTTP proxy.

18 Hewlett-Packard's advanced research division of (http://www.hpl.hp.com/)

2.2.3 Standards Stack Evaluation

The technology stack, backed by the various standardization organizations[19] we presented throughout the above sections and summarized on Figure 3, page 20, servers as both *the base* and *the benchmark* in this thesis as it represents the *gold-standard* in the area of our interest:

- many of the alternative solutions build directly on the standards stack; or they are at-least influenced-by or partly dependent-on the standard stack.
- we use the standards stack to compare these alternative solutions and in turn also our own work to the standards stack in order to measure the *added value* of a particular framework when comparing it to the *common* and *readily available* basic solution.

First of all, we proceed with an evaluation of the standards stack itself in order to reveal why there is still a need of research and development in the area even despite the existence of the relevant standards.

Mak H. Butler, involved in the design of the two standards-based frameworks briefly introduced in section 2.2.2, subsequently published several papers ([CCPPIIFD], [SEMHYPE], [BARRIERS], [IDDWG05]) reflecting on the practical experience and feedback gathered while evangelizing DELI and its underlying semantic web technologies (UAProf, CC/PP and RDF). Some of these articles became frequently cited and even considered slightly controversial, for example *Is the Semantic Web Hype?* [SEMHYPE] or from the perspective of this thesis extremely valuable *Barriers to the real world adoption of Semantic Web technologies* [BARRIERS].

The later article is trying to summarize all the various reasons behind slow adoption of the semantic web technologies including the difficulties in practical adoption of the standards-based technology stack described in the section above. Let us *cite* the reasons and briefly summarize (and comment on) those especially relevant to this thesis:

1. *"Producers, consumers and beneficiaries"* – in case of UAProf profiles, the producers of metadata are device vendors, while the consumers (and beneficiaries) are the application providers. There is insufficient *business interest* of the device vendors to sufficiently invest in metadata and so that they fully satisfy the needs of the application providers.
2. *"Classifying information is inherently hard"* and "*Metadata is inherently biased"* – even in cases of business neutral metadata, it is often difficult to settle on a standardized vocabulary, in case the business interests of different parties are not aligned, it is almost an impossible task to achieve sufficient consistency
3. *"People are inherently fallible"* – people do make errors and "*The Complexity of RDF/XML"* – it is even easier to make a mistake, if a technology is too complex, still after several years of UAProf usage by mobile devices vendors, there are instances of device profiles around with basic syntax errors.
4. *"Supporting multiple vocabulary versions"* – with every incompatible new revision of the UAProf vocabulary, the corresponding RDF Schema XML namespace changes, strictly speaking, such a namespace change means (in terms

19 XML, RDF, CC/PP backed by *W3C*, UAProf by *OMA* and JSR 188 by the *Java Community Process* consortium

of RDF) that even if the semantics of a particular attribute remains unchanged, the attributes have a different RDF type – they are incompatible and incomparable. In theory, one can built an ontology, using constructs of the OWL[20], to explicitly unify such attributes and make them semantically equivalent. In practice, employing an ontology language for solving such a simple issue is often considered an overkill and often leads to a decision to blindly ignore the namespaces at all, which in turn effectively disables profile validation.

5. *"Supporting multiple vocabularies"* – again, similar to the above, in order to implement interoperability between different vocabularies, possibly defined by different entities, there is a need to reach beyond the UAProf / CC/PP / RDF stack and use yet an additional tool or a language to provide mappings between semantically equivalent attributes. In case of UAProf, this is namely a concern for extension profile components which are not governed by OMA as a part of the UAProf standard, or which at some point are introduced as custom extensions and later become a part of the standard.

```
<?xml version="1.0"?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:param name="deli-capabilities"/>
 <xsl:template match="/">
  <xsl:if
test="contains($deli-capabilities/browser/CcppAccept/li,'wml')">
    <xsl:call-template name="wmldevice"/>
  </xsl:if>
  <xsl:if
test="not(contains($deli-capabilities/browser/CcppAccept/li,'wml'))">
    <xsl:call-template name="htmldevice"/>
  </xsl:if>
 </xsl:template>
 ...
</xsl:stylesheet>
```

Example 6: A UAProf aware XSLT stylesheet (from DELI documentation)

The observations listed above provide an excellent insight into why the semantic web paradigm as a whole and UAProf in particular has not taken off as expected. Let us add some additional notes regarding the UAProf, or better to say, its API exposed to the application developer via DELI or JSR188. The Example 6 on page 22 shows a snippet of an XSLT template given as an example in the DELI documentation, looking at the code snippet above and considering the complexity of the technology stack behind it, one can easily come to a conclusion that the result comes a bit short of expectations from the application developer's point of view:

1. The UAProf profile is essentially represented as a hash table, a key being the concatenation of a UAProf *component name* and an *attribute name.* The question is, why all the complexity of the semantic web, with its ability to express typed relationships between universally identifiable entities, is needed in order to represent such a simple collection of facts. On the other hand, it is the matter of fact that this kind of simplicity is exactly what the developer wants, as his/her focus is the application development, not the meta-modeling.

20OWL - OWL Web Ontology Language, a W3C Recommendation [OWL04]

2. There are no additional higher-level constructs on the top of the simple hash table concept to further facilitate application development. The different variants of an application need to be implemented via traditional *if-then-else* approach which can severely pollute the application code. Neither DELI nor JSR188 provide any means for clustering devices into classes and sub-classes according to their similarity given a sub-set of relevant UAProf attributes. Moreover, given the fact that both Java APIs mentioned above return raw (non-canonicalized) attribute values, it is the responsibility of the application developer to account for all vendor-specific alternatives: while investigating the existing device profiles, we encountered that a single attribute value can have many syntactical variations depending on a device vendor. We discuss this issue in further detail in following sections..

Overall, we can claim that the standards-based technology stack as depicted on Figure 3, page 20, despite all its depth and complexity, still provides only a raw basic API which, if used directly, does not fully meet the needs of the application providers and developers: in order to separate the versioning code from the business logic, allow for similarity-based device clustering (to foster artifact re-use across variants), implement semantical mapping and synthesis (merge) of information from UAProf and non-UAProf sources: there is still need to support *"legacy"* devices as most of the desktop browsers today and even some mobile devices like PDAs do not support CC/PP and UAProf and rather stick to custom vendor specific HTTP headers, and therefore one still needs to develop an additional layer on top and aside of the existing technology stack.

2.3 Related Work

Majority of the research and development in the domain of interest took place between the years 1999 and 2003 the issue was being systematically addressed primarily in a reaction to the emergence of web-enabled cell-phones based on the *Wireless Application Protocol* [WAP20] in 1999. This era concluded by the standardization efforts discussed in section 2.2 above.

The issue was, that the hype around the mobile internet came ahead of its time: the early devices were too compromised in terms of computing power and user interface features like screen size, lack of color displays and lack of multimedia features. The speed and bandwidth of the mobile networks at the time also represented a serious bottleneck. As a result of this, the end-user experience was negatively affected and the idea of mobile internet as a whole did not take-off quite well as expected. In turn, as the interest in the content adaptation faded away, the standards technology stack did not have a chance to prove itself in practice and get incrementally refined and improved from the usability perspective by sufficient number of practical applications.

It has taken another five years until the mobile networks and mobile terminals have evolved so that the user experience has improved sufficiently and the idea of mobile internet and contend adaptation is once again becoming a hot topic. Nowadays, many application and content providers[21] are starting to provide a lightweight *mobile* version of their services in order to meet the growing demand and market potential. These mobile versions are typically developed separately from their full-fledged PC versions,

21 Let us name just a few examples here: Yahoo, Google with mobile versions of their e-mail applications and productivity applications, Reuters, BBC and International Herald Tribune as typical examples of content providers.

which is apparent from a different release cycle and the fact that their feature-set, understandably limited, sometimes suffers from inconsistencies not present in their PC counterparts.

In this section, we are trying to map the landscape of the development in the domain of interest which which has been taking place more recently, in general taking place past the standardization efforts described in section 2.2; or which reflects on the standards technolog stack by attempting to extend it further to make it more practically usable. Another summary of related work from the standardization effort perspective compiled recently by the W3C Device Independence working group can be found of W3C website [DIDCO06]. Surprisingly, besides the standards discussed in section 2.2, the document only mentions WURFL – the subject of the following subsection.

Comparison of our work and the existing standards (section 2.2) as well as related work (section 2.3) is discussed in the Conclusion chapter (6).

2.3.1 Open Source Frameworks

WURFL

WURFL (*Wireless Universal Resource File*, [WURFL]) is an alternative approach trying to address the content and application adaptation in the sub-domain of mobile and wireless devices. In fact, it builds on the principle of unique device identification using the *User-Agent* HTTP header, as described in section 2.1.1, while leveraging the richness of the information provided by the existing UAProf profiles. WURFL is an XML-based repository of known devices' profiles. Together with the repository itself, there is a library providing an API allowing to:

1. resolve a given *User-Agent* string to a repository *Device ID*
2. given a *Device ID* to query attributes (*capabilities* in WURFL) of the device

The value proposition of WURFL is, that while loading profiles into the repository, the profiles are checked for syntactical and factual errors and these are corrected. In addition to that, the WURFL profiles can contain additional attributes which are not represented in UAProf but are often needed by the application developers. The profile repository is maintained centrally in a collaborative manner so that the effort to maintain and update the database as new devices emerge on the market is not replicated by each application provider separately.

Closely linked to WURFL, there is WALL (the *Wireless Abstraction Library*) provided by the same community. WALL is an extension JSP tag library[22] allowing the application developers to write their mobile application using a unified markup language abstracting away from the incompatibilities between various devices. WALL library leverages WURFL to detect actual devices capabilities and emits a markup suitable for a particular device.

The major advantages of WURFL and WALL are clearly their simplicity and ease of use from the application developer's perspective – one does not need to care about the complexity of the semantic web and UAProf nor to install and manage an implementation of CC/PP. There is only a need to install a single Java library and keep updating (downloading) the device repository on regular basis. Using WURFL, the

22 JSP – JavaServer Pages, an integral part of the J2EE platform, http://java.sun.com/products/jsp/

users are getting the functionality almost equivalent to DELI or JSR188 at much lower initial investment and enjoying much steeper learning curve. If the application domain is limited to mobile devices and it is a web-based application (not a J2ME[23] application), one can potentially take an advantage of WALL thus effectively developing a single common code base for all variants of the application.

The disadvantages of the WURFL and WALL are their sole focus on the mobile wireless devices. The WALL library is limited to the common feature *subset* across all devices: every WALL JSP tag must be possible to translate into a markup construct of a particular device, so that it is insufficient in cases when a broader set of devices is needs to be targeted. Using solely WURFL, without applying WALL, is functionally close to employing one of the CC/PP APIs mentioned in the previous section (DELI or JSR188): except the profile data cleanup executed when loading a new profile into the WURFL repository (an added value when comparing to CC/PP), the other reservations discussed in the previous section do apply to WURFL as well.

Capability Classes

The concept of *capability classes* [CAPCLASS], [CAPPROF] reflects on the authoring difficulties when authoring applications with the use of CC/PP and UAProf. The author is trying raise the level of abstraction by avoiding the direct usage of CC/PP attributes combined with (possibly) nested if/then/else statements as depicted on Example 6 on page 22. The level of abstraction is raised by defining a set of *capability classes*, each capability class being defined using a set of constraints over the existing CC/PP attributes of the device profile. Examples of capability classes are: *smallScreen*, *largeScreen*, *jpegcapable*, *wapenabled*, *color*, *blackandwhite*, *colorlesswap*, *smallbw*. Each capability class is defined by constraining (=, <, <=, >=, contains, not, true) one or more CC/PP attributes. At runtime, the constraints are applied to the device's profile which in turn returns a set of *device capabilities*, i.e., a set of capability classes the device belongs to. Then, the application author can adapt the content based on the set of capabilities rather than reasoning over the raw CC/PP attributes.

It is interesting to note, that the capability classes do not constitute any explicit hierarchy (sub-classing). Data analysis techniques like *concept analysis* (4.2.4) would need to be applied to the capability classes definitions to discover patterns and establish the capability class hierarchy implied by the set of defining constraints.

The capability classes were actually implemented as an experimental feature of DELI library [DELI], unfortunately, the idea was not taken into account in the JSR 188 specification [JSR188].

23 J2ME – Java 2 Platform, Micro Edition, java.sun.com/javame/index.jsp

2.3.2 Commercial Frameworks

Volantis Mobile Content Framework™

Volantis Mobile Content Framework [VOLANTIS02], [VOLANTIS07] represents an example of a commercial product targeted at the same application domain as WURFL described above. In fact, despite being much more comprehensive in its feature set, it shares many of the basic ideas of WURFL:

- it also leverages a proprietary database of device profiles, rather than depend on the raw unsupervised information provided by the devices themselves
- it uses an abstract markup language[24] for application authoring (compare to WALL) and then translates this abstract markup into a concrete markup of a particular device at runtime
- its application domain also limited to mobile devices, i.e., it is not trying to provide a one-size-fits-all unifying framework for application single authoring for both PCs and mobile internet.

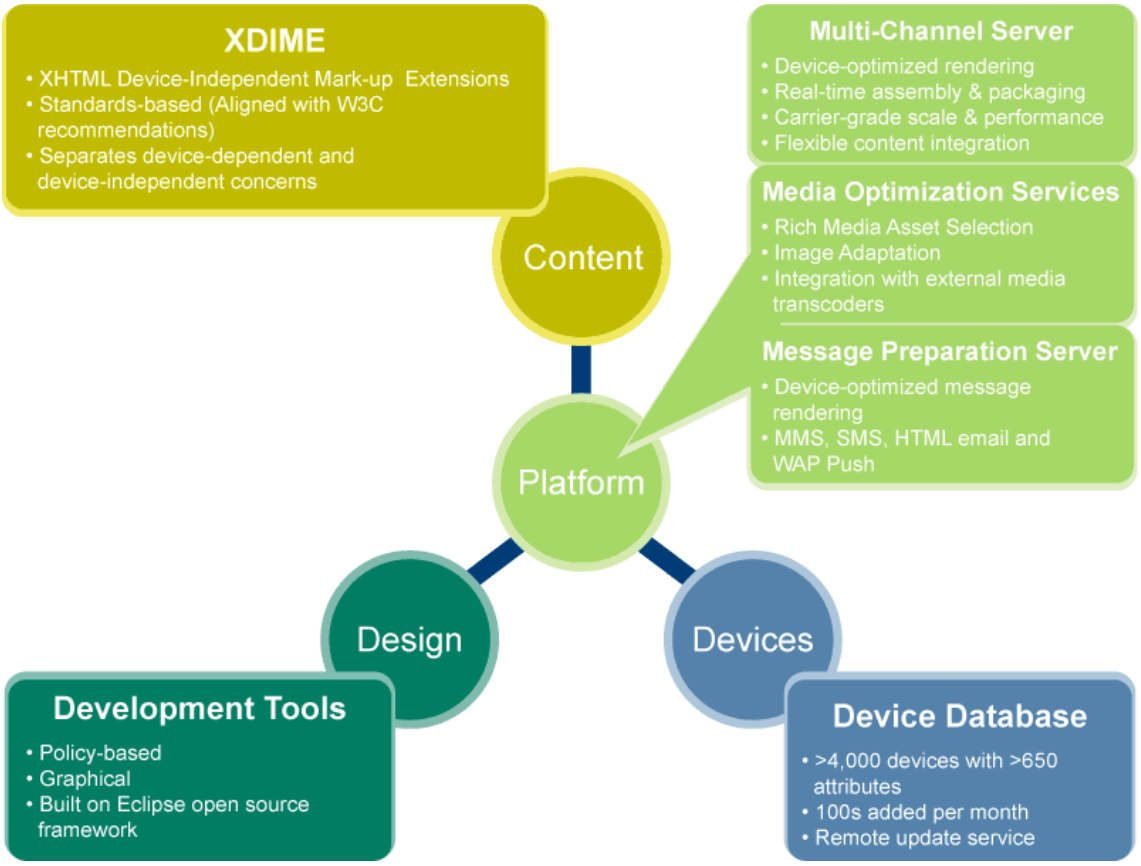


Figure 4: High-level schema of the Volantis Framework (source: volantis.com)

Remarks: Volantis Systems is a founding member of the *W3C Mobile Web Initiative* [W3CMWI], it has also participated in the *W3C Device Independence Working Group* [W3CDIWG] which has recently transformed into *W3C Ubiquitous Web Applications Working Group* [W3CUWA].

24 XDIME – XHTML Device-Independent Markup Extensions

MobileAware Interaction Server

An another commercial product which is a part of the company's mobile application suite [MAWARE]. Similarly to WURFL (2.3.1) and Volantis Framework (2.3.2), it leverages a device repository bootstrapped by CC/PP (UAProf) profiles. The device recognition and profile search uses the hierarchical device repository and the device

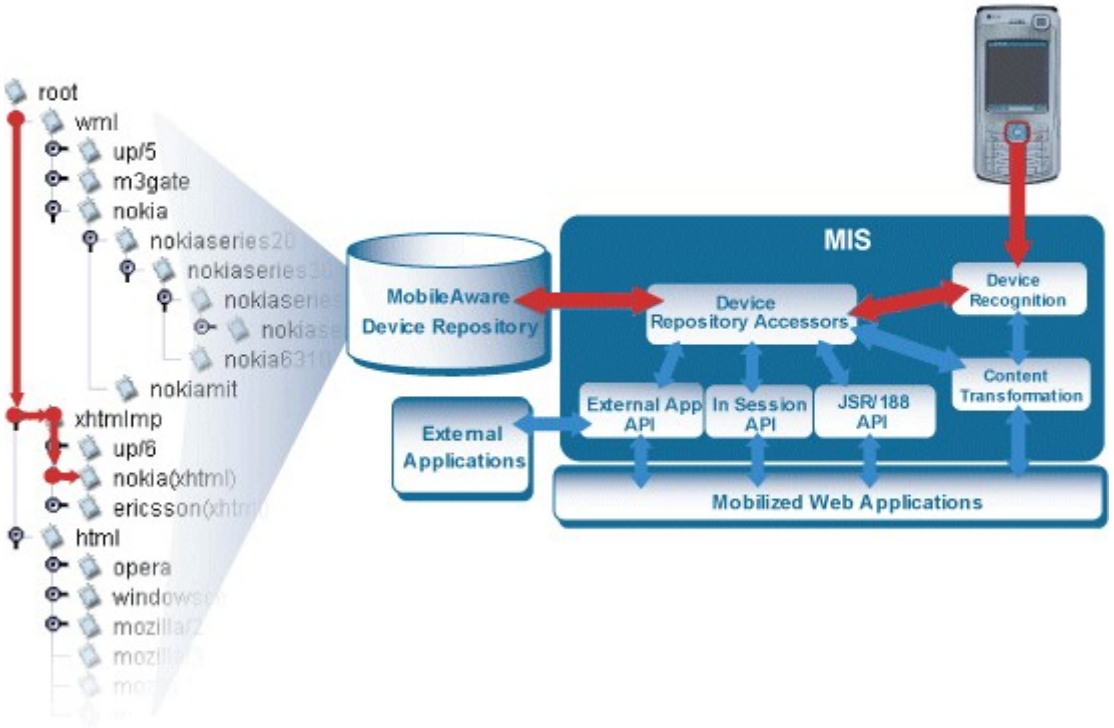


Figure 5: MobileAware - Device Recognition (source: mobileaware.com)

information submitted with the HTPP request (HTTP headers and/or CC/PP profile) to lookup the closest device profile in the device repository (Figure 5). The adaptation itself is using content and media transcoding techniques, implemented either within the interaction server (Figure 6) or alternatively a external transcoding proxy which adapts

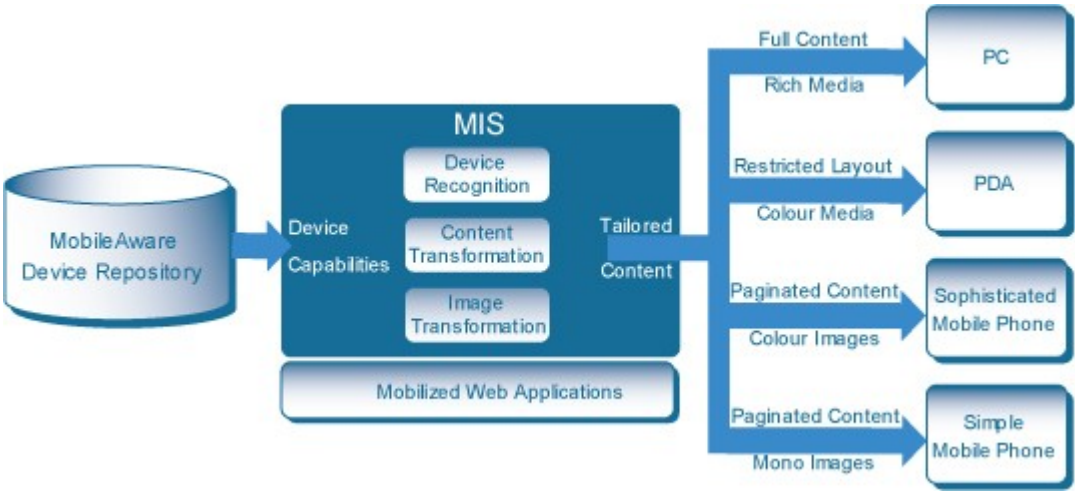


Figure 6: MobileAware - Transcoding Process (source: mobileaware.com)

the existing content (page layout, navigation links) and media (images) for a particular target device.

Remarks: MobileAware is a founding member of the *W3C Mobile Web Initiative* [W3CMWI], it has also participated in the *W3C Device Independence Working Group* [W3CDIWG] which has recently transformed into *W3C Ubiquitous Web Applications Working Group* [W3CUWA].

2.3.3 Related Research

Adapting multimedia Internet content for universal access

[MOHAN99] Even though this work is not exactly recent, we mention it because it introduces several interesting features: it uses the InfoPyramid concept [MOHAN98] for media content description (annotations). The InfoPyramid has a set of modalities (text, image, audio, video) at its base and the pyramid narrows to the top by decreasing *resolution* of individual modalities. Given the *text* modality, for example, the bottom of the pyramid corresponds to the *original* content and incrementally decreases the level-of-detail by going trough *summary* of individual textual elements to *title* and terminating at the top by *null* resolution corresponding to skipping a particular element. Similar resolution hierarchy applies to other modalities too. As this is a pre-CC/PP work, it uses a custom set of capabilities to describe the client devices: screen (resolution and color depth), network bandwidth, payload, capabilities (constraints) regarding display/playback of audio/video/image formats. What is even more important is the use of the *fidelity* function ($fidelity = \frac{1}{1+D}$) , where D represents a *distortion* of a particular resource as compared to the original content item. The value of the fidelity function equals to 1 for the original contant and converges to 0 with the growing distortion factor. The output of the fidelity function is than used as an input of the resource allocation (variant selection) algorithm. An important concept implemented in the resource allocation algorithm is the enforcement of overall device constraints (e.g. maximum page size in bytes) when choosing the individual resources.

Remarks: Similarly to [ESWA07] described below, this work only deals with individual pages adaptation, i.e., the framework does not attempt to adapt the overall application flow and/or inter-page navigation.

An End–End Approach to WirelessWeb Access

[SCCPP01] Represents an early reaction to W3C CC/PP standard. The authors argue, that CC/PP is unnecessary complex and from the practical usability perspective the contend adaptation framework needs to be simplified:

> *"CC/PP Description Framework tries to describe every possible configuration of the client machine including all little details. Such fine grained descriptiveness seems of the very limited use given that it complicates the development of the web services,"*
> *...*
> *"On the server side most details of this descriptions would likely end up being ignored, or would lead to extremely complex and hard to maintain web sites"*

As a reaction, they propose and implement a prototype of "*Simple CC/PP"*as follows: The capability description is limited to classify devices into the following four classes: PC/laptop with broadband connection, laptop with narrowband connection, PDA and a WAP device

The physical transport mechanism of CC/PP using HTTP extension headers is reused, however with the following major semantical difference: instead of actually fetching the profile over HTTP, the profile URL itself is used to map a device into one of the four classes. Nevertheless, for compatibility reasons, the authors recommend to place the actual CC/PP profile at the profile URL send in the HTTP request header

The content adaptation implements the following fall back strategy: first try to find a content variant for a particular device class, if not found, try to figure out, if the content can obtained by transforming an XML document using XSLT, the last resort is to return the content for the PC/laptop with broadband connection.

Remarks: The ideas of raising the level of abstraction and using a fall-back algorithm for finding the most appropriate content are supported by the author of this thesis. On the other hand, the simplifications proposed in the Simple CC/PP framework are substantial and limit the ability to implement fine-grained control over the content adaption process where needed: for example distinguishing WAP 1.0 and WAP 2.0 devices, or a particular version of a Java libraries present on the client device.

Enhancing pervasive Web accessibility with rule-based adaptation strategy

[ESWA07] The article describes a framework for user interface adaptation according to to the *context profile.* Context Profile consists of the following components: *Situation*, *Accessibility*, *Network* and *Device*. Each component is a closed enumeration of values, e.g. *Device* corresponds to one of the pre-defined device classes (laptop, PDA, phone). Application resources (content) are annotated on three layers: *structure layer* (a structural decomposition of the user interface – a web page[25] – into a set of objects, each having a unique ID), *modality layer* (text, image, video, audio) and *fidelity layer* (original, high, low, mute, blank). Each resource variant is annotated by a triplet (object ID, modality, fidelity). The variant selection/transformation is driven by a rule base evaluated using Jess rule engine [JESS07], a lightweight Java reasoning engine supporting *Java Rule Engine API Specification 1.0* [JSR94] developed as a part of the *Java Community Process*.

Remarks: An interesting aspects of this work are accessibility modeling, situation modelling by representing them as first class entities. Also an application of a rule based engine instead of hard-coding the rules in an imperative programming language is an interesting idea. On the other hand, the context representation is relatively high-level as the individual axes are just value enumerations without further hierarchical structuring. The sample enumerations are very coarse-grained – more resembling a laboratory experiment than a real-world richness and complexity. Additional concerns arise regarding the logical scalability (comprehensibility) of the growing rule base in case a richer (more fine-grained) context representation needs to be put in place. Similarly to [MOHAN99] described above, this work only deals with individual pages adaptation, i.e., the framework does not attempt to adapt the overall application flow and/or inter-page navigation.

25 The adaptation scope corresponds to a page level and is mainly focused on content adaption, i.e., the framework does not attempt to adapt the overall application flow and/or inter-page navigation.

Device-independent web browsing based on CC/PP and annotation

[HK06] The article describes a transcoding framework which uses CC/PP profiles to represent device capabilities (delivery context) and sophisticated hierarchical annotations to annotate the resources. Similarly to [ESWA07] and [MOHAN99], the adaptation happens on the page-level. A page is annotated by hierarchical decomposition into groups and sub-groups, each group has an importance score, primary resource variant annotated by a set of constraints referring to the CC/PP profile (similar to [CAPCLASS]) and also each group can contain one or more alternatives ordered by statically determined (constant) *fidelity score* and a set of CC/PP constraints expressed in the same way as constraints for the primary resource variant. The substitution algorithm tries to use the primary resource and if it does not pass the constraints, it tries one alternative after another in order of decreasing fidelity score until it passes the constraints determined by the CC/PP profile of the device. Besides local resource substitution, the framework allows for skipping less important groups and paginate the original page (designed for a laptop or a PC). The pagination algorithm leverages the hierarchical groups annotations to split the page on individual group/sub-groups level and generate the navigational map (hierarchical menu) automatically.

Graceful Degradation: a Method for Designing Multiplatform Graphical User Interfaces

[FLORINS06] is a doctoral thesis which builds on an earlier work [FLORINS04], thesis statement goes as follows:

> *"The design and development of multiplatform user interfaces benefits from a semiautomatic, model-based, transformational approach which applies transformation rules to a source model, conceived for the least constrained platform, in order to produce one or several target models, adapted to more constrained platforms."*

The work presents a model-driven framework which adapts the referential (PC or laptop based) web application for (constrained) mobile devices. The core of the framework is UsiXML [USIXML], a meta language for for defining application models on the following four levels:

1. Tasks & Concepts: *task* and *domain* models
2. Abstract User Interface: *AUI model* + *resource model* + *Interactor model*[26]
3. Concrete User Interface: *CUI model* + *resource model* + *Interactor model*
4. Final User Interface: actual software artifacts

Besides the layers above, there are additional models which do not belong to a particular abstraction layer: *mapping model* for defining relationships between the models above, *context model* (consisting of *user model*, *environment model* and *platform model*) and *transformation model* defining a rules sets for inter-model transformations. The process of *graceful degradation* is a sequence of transformational steps from the most abstract representation towards the final user interface artifacts. The task model of UsiXML builds on an extended version of *ConcurTaskTree*[27] (CTT) [CTT00]: *"a hierarchical task structure, with temporal relationships specified between*

26 AUI model is an instance of the interactor model – a meta-model of a particular abstract of concrete widget library, resource model contains the static resources like labels or images

27 Another framework using cascade of models and CTT is *TERESA: a transformation-based environment for designing and developing multi-device interfaces* [TERESA04].

sibling tasks." [FLORINS06]. The *platform model* is using CC/PP (UAProf) described in section 2.2 to represent device capabilities.

Tool-supported single authoring for device independence and multimodality

[SIMON05] Discusses a methodology (authoring method) and a model-driven framework for multimodal application development using single authoring approach. The motto of the article is to support the development of multimodal applications while preserving the traditional workflow the web application designers are used to:

> *"One objective of the project was to devise a more "developer-friendly" single authoring method for cross-platform user interfaces; i.e. a method that enables a smooth transition from today's work practices"*
> *...*
> *"Within our project, the fundamental assumption was that – despite platform-independence – the traditional workflow of designers should be preserved as much as possible."*

The methodology presented in the article uses an iterative approach to develop a series of prototypes for selected device classes (PDA, smart phone, wap phone and a voice interface), starting with one device class and subsequently adding more device classes. Each prototyping cycle is concluded by user acceptance testing (UAT). Once the prototyping phase is completed, the development team has sufficient knowledge to generalize the user interface model to an abstract MONA UIML (User Interface Markup Language), which has been developed by the same team as a part of the MONA project [MONA05], [MONA]. When the application development is completed, the concrete user interface representations are generated by transformations based on the delivery context (target device class) and the UIML model. An Eclipse based tool has been developed as part of the MONA project to support the development development process. The authors also discuss their standards convergence plans to modify UIML so that instead of using custom abstract widgets, the W3C XForms [XFORMS] will be used. The additional papers discussing discussing various aspects of the MONA project are [SCHATZ05], [BAILLIE05] and [TMN04].

Context-Aware Adaptation for Mobile Devices

[MDM04] and [SAINT03] describe an adaptation framework, which uses Universal Profiling Schema [UPS02] (from the same authors) to describe *delivery context* and server *resources*. The UPS is inspired by CC/PP and UAProf, however it uses a different vocabulary than UAProf and extends the coverage by not only describing the client device but also the server resources. The client profiles defined in UPS are *Client Profile* (Hardware platform, Software platform and Browser user agent) and *Client Resource Profile* (device constraints regarding the individual content categories). The server profiles are *Document Instance Profile* (includes document instance description, multimedia content, adaptable resources description), *Resource Profile* (media resource description, adaptable resources description) and *Adaptation Method Profile* (adaptation resource description, adaptation method description).

The delivery context is determined by the client profiles. The content repository is represented as a web service supporting XQuery [XQUERY] language to express the constraints of the delivery context. The resources in the repository are annotated using the server profiles. The repository supports two modes: *a negotiation module* and

adaptation module. The negotiation module is used to retrieve an appropriate variant of an existing resource, while adaptation module is applied in cases when content adaptation becomes necessary, either a *structural* transformation or *media resources* transformation is applied.

Structural transformation (for content like XML, XHTML, SMIL) uses *XSL Transformations* [XSLT] standard. It is actually a two stage process: first a concrete XSLT template is generated from the XSLT meta-template using the delivery context, second the concrete template is applied to the actual resource. The structural transformation implements a *semantic hierarchy adaptation* similar to [MOHAN98] so that it allows to choose a level of detail appropriate for a particular device. Similarly to [HK06], the structural transformation supports pagination and navigational links generation. The media resources transformation is typically implemented in Java or as a library embeddable to Java.

*Remarks:*This works seems to ignore the fact, that RDF (and in turn CC/PP) is quite benevolent in terms of serialization of RDF graphs into XML: the same semantical statements can be encoded as XML elements or XML attributes, this effectively disables usage validation of RDF documents using XML Schema language as the validation needs to be done on the semantical level, not the syntactical one. The same issue applies to using XQuery (which uses XPath) to extract data from RDF documents. A query language specifically designed for RDF [SPARQL] is now under development in W3C. The combination of CC/PP and XQuery may only work assuming XML serialization conventions are systematically followed for CC/PP XML serialization.

Experiences in Using CC/PP in Context-Aware Systems

[Indulska03] Similarly to [UPS02] above, this work employs CC/PP with custom extension vocabularies to represent richer delivery context that UAProf standard. The authors use CC/PP to model *LocationProfile* (physical location – address, logical location – IP address, geodetic location – coordinates, orientation and modifications), extended *NetworkCharacteristics* (disconnection status, quality of service). Besides the delivery context, the authors leverage CC/PP for expressing *application requirements* and *current session*. Based on the extended context and requirements information the article also discusses a three-layer *Context Management Infrastructure* constituted from a set of *sensors*, *actuators* and *awareness modules* on the first layer, *context manager* and *context repository* on the second level and the actual *context aware application* on the third layer. The layers of the architecture communicate using *subscribe/notify* mechanism: the awareness modules communicate with the context manager, which in turn updates context repository and notify subscribers on the application layer.

2.4 Important Observations

In the following subsections, we are trying to outline in more detail some interesting issues we have briefly touched in the sections above. We consider these topics to be very important from the perspective of configuration management and content adaptation. We believe that the fact these issues are not sufficiently resolved by the existing standards and technologies for content adaptation represents the key reason behind the unsatisfactory situation regarding their adoption in the day-to-day practice.

2.4.1 Metadata Consolidation

An important aspect of applying metadata to version and variant control is the ability to consolidate information acquired from various information sources and define resolution rules used to combine (sometimes contradictory) information pieces into a single coherent view. None of the technologies described above addresses this issue completely. For example CC/PP together with UAProf define resolution rules[28] required for merging complete profiles and partial profile-diffs, as well as handling the CC/PP *default* construct in addition to that. However, the standard does not provide any means for handling possibly overlapping data coming from other sources than CC/PP.

Let us demonstrate this issue using probably the most frequently used metadata attribute used for content adaptation: the *locale*[29] information. In a web application, the locale information may be a part of the HTTP header *Accept-Language* as described in section 2.1.1. If a client device supports CC/PP and UAProf, the very same information can be delivered as a part of the UAProf profile: the *CcppAccept-Language* attribute of the *SoftwarePlatform* component. Both *Accept-Language* and *CcppAccept-Language* belong to the HTTP request scope so they can potentially change with every HTTP request. They are set-up in web browser preferences of the client device. In addition to these attributes, many web applications incorporate locale selection directly into their user interface: Remember the familiar *country flag* icons found on many web sites? The users can choose their preferred language without the need to modify the browser preferences. This type of locale selection is typically implemented using *cookies*. Last

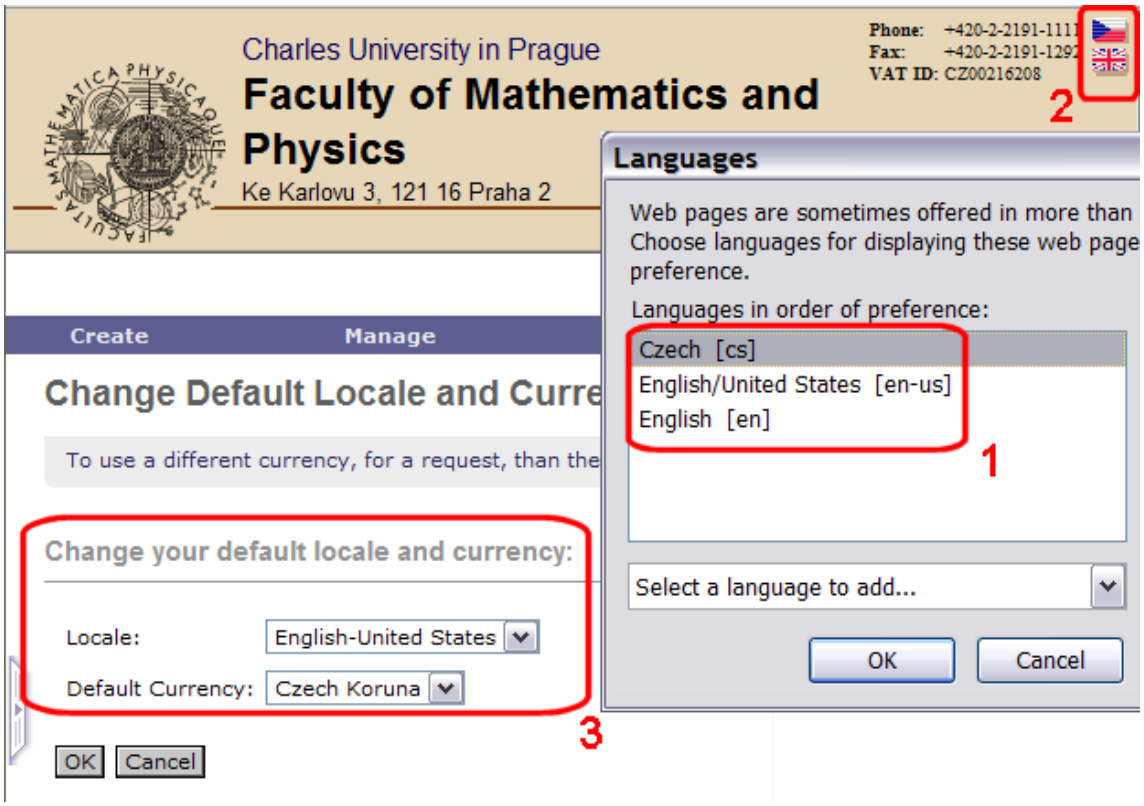


Figure 7: sources of locale setting (1) web browser, (2) application, (3) user profile

28 UAProf defines three resolution rules: *Locked, Override* and *Append*, the last one only applicable to lists

29 *Locale* – typically a *language* and optionally also *country* ISO codes, see also [LC142]

but not least, some applications, which require a user registration, store locale preference as a part of server-managed user profile.

In a general case, the locale information can be extracted from the following three sources, as depicted on Figure 7 above: (1) web browser settings (*Accept-Language* or *CcppAccept-Language*), (2) web application settings (a session or a persistent cookie), (3) a server-managed user profile

Given the above, there could be various resolution rules and strategies in place to decide which source is to take precedence over the others if conflicting values are retrieved from multiple sources and, on the other hand, to fall-back to a less-trusted metadata source in a case the preferred one is unable to provide the requested value. A typical order of precedence could look like:

1. server-side user profile
2. cookie (web application setting)
3. web browser settings (UAProf or HTTP header)
4. application default

In other words: if the user is a registered user, use the locale value from the user profile, if not, see if the user has expressed his/her preference by clicking a flag on the web site, if there is no such preference set, see what the web browser states in the *Accept-Language* HTTP header, if the browser does not state *Accept-Language* nor does it support UAProf, use the application default locale.

There could be many different application-specific variations of the above, pending the nature of the application. One could for example consider the choice in the web application user interface to temporarily override the permanent user profile settings or prefer the web browser settings over all the others. The key take-away from this lesson is, that no matter how many different resolution strategies we can envision, it is not possible to enumerate all of them. The goal should be to provide a flexible framework which allows the web application designers to define their own strategy for resolving a particular metadata attribute from multiple sources, while providing an abstraction layer which let us them to achieve this goal without the need to deal with the technical details and differences between those sources.

2.4.2 Metadata Canonicalization

Another important and often overlooked topic is *canonicalization*, i.e., mapping all syntactical variations of the same semantical entity to a single canonical representation.

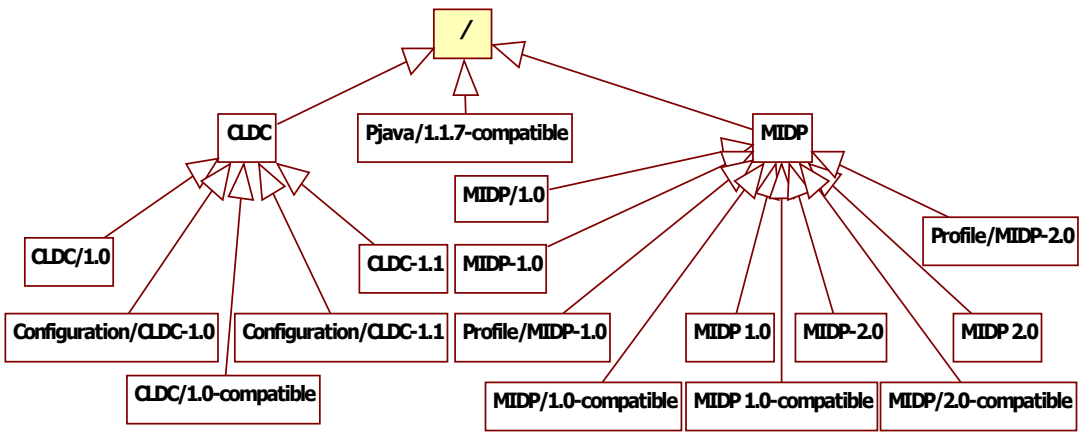


Figure 8: UAProf JavaPlatform attribute values (raw sampling)

This is closely related to the issue of consolidating multiple different metadata sources discussed above, as the likelihood that different systems represent the same data differently is inherently higher, but as we show in the following example it can represent a severe issue even within a single metadata domain.

The Figure 8 above shows a taxonomy built by sampling the *JavaPlatform* attribute of the *SoftwarePlatform* component in available UAProf repositories [UAREP1], [UAREP2] representing the actual mobile devices produced by various vendors. Obviously the *JavaPlatform* attribute is not very well defined in UAProf, as it is in fact overloaded to represent two different attributes: J2ME *Configuration* and J2ME *Profile*. From the canonicalization point of view, we can see that the value-set of the taken sample contains many redundancies, meaning that different device vendors and even different product lines are not using metadata values consistently.

We tried to clean up the value set, establish a naming convention and canonicalize the raw *JavaPlatform* attribute values. The result can be seen on Figure 9. The original raw values still remaining in the taxonomy have clear background, new nodes introduced in order of the naming convention are in italic with yellow background. For the purpose of the taxonomy, we semantically distinguish the nuance of *being an implementation* of a certain J2ME configuration or profile and claiming to *be compatible* with. Even though this distinction can most likely be ignored on most occasions, we try to make sure our canonicalization process does not loose any semantical information contained in the original raw data set. Also, we did not extend the taxonomy in any way by enriching it with additional information not represented in the raw data sample, for example, there was no device claiming to be CLDC 1.1 *compatible* and thus there is no such a node in the cleansed taxonomy on Figure 9.

The lesson learnt from the example above is, that even though there are rigorous standards in place, like UAProf, it is quite dangerous to directly employ unsupervised metadata in application versioning as there is often a need to cleanse and canonicalize the data before they can be meaningfully used. The issue of canonicalization is usually amplified in case of consolidating multiple independent metadata sources.

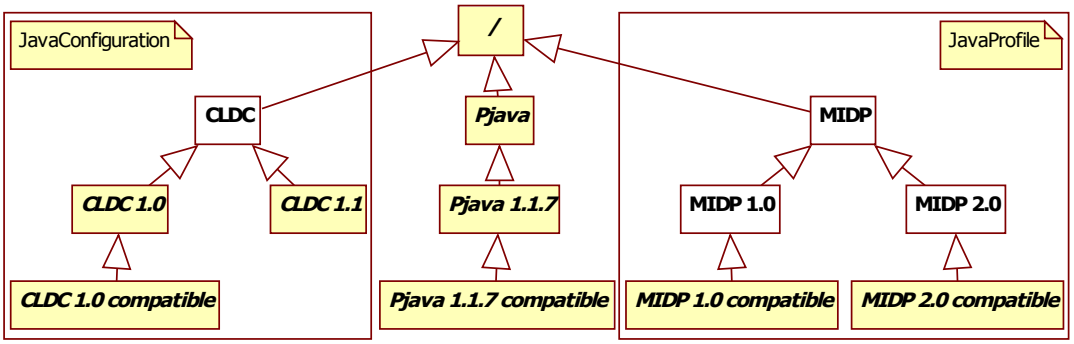


Figure 9: UAProf JavaPlatform attribute values (after manual cleansing)

2.4.3 Metadata versus Knowledge

Throughout the history of computer science, there were several attempts to popularize various frameworks allowing dynamic (just-in-time, on-demand) binding of software components or software services. Examples of these could be the CORBA *Trading Object Service* [CTOS], Sun's *Jini* technology [JINI] or the most recently *Universal*

Description Discovery & Integration [UDDI]. We put the *CC/PP* and *UAProf* discussed in section 2.2.1 on the list too, even though they are tailored more towards content adaptation than dynamic component binding. What is common to all these systems is that they provide a *framework* for resource annotation and retrieval. Some of them, for example UAProf, even define a concrete *vocabulary*, or better to say a *meta-model* of their subject domain specifying what attributes are used to represent the resource properties as well as their data types and/or format.

The common problem of all these frameworks is, that they stop on the meta-model (vocabulary) or even meta-meta-model (generic framework) level and do not provide sufficient constructs and tools to sample, analyze and efficiently leverage the actual metadata – the value sets found in their repositories describing and the existing software components, services or resources. As we show in this section, even having a typed vocabulary in place is insufficient to start developing an application while utilizing the underlying meta-model: there is still additional information needed and even though many frameworks mentioned in the quite formal and rigorous, they come short on this point and somehow expect that the missing unspoken-of information is informally or even miraculously added into the mix to make the framework actually work.

Let us start with a simple motivational example to make the point: the task is to adapt the user interface according to the device's screen size. UAProf defines attribute *ScreenSize* of type `Dimension` within its *HardwarePlatform* component. UAProf `Dimension` *is a pair of positive integers* [JSR188]; and in the case of *ScreenSize* it represents screen width and screen height in pixels. Given the information provided by the UAProf RDF Schema, assuming *integer* is meant to be the commonly used 32 bit signed number[30] and while not having any other information, one could assume *normal distribution*[31], split the available range by half in both axes (*width*, *height*) and define *small screen* devices as those having *ScreenSize* smaller than `1 073 741 823 x 1 073 741 823` pixels and *large screen* devices as those having higher resolution than that. For anybody familiar with typical screen sizes (display resolutions) used in today's PCs, workstations, laptops or PDAs and web phones, the example above most likely appears totally absurd: it is clear that 100% of the existing devices would belong to the *small screen* class, so there is no point developing a specific screen layout for the *large screen* devices as defined above.

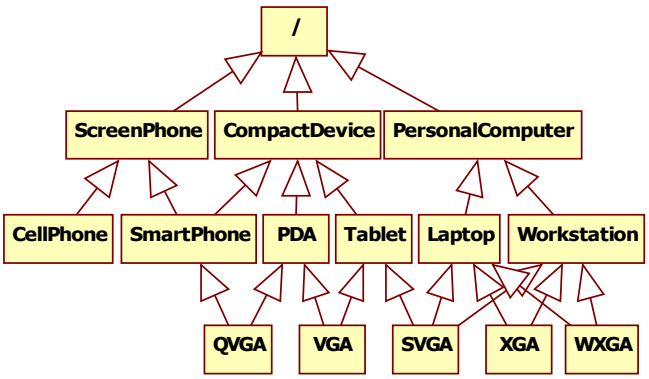


Figure 10: An example of a generic ScreenSize classification hierarchy

30 Curiously, the exact datatype is not mentioned anywhere in [UAP06].

31 From statistics, normal distribution, a. k. a. *uniform distribution* – all values occur with equal probability

The problem is that the UAProf vocabulary – the meta-model describing the properties of web-enabled portable devices – does not give us any hint regarding what is the actual distribution of values within the `Dimension` domain nor does it mention the fact that screen resolutions are actually highly standardized and majority of the devices on the market use one of a relatively few well-known resolutions[32]. To make the *ScreenSize* attribute useful, so that it helps us to classify devices into clusters suitable for developing tailored screen layouts, we need to bring in the additional information about display resolutions of the existing devices on the market and construct a mapping from the `Dimension` value space onto our *ScreenSize classification taxonomy*.

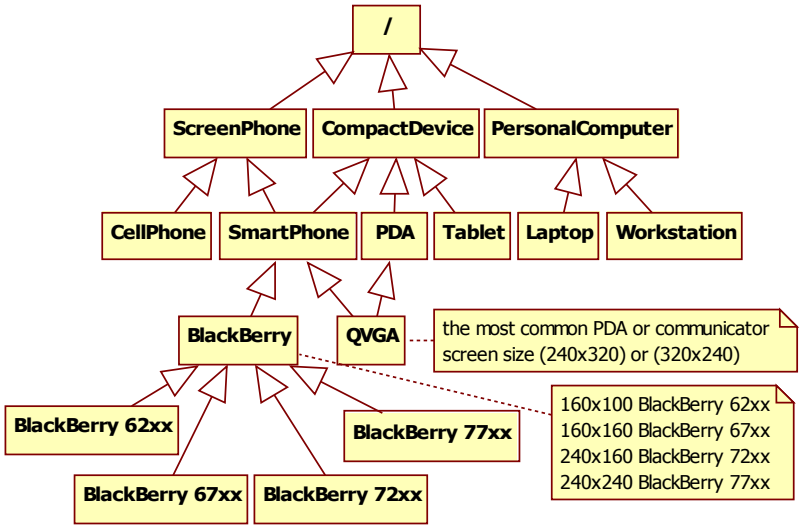


Figure 11: An example of a specialized ScreenSize classification hierarchy

In order to maximize the gain from this exercise, the taxonomy needs to be constructed in a way so that their nodes directly correspond to the variants we want to support in our application. It is important to choose the right level of detail, so that we can dive down into the classification hierarchy if a need to distinguish subtle nuances between devices arises, but at the same time, we need to be able to abstract away from the details and cluster devices efficiently together, so that we can share resources and artifacts across application variants whenever appropriate: Figure 10 shows a portion of a generic (potentially reusable) classification hierarchy, while Figure 11 shows an example of a taxonomy specifically focused on mobile business applications market where it captures much higher level of detail.

32 Well known resolutions (examples): QVGA (320x240), VGA (640x480), SVGA (800x600), XGA (1024x768), etc.

Let us go through another example to demonstrate how little information is provided by a meta-model alone: UAProf *Keyboard* attribute of the *HardwarePlatform* component is defined in the RDF Schema as a `Literal` which corresponds to an arbitrary string. Any idea what the actual values of this attribute may look like? A quick scan through the publicly available UAProf repositories turns out the result (Figure 12): besides the canonicalization problem discussed in the previous section, we observe that even the data sample does not provide much information regarding the qualitative measures of particular devices with the respect to entering data. For example, whether a device is

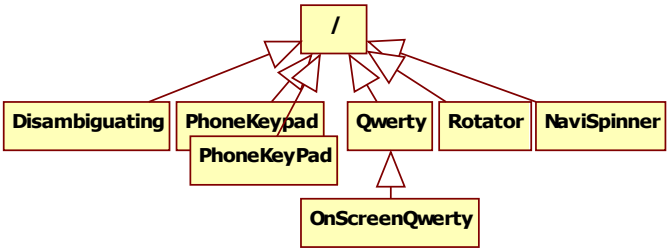


Figure 12: UAProf Keyboard attribute values (raw sampling)

able to accept alphanumeric text and how fast/convenient such an input is from the end user perspective. Similarly to the *ScreenSize* attribute, also the *Keyboard* attribute requires additional work to investigate the semantics of the individual literal constants collected by data sampling and construct a taxonomy capturing the information needed to classify the devices with the respect to their data input qualities (Figure 13).

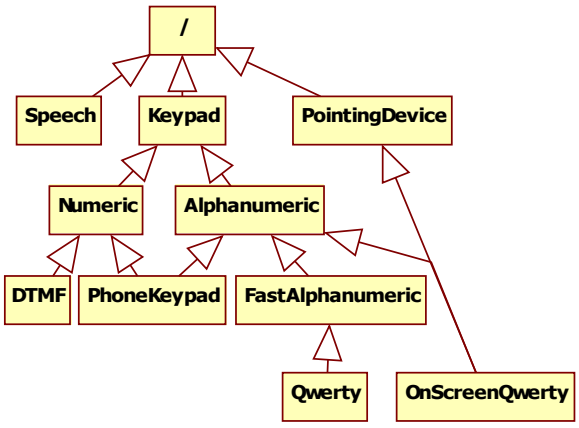


Figure 13: A custom InputClass classification hierarchy

To summarize this section: It is not sufficient to rely just on a framework (CC/PP) or even a domain-specific meta-model (UAProf) when trying to employ metadata for versioning and variant support. Several steps need to be taken to cleanse and enrich the raw data in order to truly realize the potential provided by a framework like CC/PP:

1. Proceed with representative data sampling to have an idea about the actual attribute values, this is a prerequisite for being able to implement canonicalization as per section 2.4.2.
2. Build a hierarchical classification (taxonomy) which is necessary for tagging application resources. Annotating using with the use of a taxonomy has an

advantage of being able to attach a more generic tag value to a resource which can be shared between multiple variants.

- One can re-use an existing standard taxonomy[33], if applicable, refine an existing generic taxonomy to provide more detail where necessary or develop a custom taxonomy specifically tailored for his company and/or a particular application.
- When building such a taxonomy and deciding what level of detail to choose, it is useful to perform targeted market analysis and/or statistical analysis of the existing traffic on the web site, to figure out what clients (devices) are likely to access your application and even more importantly how often: It makes sense to dive into more detail for the devices which have a significant market share in your application domain while *default* to common artifacts for marginal devices.

3. Building and maintaining taxonomies for individual meta-model attributes is expected to be an iterative (recurrent) process. After boot-strapping an application based on the initial analysis and survey, one can revisit the taxonomies based on the actual traffic and repeat this step on regular basis to keep the taxonomies up to date. Also, as many new devices are emerging on the market every year and due to that the capabilities of an average device in each category are shifting over time, it is necessary to update the mappings from raw metadata to taxonomies[34].

2.5 Background Conclusion

We have started this chapter with two motivational case-studies to introduce the reader into the problem domain. We followed by describing and evaluating the current state-of-the-art standards based technology stack. We also discussed selected representatives of open source, commercial and research works related to the topic of this thesis. To our best knowledge, the works presented include the most relevant related work in each category, with the focus on the most recent work in the domain, aiming at developments taking place after year 2000.

We compare the versioning and adaptation framework presented in this thesis to the standards stack (2.2) and the related work (2.3) in section 6.3 (Related Work Evaluation).

33 For example UNSPSC (United Nations Standard Product and Services Classification) or NAICS (North American Industry Classification System)

34 For example, today the average PDA ScreenSize is QVGA (240x320), while in a year or two, it is likely to be twice as much – VGA (480x640).

2.5 Background Conclusion

3 Setting the Goals

Today, developing multimodal or multi-variant applications is too complex and expensive which often forces the application providers to focus on a few blockbuster devices while abandoning the rest of the market. The author of this thesis believes, that by providing the right tool-set to the application designers and developers, we can spur a wider adoption of the existing standards for capturing device capabilities and user preferences, which would ultimately lead to a richer user-experience available for a broader audience.

The aim is not to replace the existing standards, it is to implement an integration layer on top of the existing technologies to provide a *consolidated* and *application-centric* view of the versioning and configuration metadata to the application designers and developers. The goal is to practically enable the *single-authoring* approach, i.e., developing all the variants in parallel in a single framework and share as much artifacts as possible between those variants to reduce the amount of redundant work.

As demonstrated in sections (2.2 and 2.3), there are various existing standards and technologies for describing device capabilities and user preferences (metadata), yet problems arise while trying to efficiently *interpret* and *consume* the information provided by these frameworks when building the actual application. As we are trying to bridge the gap remaining between the current metadata technology stack (Figure 3, page 20) and the application itself, we need to keep in mind the known issues and make sure that these are addressed and resolved. Let us summarize the essential findings here in a few bullets, as it will make easier referring to them in the later chapters, when evaluating whether the proposed solution fulfills the goals set:

1. *Metadata Consolidation:* Metadata are coming from various overlapping sources (configuration files, server-managed user profiles, HTTP headers, UAProf profiles). It should not be left up to the application developer to consolidate all these information sources and define resolution rules by embedding them the application code. (section 2.4.1, and also *Supporting multiple vocabularies and vocabulary versions* on page 21)

2. *Metadata Canonicalization:* Practical experience shows that raw metadata sources can not be blindly trusted. There are multiple reasons for that situation (see *Producers, consumers and beneficiaries*, *People are inherently fallible* on page 21) and therefore there is a need to cleanse and canonicalize metadata before they can be manipulated programmatically. (2.4.2)

3. *Level of Abstraction Gap:* The metadata sources, as being domain-specific and application-agnostic at the same time, often provide the information on a different level of detail and using different terms than a particular application needs. It is highly desirable to transform and enrich (pre-process) the metadata so that they represent the actual knowledge directly realizable by a particular application.(2.4.3) Pre-processing data for a specific application may also help to mitigate some of the other issues discussed above (*Classifying information is inherently hard*, *Metadata is inherently biased*, page 21)

4. *Domain Expertise Issue:* The existing W3C standards for metadata annotations look like they were designed *by* the experts in the field of knowledge management and meta-modeling *for* the experts in the field of knowledge management and meta-modeling. They are based on the foundations of *Semantic Web,* which makes them very generic and powerful, but it also makes them quite difficult to learn and use in a day to day practice for those, who are not experts in the meta-modeling domain. There is a need to insulate the application developers from the complexity of the metadata frameworks and present metadata in a form which corresponds to the application architecture and design perspective. In other words: in addition to the existing metadata-centric tools, which are designed for metadata modeling and manipulation, we need to provide simplified tools solely focused on metadata consumption. General knowledge of web-based authoring, object oriented programming and design should be sufficient for being able to use the framework.

5. *Best Practices Enforcement:* Another important requirement, which needs to be addressed, is the ability to *maintain and evolve* the application over time while containing the total cost of ownership. It is too easy to let the versioning logic *proliferate* to the application logic, leading to unmanageable spaghetti code base. The proper solution for the versioning logic is apply *separation of concerns* and separate it from the application code in a way similar to the technique used for separation of business rules or branding from the core code in order to be able to flexibly and cheaply modify these cross-cutting concerns without the negative regression impact on the entire application. As of the versioning logic itself, we need to ensure proper *modularization* of the versioning rules, to encourage reusability and avoid ending-up with one incomprehensible cloud of code which is hard to comprehend, evolve and maintain.

4 Addressing the Goals

4.1 Design Considerations

Before starting to actually evaluate possible approaches to addressing the goals of this thesis, let us discuss the functional and technical consideration which need to be taken into account. In the Functional Considerations section we refine the goals stated in the previous section. The technical aspects like performance are discussed in Technical Consideration section.

4.1.1 Functional Considerations

- *Metadata Consolidation:* when merging data from multiple, possibly overlapping metadata sources, we need to be able to:
  i. define priority of individual metadata sources to ensure the resolution rules unambiguous
  i. implement a safe fall back mechanism, for the cases the preferred source(s) are unable (temporarily or permanently) to provide desited data
  ii. define the resolution rules on the level of individual metadata attributes, because each attribute has different semantics

- *Metadata Canonicalization:* the canonicalization process can be as simple as using a mapping table to map a set of well-known alternatives to a single selected value, or as complex as a need to parse and interpret composite literal values (see section2.1.1, *User-Agent* HTTP header) or to employ an algorithm to determine the canonical value given an unconstrained value space, and a set of interpretation rules. Therefore we need to make sure the canonicalization feature is very flexible and allows the user of the framework to choose his/her preferred tool most appropriate for a particular situation (mapping tables, declarative rule-based languages, imperative programming languages, external services – e.g. a statistical data analysis)

- *Level of Abstraction Gap:* next to canonicalization, is constraining the values further using *classification.*
  i. The simplest form of classification is to introduce a *controlled vocabulary*, constructed from the source value set by applying some sort of equivalence relation (similar as in the case of canonicalization) to factorize the source data into a set of equivalence classes represented by the values of the controlled vocabulary.
  ii. More advanced option (if applicable) is to implement *hierarchical classification* (a *taxonomy*), as presented on many examples earlier in this thesis. The major advantage of using the taxonomy over the flat controlled vocabulary is to represent generalization/specialization relation which can in turn to be used to implement a fall-back strategies, both on input side (an unknown attribute value can be mapped to the *default* root value) and on the output side (if there is no such resource variant having the desired attribute value, we can progressively generalize the requirement until reaching the closest available resource variant – closest in terms of a given taxonomy)

iii. Besides classification, we need to be able to represent relations between attribute values, which are present in many ontologies and meta-models to represent relationships between classes and/or instances.
iv. One specific case is a need to support an explicitly stated *ordering relation*: ordering of attribute values can depend on a particular situation (point of view): the author of this thesis witnessed a situation when a customer was trying to claim a computer game he bought with the sticker "Windows 95 or newer" and was not able to run it on his Windows NT 4.0 workstation. The customer (without realizing it) implicitly assumed *total ordering* determined by the time axis (Windows 95 was released 1995 while Windows NT 4.0 in 1996) and apparently was not aware of the fact, that that at the time Windows operating systems family constituted of two separate product lines, and the order relation was in fact a *partial ordering*.

- *Domain Expertise Issue:* In recent years, the development of (web) applications became increasingly complex due to the high fragmentation of development tools market and ever-growing amount of technologies tools and languages needed to master in order to become productive in the end-to-end application design and development. Moreover, the developer needs to change mindset very often when switching from one technology or language on the stack to another. The aim of this work is to try to avoid further complication of the application development landscape by limiting itself to the common tools and technologies the application developers are already familiar with and only use such concepts which are easily transferable to other commonly used tools of similar expressive power (e.g. Java versus C#).

- *Best Practices Enforcement:* The aim is to provide guidance which naturally leads to *separation of concerns*, encourages *modularity* and avoids *proliferation* of the versioning logic in to the core application code. If these goals are met, it helps to increase overall robustness, long-term manageability and evolvability of the system. On the other hand, the framework should not interfere with the development proces and practices: frameworks which strictly enforce some particular methodology throughout the entire development process are harder to integrate into the existing systems and tend to discourage the developer to adopt them. Given the above, the goal should be to enforce the properties mentioned above only withing the framework, but avoiding infliction of the entire application: from the application perspective, the framework should act as a library easily pluggable into existing as well as newly developed applications.

4.1.2 Technical Considerations

Modern object-oriented frameworks like the CATCH2004 Multi-Modal Portal ([ICSM2001], [IIWAS2001] or Sun Microsystems Java Server Faces [JSF] use server-side widget libraries very similar to those used on desktop computers. Such libraries support hierarchical composition of widgets, layout definition, event bindings and last but not least a pluggable look-and-feel which is needed for user interface adaptation for different client platforms and/or modalities.

When talking about multimodal applications as defined in this thesis, we may consider the following kinds of adaptations taking place to accommodate an application to

a particular device and adapt the application as the device settings change during the session:

- **Application Dialog Flow**: depends of modality or a set of modalities supported by the client device, for example: XHTML+Voice, if the user set device on mute (or on contrary to hands-free/eye-free mode – e.g. while driving) during the session, it becomes unimodal and the dialog flow needs to be re-drawn accordingly
- **User Interface Layout**: depends on device capabilities, modality and/or screen orientation (if applicable) which may change many times during the session
- **Individual Widget Look-and-Feel**: depends on device capabilities, modality and screen orientation (if applicable), a set of widget instances in use can be different in each turn, there can be tens or hundred of instances on each "screen" (considering a PC application)
- **Static Resource Variant**: depends on device capabilities, modality, user's language, each widget can have tens of resources like labels, messages, images

Adaptation Kind	Adaptation Scope	Approximate Number of Occurrences
Application Dialog Flow	session	typically less than 10 times per session
User Interface Layout	request	typically less than 100 times per session
Individual Widget Look-and-Feel	request	tens or hundreds per request
Static Resource Variant	request	Hundred or thousands per request

Table 1: Adaptation kinds, scopes and estimated number of occurrences

Given the estimates above, which apply to a rich user interface of a PC-like client device, one thing becomes immediately clear: the adaptation process needs to be designed so that the resource variant selection (adaptation) is fast enough so that the delay does not discomfort the user. There can be thousands of instances of resource adaptations in a single turn (client-server round trip) in case of rich user interfaces like PCs and laptops. If possible, the results of reasoning for individual resources and delivery contexts should be cached, so that the next instance of adaptation request for a given resource is much faster and less server resource demanding in cases when the relevant portion of the delivery context has not changed.

4.2 Possible Approaches

4.2.1 Web Ontology Language

While looking at how to best address the goals stated above, the first place to look for a solution is the W3C stack of *Semantic Web* technologies. The reason for that is the fact, that CC/PP and UAProf are built using technologies (RDF, RDF Schema) which belong to the *Semantic Web. Web Ontology Language* (OWL)[35] is an XML language which builds on RDF Schema and extends it with additional constructs for defining ontologies and meta-models:

35 The abbreviation for the Web Ontology Language is surprisingly OWL.

> *"RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes. OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes."* [OWL04]

OWL comes in three increasingly powerful editions: OWL Lite, OWL DL (DL stands for *description logics*) and OWL Full. OWL DL and OWL Full share the same set of language constructs, yet OWL DL imposes certain restrictions on those constructs to ensure computational completeness and decidability, therefore for the purposes we would like to employ OWL, the OWL DL seems to be most appropriate.

The idea is to let CC/PP (UAProf) to represent raw (source) data, OWL to represent a set of additional rules governing the data (ontology, meta-model) and a reasoning engine to process the two and present the results in a consolidated form suitable for the application development. Out of the four requirements listed above, OWL seems to best suit the third one – to overcome the *level-of-abstraction gap* by creating appropriate classifications and ontologies. With regards to the *metadata consolidation* (combining multiple sources and defining resolution rules), this is also achievable in OWL, but partly remains out of scope – especially the property value acquisition from external sources: all information needs to be represented in RDF to make it available to OWL reasoning. On the *metadata canonicalization* point we hit the first serious weak point of OWL: as it lacks arithmetic primitives and string (regular expression) operations, it may be extremely difficult if not impossible to implement required data cleansing entirely in OWL. This shortcoming may also affect data transformations, for example: given the *ScreenSize* (`Dimension`) from UAProf and trying to construct a *ScreenOrientation* property (Figure 14) based on screen *width* and *height*, there is no straightforward way to do that in OWL, even though it is a trivial task using most programming languages. Quite understandably, OWL can not meat the fourth point on our requirements list (due to the definition of the requirement): *domain expertise issue*: being on the top of the *Semantic Web* stack, it suffers it requires its users to be an expert in the field of Semantic Web, ontologies and meta-modeling to make for an efficient use of its powers. Regarding the *best practices enforcement*, OWL does not support modularity – a notion of *ontology modules* with clearly defined public interfaces. With growing ontology bases, this can represent a serious maintenance and evolutionary issues.

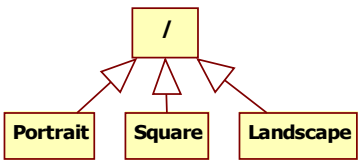


Figure 14: ScreenOrientation derived from ScreenSize

To wrap up the OWL technology: OWL does not let us to fully reach our goals stated above. It comes short in some areas which are easy to handle in common programming languages. On the other hand, only a portion of its features is needed in our domain of interest, the bulk of its features is not necessary for our needs, which makes us to attempt to design such an framework, which will consider application of OWL as an optional component.

4.2.2 Rule-Based Systems

Majority of the reservations which apply to OWL apply to the other knowledge representation systems too. They are usually based on *first order logic* (PROLOG[36], KIF[37], Common Logic[38]) and they suffer the same functional mis-fit (for our purposes) as OWL: some commonly needed features are missing or cumbersome to implement, on the other side, the languages wield too much of an expressive power which can very easily lead an unexperienced developer to accidentally design rule sets of extreme computational complexity. Another issue with the rule-based systems above and also rule-based engines like Jess [JESS07] is the problem of maintainability and the ability to evolve and maintain such a rule base as it grows and gets complicated.

Considering our goals set in the previous section (3), the rule based systems in general do not address the *metadata consolidation* and *metadata canonicalization*, they excel in raising the *level of abstraction*, they may suffer from the *domain expertise issue* as we do not assume a web application designer to be fluent in usage of rule-based engines. Last but not least, the rule-based systems may suffer from insufficient *modularity* of their knowledge bases.

As a result of the above discussion, we do not rule out usage of rule-based systems completely, however, we aim for these tools not to become the centrer piece and a prerequisite of the framework, and rather let the users to plug them into the framework a well-defined manner if needed.

4.2.3 Ontology Definition Metamodel

Besides the functional aspects, there is also the issue of the language gap: while CC/PP and OWL are using XML syntax, and others are also either using XML or their own proprietary syntax, majority of the applications are developed using an object oriented languages like Java, C# or Python. As soon as another language is being integrated in to an application, there needs to be a bridge translating concepts between the two languages to let them communicate. Such a bridge is typically presented in a form of an Application Programming Interface (API) expressed in terms of the "client" language (Java, C# or Python referring to our list above) making the features of the "server" language (OWL, KIF, XML or SQL) accessible in the client environment. Such an API can be language specific (allowing to embed a particular language) or generic, allowing to embed a family of similar languages. To our knowledge there is only one attempt to implement a generic interface for knowledge representation languages, which is *Ontology Definition Metamodel* (ODM) developed by Object Management Group (OMG) [ODM06]. ODM is trying to cover RDF, OWL, Topic Maps and UML languages. It defines meta-model for each language using EMOF[39] and it also defines mappings (transformations) between the individual languages. However, the focus of this work is primarily on meta-modeling and model transformations, i.e., it is aimed at supporting modeling (meta-modeling) tools and research rather than general application development.

36 A declarative porgramming language based on direct application of first order logic
37 Knowledge Interchange Format (KIF) – a declarative language for knowledge interchange
38 Common Logic (CL) – a standardized format for expressing statements in first order logic
39 EMOF - Essential MOF, MOF = Meta Object Facility – another OMG specification

4.2.4 Concept Analysis

Another technique potentially applicable to address certain goals of our work, notably the *level of abstraction gap*, is the *Formal Concept Analysis* [FCA] technique. The formal concept analysis is a technique to analyze source data in the form of a matrix (objects x attributes) and represent them in the form of *concept lattice* [LATORD]. The concept latice is such, that the top node of the lattice is the all encompassing *universal concept* which is a generalization of all objects in the source matrix. The individual concepts correspond to a set of objects with certain attributes and are organized in the hierarchy, so that objects sharing exactly the same set of attributes correspond to the same concept node, while objects having some additional attributes are sub-concepts of that node. The bottom node of the latice is such a concept, which contains objects which have all the attributes from the source matrix.

The concept lattice is used to discover natural object and property clusters. [CONAL] The lattice also represents a hierarchical partially ordered structure, which can be used to represent knowledge in terms of implications between the sub-concepts and their parent concepts. [LATEO]

An example application of the concept analysis can be class hierarchy analysis and optimization: Let the source data be a set of classes (corresponds to a set of *objects*) and set of all class members and methods corresponding the the set of *attributes*. The top node is a set of classes with no members or methods. The bottom node is a set of classes which have all the class members and call the methods present in the source data. The individual concepts corresponds to classes which have exactly the same class members and methods and the only difference between them is their name. The concept-subconcept hierarchy captured in the concept lattice indicates how (given the source data) the optimum class hierarchy looks like and if there are any redundancies in terms of having multiple classes with different names while having exactly the same features (class members and methods)

While the formal concept analysis looks very promising and it clearly was a strong inspiration for our own work. We hit the wall when trying to use concept lattices directly for delivery context representation, resource annotations and matching provisions to requirements: the problem is, that in concept analysis, the only we can only check whether a particular object has or has not an attribute, but not a particular attribute value. Trying to apply concept analysis on UAProf profiles, is difficult, because UAProf profile has many CC/PP attributes and and many of them are not boolean values but instead they are literals, integers or dimensions. To represent such data in a concept lattice would lead to explosive growth of number of attributes: for example, given the UAProf *Keyboard* attribute (see Figure 12 on page 38) we would need to introduce the following attributes:

1. Keyboard_Disambiguating
2. Keyboard_PhoneKeypad
3. Keyboard_Query
4. Keyboard_OnScreenQuerty
5. Keyboard_Rotator
6. Keyboard_NaviSpinner

in order to be able to annotate individual *objects* (devices or resources) with the respect to their keyboard capabilities. Even if consolidation, canonicalization and some sort of abstraction (e.g. representing UAProf *ScreenSize* as an enumeration) takes place before

actually building the concept lattice, still the resulting lattice would have hundreds or thousands of attributes (depending on the choses level of granularity). Such a data structure would be hard to understand and manage and in addition to that, the concept lattice would be highly subjective depending on the required level of granularity and grouping rules applied the raw individual attribute values when turning them into enumerations necessary for the concept analysis.

4.3 Design Conclusion

The results of the research briefly summarized in this section led us to the decision to address the goals stated in section 3 by designing a versioning framework from scratch, independent of the standards-based technology stack (Figure 3, page 20) and/or of a particular underlying technology for capturing the versioning metadata. This does not mean to re-invent everything from scratch, it only means that the versioning framework interfaces are technology neutral and has no dependencies on the technologies discussed in the sections above. On the other hand, the framework is designed to be compatible with both the standards technology stack (section 2.2) and also allows to incorporate the technologies discussed in section 4.2.

Due to the fact, that the primary expected application environment of the framework is an application implemented in a modern object oriented programming language, the framework itself has a form of an object-oriented API to allow for a straightforward implementation in an object-oriented language and a seamless integration to an object oriented application. Java programming language is used for the purpose of communicating the framework's technical design, as it has effectively become a *lingua franca* and an IDL language[40] of choice of the IT community. The framework is presented in the following chapter.

40 A sub-set of Java is commonly be used as an IDL (Interface Definition Language) instead of a single-purpose languages like OMG IDL.

4.3 Design Conclusion

5 The Versatile Framework

Background

The design decisions leading to the presented framework were driven by the author's former experience in the versioning domain ([JG99] and [JG03]) and the domain of multimodal systems ([ICSM2001], [IIWAS2001], [SEKE02], [PA20020198719], [PA20030046316], [PA20060036770] and [DMSP]).

Overall Roadmap

The overall scope of this work is presented in the following sections:

- 1.1 Application Domain (page 9)
- 1.2 Versioning Domain (page 10)
- 1.3 Usage Domain (page 11)

Important observations discovered in the course of the research activity are discussed in the following section:

- 2.4 Important Observations (page 32)

The thesis goals and high level design considerations are presented in the following sections:

- 3 Setting the Goals (page 41)
- 4.1 Design Considerations (page 43)
- 4.3 Design Conclusion (page 49)

This Chapter Structure

In this chapter, we first try to introduce the framework from a high-level perspective and pin-point its key design principles. The details of individual elements are discussed in subsequent sections. The entire chapter is supplemented by *Versatile 1.0 API Reference* [VERSAPI], a complete API reference manual available as a separate document. It is recommended to have the reference manual readily available while reading this chapter to look-up details for individual elements when needed.

5.1 The Elevator Pitch

The main idea behind the Versatile framework is describing device capabilities (*requirements*) and application artifacts (*provisions*) using semantically rich properties – mostly *hierarchical classifications* (taxonomies) – and employing the semantical information captured in the properties for implementing a best-effort (approximate) requirements/provisions matching algorithm. Thanks to the application of hierarchical classifications, the best-effort algorithm can incrementally generalize the requirements while searching for the artifacts most closely corresponding to device capabilities. This ability of *constraint relaxing* via *generalization*, allows for extremely efficient metadata annotation of application artifacts: using generic property values for shared resources

while using more specialized property values for resources intended for specific device clusters or even individual devices.

In addition to the above, the framework provides services for flexible definition of *priorities* and *resolution rules* for property value acquisition from multiple sources and services for property *transformations* including *canonicalization*, *information extraction* and *information synthesis*. These services are aimed at separating the versioning code from the application code and encapsulating it by a set of well defined interfaces in order to enforce proper code structure resulting in maintainable and evolution-friendly code base.

5.2 Conceptual Overview

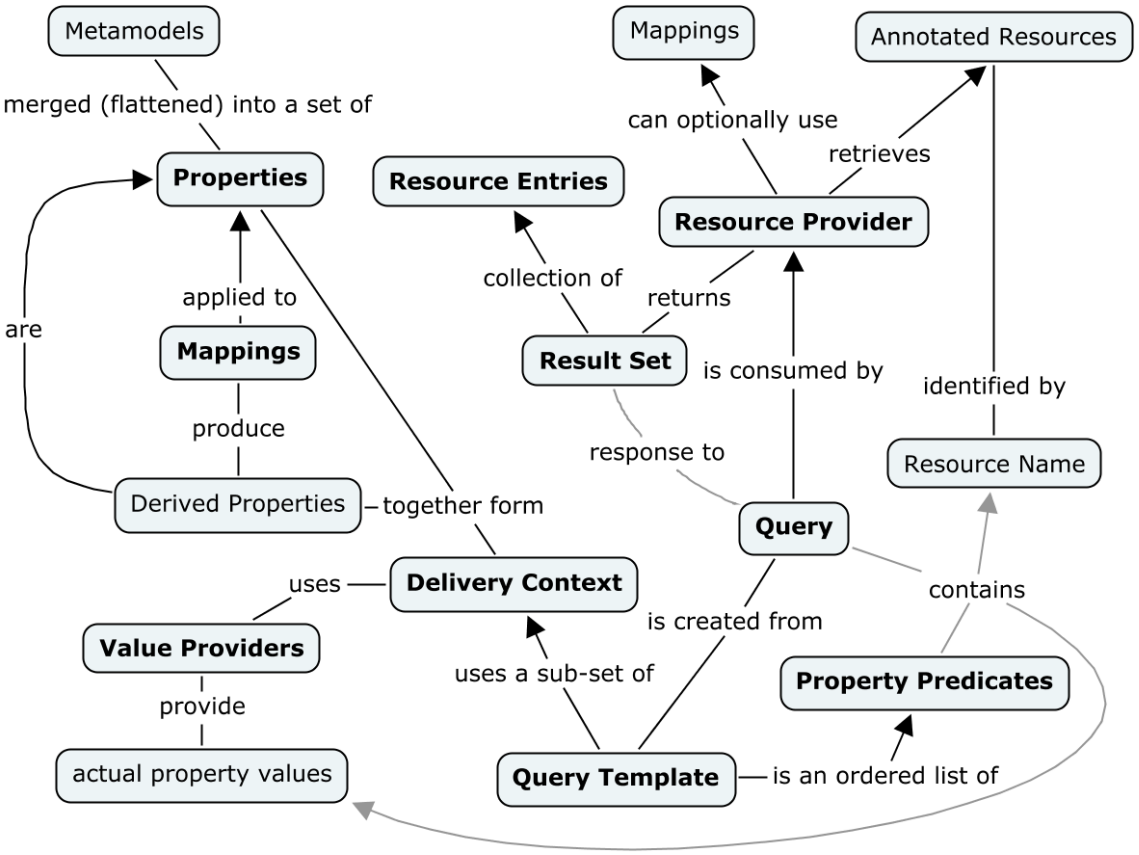


Figure 15: Versatile – The Key Concepts (a concept map)

The above concept map[41] captures the key elements of the Versatile framework. Let us walk over the concept map and briefly introduce each element to give the reader a high-level understanding of the framework, the details follow throughout the rest of this chapter.

Properties represent metadata definitions. Each Versatile property has a *unique identification* and a *data type*. This essential information can be further extended by

41 Concept maps is a formalism used for *cognitive learning* developed by *Joseph D. Novak* [CMAP06], [CMAP04]and supported by a tool provided by the *Institute for Human and Machine Cognition* (IHMC). The author of this thesis has been using them throughout different work stages (complemented by UML tooling) for various purposes, including but not limited to recording related domain knowledge, early system design drafts and charts. More concept maps developed in the course of this work are available on http://dsrg.mff.cuni.cz/~gergic/versatile/.

using one of the semantically richer property sub-types. The set of property sub-types is extensible, out-of-the-box we provide the data structures identified in section 4.1 (Design Considerations, page 43): *controlled vocabulary* (an enumeration), *relational property* (to represent arbitrary binary relations), *order property* and last but not least the *taxonomy.* The chosen set of property types allows for a straightforward mapping of information from OWL ontologies or UML meta-models to Versatile: if the source of metadata information is an ontology or a meta-model (representing a richer structure), the selected facts, only those needed for the purpose of versioning task, are projected to Versatile as properties. The flattening of the structurally richer representations into a set of properties has the advantage of *comprehensibility* and *interoperability:* the chosen representation of metadata serves as a common denominator across the potential metadata sources, while keeping the necessary expressive power. The designed typology of properties is provided such that it addresses the *Level of abstraction* goal as per section 3 Setting the Goals (page 41).

Delivery Context serves as a property registry and it is in some sense the central entity of the framework: a property, to become available in Versatile, needs to be registered to the delivery context alongside its *value provider* or a *property mapping*. All used properties must be registered in the delivery context and it should be the only source of versioning relevant meta data and configuration settings. Due to its exclusive role in the framework, it can be used to track dependencies of the application on the external metadata sources. Delivery context is provided in order to address the *Best Practices Enforcement* as per section 3 Setting the Goals (page 41).

Value Providers are used for *property value* acquisition at runtime. Value providers are specific to the underlying metadata source, for example an HTTP request, HTTP session, user profile, cookie, CC/PP engine or a configuration file. Value providers usually form chains, thus effectively defining *resolution policies*: value providers in the chain are visited one by one until the property value is determined. Each property can have a its own uniquely configured value provider chain thus allowing to define property-specific rules. The main role of value providers is to address *Metadata Consolidation* as per section 3 Setting the Goals (page 41).

Property Mappings are used to calculate values of derived properties[42] via transformations from other properties registered in the delivery context. Mappings are used to implement canonicalization or other necessary metadata enrichment, for example to map values from an RDF Literal property (an unconstrained string) to a well-defined application-specific taxonomy. By using term derived we mean only the property value is derived (inferred by calculation); not necessarily its data type. The concept of separating the value acquisition (value providers) and transformations (property mappings) is very important for transparency and reusability – separation of concerns. Property mappings are provided so that they contribute to achieving the *Level of abstraction, Metadata Canonicalization* and *Best Practices Enforcement* goals as per section 3 Setting the Goals (page 41).

Query Templates are used to express reusable metadata constraints and preferences using an ordered list of *property predicates*. When specifying a property predicate using a query template, we uniquely identify the property and specify a property operator (a relational or a functional operator) to be applied to a property value. Each query template is associated with a particular delivery context and it validates that the

42 The properties whose values are provided directly by value providers are called *leaf* properties.

properties being referred to are registered in the context and that the property operators are compatible with the property type. To actually search for resources, we create a Q**uery** based on a particular query template by specifying a *resource name*. In the course of query initialization, the query template retrieves the property values from its associated delivery context by invoking the corresponding value providers and property mappings and substitutes them to the property predicates. The resulting query object contains all the information needed for evaluation – it has no external dependencies as all the property values are already fixed. Query Template and Query concepts contribute to addressing the *Level of abstraction* and *Best Practices Enforcement* goals as per section 3 Setting the Goals (page 41).

Resource Provider consumes a query, searches its underlying repository of metadata annotated resources and returns the resource (or – depending on query settings – a list of resources) which most closely corresponds to the metadata constraints expressed in the query. Resource providers are purpose and data-store specific: they can serve as class factories for application objects, resource bundles for static resources like labels, messages or graphics and also as content transformers/transcoders used to dynamically transform application resources according to the specifications provided by the Query. The framework specification does not assume any particular implementation or a data store type for the resources repositories, the only requirement is that all resource provider implementations must fully implement the query semantics as described in this paper. Resource Provider concept is introduced in order to address the *Best Practices Enforcement* as per section 3 Setting the Goals (page 41).

Result Set is an ordered collection of **Resource Entries**. The primary order of the collection is determined by the result *score* of individual resource entries in the collection. **Score** measures how closely a particular resource entry matches the *query*: Due to the framework's capability to execute approximate matching in addition to standard exact matching, we need to measure *quality* of individual results with the respect to the original query. Higher the score, higher the entry in the result set; the exact matches come before any approximate matches. Besides the score, the resource entry also caries all *the metadata annotations* of the underlying resource it represents and most-importantly also a reference to the actual resource. There are two main reasons the framework returns the results in the form of collection of resource entries: (1) let the user examine the results with regards to the actual metadata; (2) performance consideration: having a resource represented by the resource entry proxy object, the actual resource can be retrieved or instantiated only upon accessing the underlying value of the resource entry, this is especially significant in case of using larger collections of result entries.

5.3 Technical Overview

Before we actually start describing the Versatile Framework, we present one more motivational example to give the reader an idea how it feels to think and work in the Versatile mindset. The task is as follows: Developing a multi-modal application for a large number of device categories, there is a need to use different *user interface layout* depending the nature of the device and its hardware and software capabilities.

Certainly, we need to take into account the *size* and *shape* of the screen and consider a possibility that the device has *no screen at all* – a speech application may have no visual user interface which does not prevent it from having a *logical* user interface

"layout". Second, we need to consider the markup rendering capabilities of the device: some devices, even though they have a large screen they may not be capable of rendering rich user interface layout due to their browser limitations.

For the screen *size* and *shape*, let us re-use the taxonomies already presented earlier: *ScreenSize* (Figure 10, page 36) and *ScreenOrientation* (Figure 14, page 46). For the rendering capabilities let us use a very simple *DeviceMarkupClass* demonstrative taxonomy (Figure 16 below).

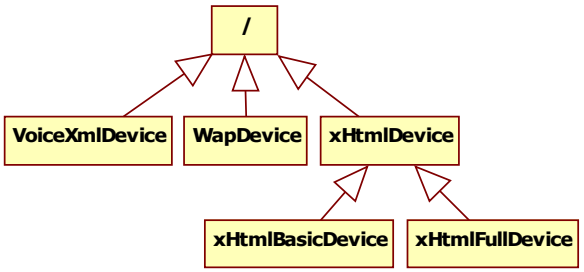


Figure 16: DeviceMarkupClass sample taxonomy

For now, let us assume we have all the three properties setup in our application (registered alongside their value providers or property mappings) so that their values can be evaluated at any time and let us focus on the piece of code which actually deals with instantiating a context-specific *layout manager* (Example 7, page 56) :

1. We instantiate a custom implementation of the Resource Provider interface, which represents a resource repository, or in this particular case a class factory.
2. We setup a Query Template: during initialization, the template is bound a particular instance of the delivery context, then three constraints are added for three different properties using the *bestMatch* operator. The *bestMatch* operator can be applied to *taxonomies* and expresses a soft contraint (a preference), when adding the constraints to the query template, the query template validates the properties used in constraints whether they are registered in the associated delivery context and whether the operator of the constraint is applicable to a given property type.
3. The *newQuery* method is invoked for a particular resource (LayoutManager), during the execution of this method, the metadata constraints are copied from the query template to a newly created query and the delivery context is asked to provide the current values for the properties used in the constraints. The result of the operation is an immutable query object, which does not depend on the delivery context anymore as all its valued are already fixed: for example:
   - DEVICE_MARKUP_CLASS = "xHtmlBasicDevice"
   - SCREEN_SIZE = "QVGA"
   - SCREEN_ORIENTATION = "Portrait"
4. The resource provider is asked to evaluate the query and return a collection of results. The default result collection size is 1 so in this particular case (given the type of properties and the operator in use), the resource provider proceeds as follows: it first tries to search for the *exact match,* i.e., a resource named "LayoutManager" with metadata attributes DEVICE_MARKUP_CLASS, SCREEN_SIZE and SCREEN_ORIENTATION equal to the above. If no such

resource variant exists, it progressively relaxes the constraint via generalization relation of the underlying taxonomies, until it finds the closest resource variant. If there is no resource variant for the "LayoutManager" resource name at all, the resource provider fires MissingResourceException Java exception.[43]

5. Please notice that the example includes two variants of the query execution part: the first one reveals a bit more about the concepts of the framework, while the second variant demonstrates the shortest possible syntax suitable for the common usage (in this case equivalent to the first extended syntax).

```
public LayoutManager getLayoutManager(DeliveryContext deliveryContext)
    throws  UnsupportedPropertyOperatorException,
            UnregisteredPropertyException {
    LayoutManager lm = null;
    // use a custom implementation of the ResourceProvider interface
    ResourceProvider uiFactory = new UIClassFactory();
    //define a reusable meta-data query template linked to a context
    QueryTemplate qt = new QueryTemplateImpl(deliveryContext);
    //add predicates in descending order of significance
    qt.addBestMatch(DEVICE_MARKUP_CLASS);
    qt.addBestMatch(SCREEN_SIZE);
    qt.addBestMatch(SCREEN_ORIENTATION);

    // VARIANT 1 - the expanded syntax
    //obtain the query object for a particular resource
    //(also retrieves all property values from delivery context)
    Query query = qt.newQuery("LayoutManager");
    // pass the query to the resource provider
    ResultSet rSet = uiFactory.get(query);
    // return the first entry of the N-best result set
    lm = (LayoutManager)rSet.getValue();

    // VARIANT 2 - shorthand notation equivalent to the above
    lm = (LayoutManager)uiFactory.getValue("LayoutManager", qt);
    return lm;
}
```

Example 7: Instantiating LayoutManager - an end-to-end example

5.4 Versatile Properties

Properties represent metadata definitions. Each Versatile property has a *unique identification* and a *data type*. This essential information can be further extended by using one of the semantically richer property sub-types. The set of property sub-types is extensible, out-of-the-box we provide the data structures identified in section 4.1 (page 43): *controlled vocabulary* (an enumeration), *relational property* (to represent arbitrary binary relations), *order property* and last but not least the *taxonomy*.

Properties are used in the Versatile framework to represent metadata describing the actual artifacts. The focus of the framework is on the use of semantically rich property

43 The exact semantics of query processing is described in section 5.7.3 Query Semantics.

definitions of metadata attributes so that the semantical information captured in the property definition can facilitate metadata driven search capabilities of the framework. The framework comes with a set of predefined property types shown on Figure 17 on page 57. This type hierarchy can be extended as needed: in extreme case, each and every particular property can declare its own type by extending one of the predefined types, however, it is expected that generic implementations of the pre-defined property types will be used in most applications.

Each **Property** (`cz.cuni.versatile.api.Property`) has a *unique name* which is composed of a *namespace,* a *separator* and a *local name*. Versatile does not impose a particular naming convention, the structure of the unique name is designed for compatibility with the frequently used naming schemes like XML namespaces, Java package names or C++ and CORBA IDL namespaces. Each property also exposes information about its *data type* which corresponds to a type defined in terms of the underlying programming language. Unique identification and data type information represent the minimum set of requirements each property must satisfy.

Controlled Vocabulary (`cz.cuni.versatile.api.ControlledVocabulary`) is an extension of the base property type which adds the possibility to enumerate all allowed property values, thus further restricting the data type of the property.

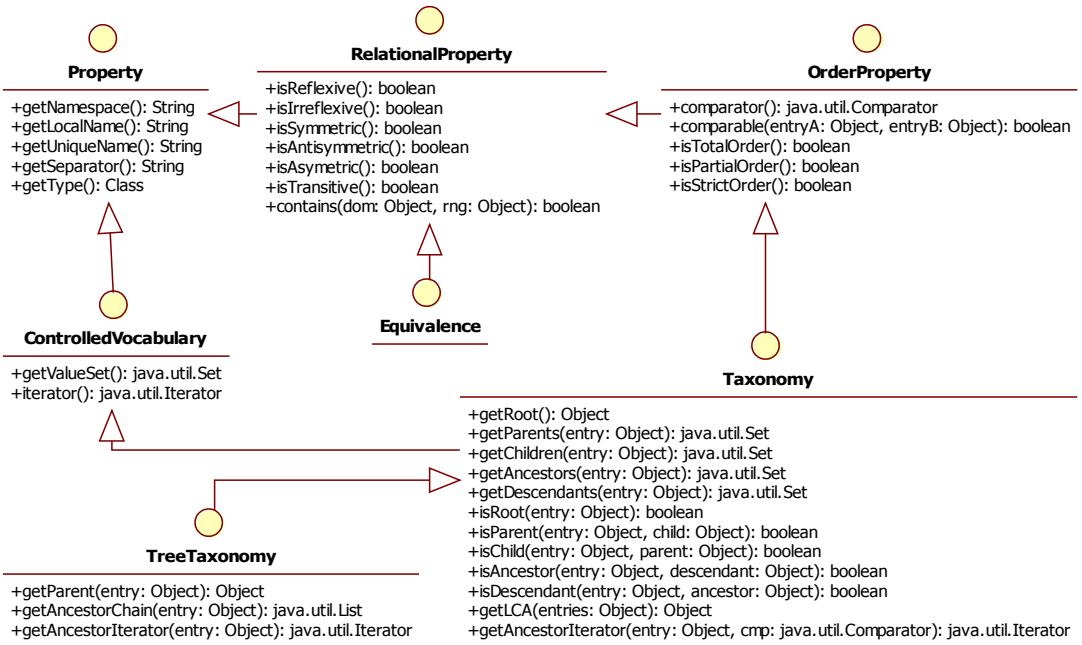


Figure 17: Versatile built-in property type hierarchy

Relational Property (`cz.cuni.versatile.api.RelationalProperty`) allows to represent a binary relation over the set of property values. Given a pair of property values, it can determine whether they belong to the binary relation or not, i.e., whether the two values are *related* in terms of the relation or not. Besides this, there is a set of methods which allow to query the usual algebraic properties of the relation: *reflexivity*, *symmetry* and *transitivity*. The relational property type can be employed for capturing *relations* between elements of ontologies and meta-models when projecting them

(flattening) to the data model of the Versatile framework, of course, it can be used for representing binary relations over a set in general.

Equivalence (`cz.cuni.versatile.api.Equivalence`) is a specialization (restriction) of the relational property following the usual definition of *equivalence relation* in algebra: a relation which is *reflexive*, *symmetric* and *transitive*. The purpose of this property type is to be able to factorize the value set of a particular underlying data type into equivalence classes as needed in a particular context: some property values may be considered equivalent in one context while may need to be distinguished in another context. In Versatile, the developer can construct as many logical "views" (equivalence properties) as needed to represent different perspectives on the same source value set.

Order Property (`cz.cuni.versatile.api.OrderProperty`) is another specialization of the relational property used to define both *partial* and *total* order relations. In algebraic terms, the relation must be *antisymmetric* and *transitive*, in case of strict ordering (e.g. < or >) it must be also *irreflexive* (commonly used ≤ and ≥ are *reflexive*). Total order relation requires all values to be comparable which may not be the case for the partial order. In Versatile, when using the term *order,* we always mean *partial order*, *total order* is always referred to explicitly. Order properties have many applications, for example when we want to express constraints like: *the Java midlet requires MIDP 2.0 or newer*; we may go ahead and define *JavaPlatformSuccessor* order property, backed by the *JavaPlatform* value set as depicted on Figure 9, page 35, by explicitly stating which value pairs are considered to belong to the *successor* relation. By the way: given the statement "*x" or newer* we need to make sure the relation is *reflexive* so that the pair ("MIDP 2.0", "MIDP 2.0") is also a member of the relation.

Taxonomy (`cz.cuni.versatile.api.Taxonomy`) and its specialization, the tree taxonomy (`cz.cuni.versatile.api.TreeTaxonomy`), are crucial elements of the Versatile data model. They both enable construction of properties with hierarchical classifications of their value sets. The only difference between the two is, that the tree taxonomy does not allow a node to have mode than one parent. The ability to taxonomize a value set of a property into a hierarchical classification is a pre-requisite for enabling *constraint relaxing* via *generalization* applied when matching device capabilities (*requirements*) to *provisions* of the individual software artifacts. The idea is to express requirements as concrete as possible while annotate software artifacts with taxonomy values as generic as possible, then, when searching for a resource suitable for a particular device, the requirements can be incrementally generalized until an artifact closest to the requirements is found.

It is important to note that `Taxonomy` is a subtype of `OrderProperty`: the hierarchical classification encoded by the taxonomy imposes a *partial order* on the property values. The root of the taxonomy is the *all-encompassing universal concept*, all its sub-concepts (children) are then "<" (less-than) as compared to their parent; this applies recursively throughout the taxonomy.

The careful reader certainly already noticed, that most of the property examples in this paper are depicted as taxonomies, this only demonstrates the importance of the property type to the author. More examples follow, complemented by a detailed explanation of the taxonomy-based requirement/provision matching algorithm.

Relational Operators

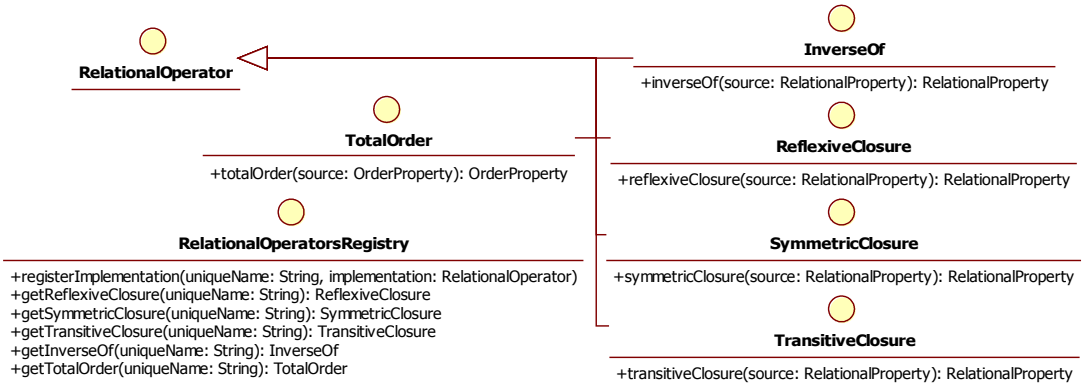


Figure 18: Relational Operators Library

The Versatile framework further facilitates manipulation with relational properties and creation of new properties based on the existing ones by providing a library of basic **relational operators** (Figure 18 above) located in package `cz.cuni.versatile.api.relops`. The library can be used to create new properties in a declarative way by applying the common algebraic operators like *Inverse-Of*, *Reflexive-Closure*, *Symmetric-Closure*, *Transitive-Closure* and *Total-Order*. The framework allows to register custom implementations of these algebraic operators with the factory class `RelationalOperatorsRegistry` on per-property basis – using the property's *unique name*. If a custom implementation is registered for a given property, it overrides the generic implementation provided by the framework. For further details please refer to the API reference manual.

5.5 Delivery Context and Value Provider

Delivery Context

Delivery context (`cz.cuni.versatile.api.DeliveryContext`) serves as a property registry and it is in some sense the central entity of the framework: a property, to become available in Versatile, needs to be registered to the delivery context alongside its *value provider* or a *property mapping*. All used properties must be registered in the delivery context and it should be the only source of versioning relevant meta data and configuration settings. Due to its exclusive role in the framework, it can be used to track dependencies of the application on the external metadata sources.

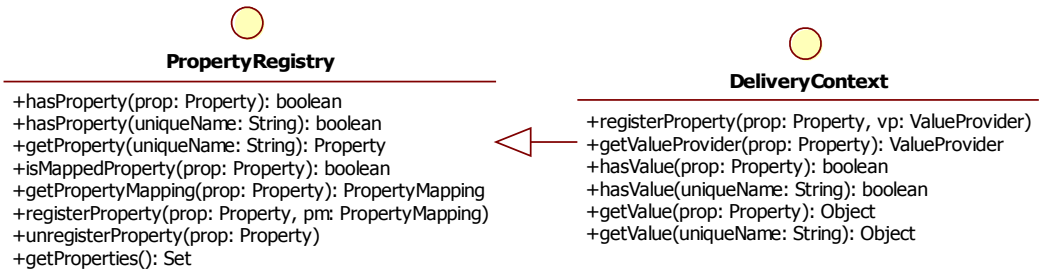


Figure 19: Delivery Context extending Property Registry

There can be more than one delivery context instance in the application, reflecting the logical structure of the application, in such a case, these instances are independent and logically correspond to multiple instances of the Versatile framework. If a property is used in multiple delivery contexts, it needs to be registered to each delivery context separately.

The main purpose of using *derived* (mapped) properties in the `DeliveryContext` is to transform raw (typically domain-specific -- e.g. UAProf) meta-data into pre-processed application-specific properties which better correspond to the inherent logic of the application. The transformations in such a case are typically value adding (information adding): e.g. canonicalization, hierarchical classification. Alternatively, the usage of the property mappings can be as simple as property renaming (aliasing) due to the need to use multiple overlapping vocabularies (namespaces).

Value Provider

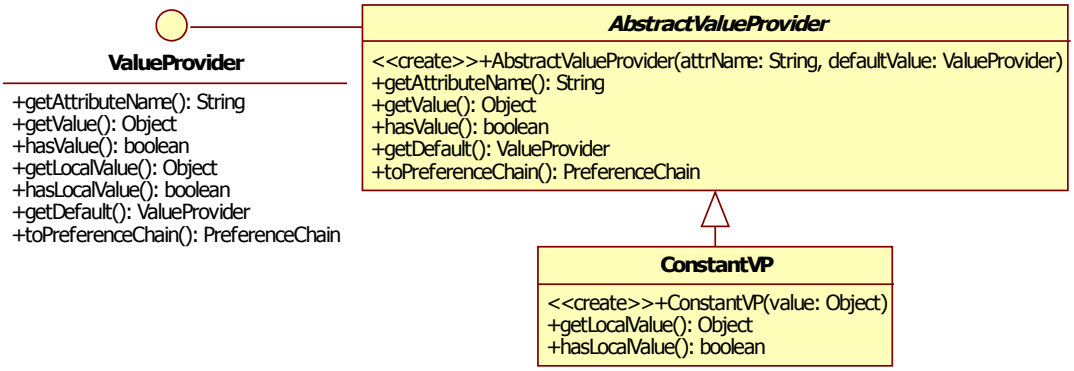


Figure 20: Value Provider, its abstract and concrete implementations

Property values can be obtained in two different ways: via *property mappings* or *value providers*. Property mappings are discussed in a separate section, so let us focus on value providers first. The role of the value provider concept (`cz.cuni.versatile.api.ValueProvider`) is to represent an abstract attribute value getter, which can be chained in order to implement a particular *fall-back strategy* or a *resolution policy:* when a value provider chain is requested to retrieve a property value, value providers in the chain are visited one by one until the property value is determined. Value provider implementations are specific to the underlying metadata source, for example an HTTP request, HTTP session, user profile, cookie, CC/PP or a configuration file. Each property can have a its own value provider chain thus allowing to define property-specific rules.

Example 8 on page 61 demonstrates construction of such a property-specific value provider chain and registering it with a property to the delivery context. Please note, it is not an ad-hoc example: in fact, it is an implementation of the resolution rules described in section 2.4.1 (Metadata Consolidation) in the Versatile framework applied to the *locale* taxonomy introduced in 2.1.2 (CATCH 2004), Figure 1. The values of the *locale* taxonomy are instances of standard java.util.Locale class, however, the taxonomy wrapper is needed to represent the taxonomy semantics in the Versatile framework. Delivery context is provided by the framework. The individual value provider implementations represent custom environment specific extensions of the

`AbstractValueProvider` provided by the framework (even though in this particular case they are generic enough to consider them to make them a part of the framework in the future).

```
public void registerLocale(DeliveryContext ctx, Taxonomy locale) {
    ctx.registerProperty(locale,
            new UserProfileVP("userLocale",
                new SessionCookieVP("app-Language",
                    new UAProfVP("SoftwarePlatform.CcppAccept-Language",
                        new HTTPRequestVP("Accept-Language",
                            new ConstantVP(Locale.ENGLISH))))));
}
```

Example 8: User locale value provider chain example (resolution rules/fall-back)

When describing value providers, it is important to briefly mention two special data structures supported by the framework, which can be used to represent property values: *preference chain* (`cz.cuni.versatile.api.PreferenceChain`) denotes an ordered list of values and *preference bag* (`cz.cuni.versatile.api.PreferenceBag`) denotes an unordered set of values. For those familiar with RDF: these structures correspond to RDF:Sequence and RDF:Bag respectively. In general, a `ValueProvider` for any `Property`, regardless of its data type, can return `PreferenceChain` or `PreferenceBag`, i.e., a collection of multiple values, instead of a single value of the property data type. Given the Example 8 above, the `ctx.getValue(locale)` may return `PreferenceChain` of locales corresponding to the prioritized sequence of user preferences, for example, taken from the user's web browser as shown on Figure 7 on page 33.

5.6 Property Mappings

Property mappings are used to calculate values of derived properties via transformations from other properties already registered in the delivery context. Mappings are used to implement canonicalization or other necessary metadata enrichment, for example to map values from an RDF Literal property (an unconstrained string) to a well-defined application-specific taxonomy. By using term derived we mean only the property value is derived (inferred by calculation); not necessarily its data type.

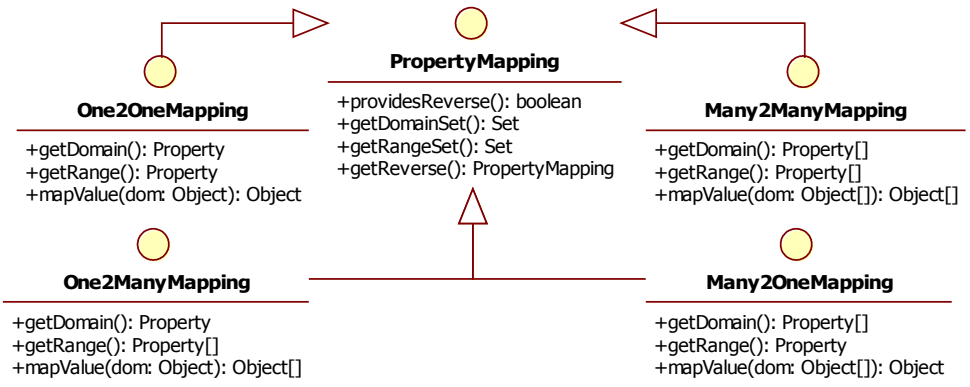


Figure 21: Transformational Property Mappings

Property mappings implement transformational maps between properties. They allow to separate property value acquisition (implemented by value providers) and subsequent transformations (like canonicalization, semantical enrichment or data extraction). This separation of concerns is very important for transparency and reusability.

Referring to the discussion in section 4.3 Design Conclusion, property mappings are exactly the placeholder in the Versatile Framework to plug in an external inference (rule) engine if applicable to a particular property mapping. In general, it is assumed, that the property mappings are implemented directly in Java, in cases a declarative engine is being employed, it needs to be embeddable into Java. In any case, the property mappings are completely opaque to the framework user who can only see the mapping signature: set of domain and range properties.

The interface `PropertyMapping` in package `cz.cuni.versatile.api.relops` represents a generic a transformational map from a set of properties to another set of another properties. Mathematically speaking, it is an n-ary function `(P1, P2, P3, ..., Pn) -> (P1', P2', P3', ...Pm)`, `n` and `m` being positive integers, so that given input values (`x1:P1, ..., xn:Pn)` it calculates a tuple of output values (`y1:P1', ..., ym:Pm).` Actually, the `PropertyMapping` interface serves only as a marker interface and for meta-modeling capabilities like property dependency tracking. The actual transformations need to be based on one of its sub-types. The reason behind the chosen type hierarchy organization is to make implementation of the most common mappings (e.g. the unary function) easier.

One2OneMapping (`cz.cuni.versatile.api.relops.One2OneMapping`) represents *unary function* `(P1) -> (P2)` transforming values of one property to values of another property. It can be used for *canonicalization* and/or to generate taxonomies (hierarchical classifications) out of the raw unchecked meta data values (*semantical enrichment*). It is expected to be the most common property mapping. The Versatile package provides a generic implementation of the most essential `One2OneMapping`: the `IdentityMapping` (identity function), usable for property renaming/aliasing.

One2ManyMapping (`cz.cuni.versatile.api.relops.One2ManyMapping`) represents *information extraction* mapping `Px -> (P1, ..., Pn).` There are quite a few instances of existing metadata properties which contain *composite literal* values or combine multiple semantical entities into a single named property and in turn require further parsing to extract the individual semantical entities – the actual metadata properties. For example:

- HTTP header `User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4`
- UAProf attribute `SoftwarePlaform/JavaPlatform` can contain both `Configuration/CLDC-1.0` versus `Profile/MIDP-2.0` thus combining *Profile* and *Configuration* under one UAProf attribute (see also Figures 8 and 9)

From the perspective of semantics, `One2ManyMapping` can be replaced by a set of unary mappings. The motivation for introducing this type of mapping is to be able to acquire and parse the source property value only once and then extract all the output data values in one pass. This can significantly improve performance when multiple properties derived from a single source property are used in the same query template, which means they need to be evaluated at the same time.

Many2OneMapping (`cz.cuni.versatile.api.relops.Many2OneMapping`) corresponds to an ordinary *n-ary function* `(P1, P2, P3, ..., Pn) -> Pm`. It is used if there is a need to combine several simpler properties into a single derived property and then use this sythesized property in the versioning code. In Example 9 and taxonomy on Figure 16, page 65, we introduced *DeviceMarkupClass* taxonomy which for a given device pin-points its most natural (preferred) markup class (a very high-level classification indeed). In order to compute values of the hypothetical *DeviceMarkupClass* property, we would need to take into account the following source UAProf properties: `BrowserUA` (`HtmlVersion`, `XhtmlVersion`, `BrowserVersion`, `BrowserName`) and `WapCharacteristics` (`WapVersion`, `WmlVersion`, `WmlScriptVersion`).

Many2ManyMapping (`cz.cuni.versatile.api.relops.Many2ManyMapping`) represents the most generic property mapping as it implements the general `n:m` arity map described above, i.e., a complex transformation between two sets of properties. However, its usage should be considered carefully, and justify its usage, because if overused, the application designer may end-up with few general purpose complex mappings, which goes against the philosophy of the framework: modularity and reusability of small and comprehensible code units.

5.7 Query and Query Template

Query (`cz.cuni.versatile.api.Query`) interface is a data structure representing a multi-variant resource query. It encapsulates all the information necessary to retrieve a *resource* which possibly exists in many different variants, revisions and flavors (resource means a versioned entity – a set of artifacts – not a particular version/variant of a resource – an individual artifact):

- *resource name* (mandatory) which uniquely identifies a *resource* in the scope of a particular `ResourceProvider` (5.7.5) instance. Please note, that unlike property names, resource names are not globally unique in Versatile.
- *ordered list of property predicates* (optional, empty by default) which specifies the metadata constraints and/or preferences (5.7.1)
- *n-best size* (optional, default = 1) which is the maximum number of results to return
  (result set capping)
- *score threshold* (optional, default = 0.0) which determines the lowest acceptable score value[44], all values are accepted by default
- *scoring factor* (optional, default = 0.99) which determines the relative significance of individual property predicates when calculating the score of individual result entries

44 Score measures how closely a particular resource entry matches the query, this only applies to queries which use constraint-relaxing operators like *bestMatch* which may return approximate matches.

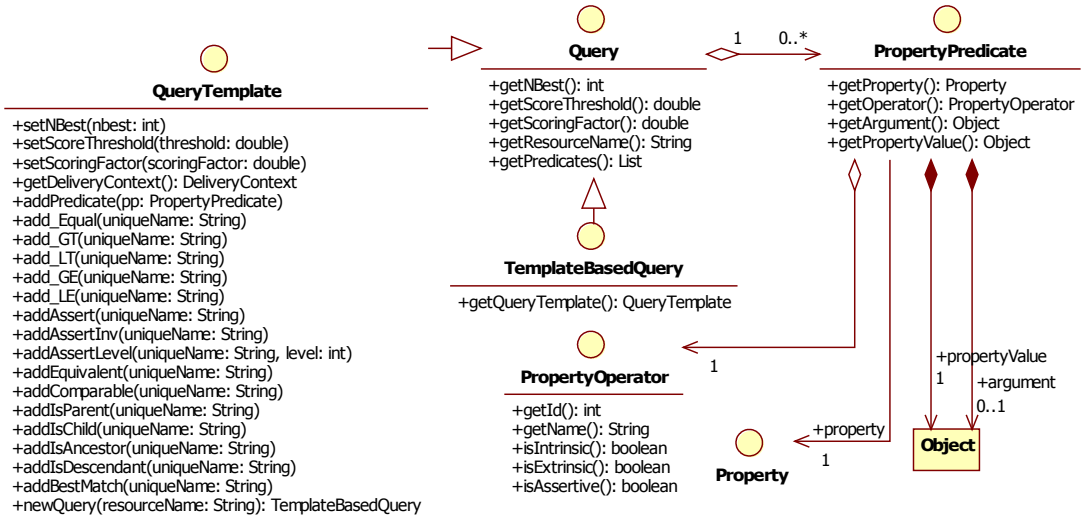


Figure 22: Query, QueryTemplate and Property Predicate

Semantics of the individual query attributes is explained in the following sections, before we get there, let us make couple of remarks: In a typical setup, `Query` instances are not created directly by the developer they are rather instantiated as a spin-off of a reusable `QueryTemplate` object. `Query` and `ResultSet` (5.7.2) data structures can be seen as the messages of the communication protocol between the framework and `ResourceProvider` implementations. There are also the performance considerations reflected in the framework design: `Query` is meant to be an immutable object: once it gets instantiated, it never changes, `Query` implementations should override `Object.equals()` and implement an efficient comparison algorithm to detect whether two queries are identical, this enables `ResourceProvider` implementations to improve performance by result *caching*.

Query Template (`cz.cuni.versatile.api.QueryTemplate`) represents the actual Versatile application programming interface (API) used by the application developer to express the metadata constraints. Once the framework is configured by setting up the delivery context and relational operators registry, the actual usage of the framework means defining a set of query templates for individual resource classes and categories (static resource bundles, various class or component factories, data retrieval) and then triggering queries for individual resources and passing them to the corresponding resource provider. The instances of the `QueryTemplate` class serve the following purposes:

1. define a set of metadata constraints and preferences (an ordered list of predicates)
2. define the query evaluation preferences (N-best, score threshold, scoring factor)
3. allow to re-use the same settings for many different queries (with different resource name)
4. allow to fully automate property value acquisition and substitution: during query instantiation (`newQuery` method), the property values for each predicate are automatically retrieved from the delivery context the query template is linked to.

Because the sets of property types and property operators are extensible in Versatile, the `QueryTemplate` interface provides a generic method for adding property predicates (`addPredicate`) and a set of methods allowing a short-hand notation for all the built-in operators.

```
public void stringResourcesSample(DeliveryContext ctx)
    throws UnregisteredPropertyException,
           UnsupportedPropertyOperatorException {
    // use a custom implementation of the ResourceProvider interface
    ResourceProvider strings = new StringResourceBundle();
    //define a reusable meta-data query template linked to the context
    QueryTemplate qt = new QueryTemplateImpl(ctx);
    //setup scoring factor (bias) for this query template
    qt.setScoringFactor(Query.BIASED_SCORING_FACTOR);
    //add predicates in descending order of significance
    qt.addBestMatch(LOCALE);
    qt.addBestMatch(DEVICE_MARKUP_CLASS);
    qt.addBestMatch(SCREEN_SIZE);

    //use the query template to retrieve localized strings
    print(strings.getValue("TitlePageLabel", qt));
    print(strings.getValue("WelcomeMsg", qt));
    for(int i=0; i<10; i++) {
        print(strings.getValue("MenuItemLabel" + i, qt));
    }
}
```

Example 9: Query Template reuse, string resources, biased scoring

Example 9 above demonstrates a typical usage of query template in the application. It first sets-up a set of metadata preferences applicable for retrieving static strings from a resource bundle. The example is taken from a multi-modal application and therefore the application labels and messages not only depend on the user *locale* (language, country) but also on the *device class* (speech versus GUI) and last but not least *screen size* – to use shorter messages on small-screen devices. The template uses *biased scoring factor* to avoid mixing different languages in the user interface. (The scoring factor parameter is explained in section 5.7.4.)

5.7.1 Property Predicate and Property Operator

Property Predicate (`cz.cuni.versatile.api.PropertyPredicate`, Figure 22, page 64) is a data structure representing a single metadata *constraint* or *preference*[45]. It holds a reference to the property definition (`Property`), the operator (`PropertyOperator`) to apply to the property value and optionally some additional operator-specific arguments encapsulated in a single data structure represented by the `argument` attribute (`java.lang.Object`). While the property predicate is a part of a query template, the `propertyValue` (`java.lang.Object`) is not set; it is set only upon query creation, when the predicated is cloned as a part of newly created `Query` instance and the current property value is acquired from the delivery context. (The cardinality depicted on the UML diagram on Figure 22 only applies to queries, not to query templates.)

45 Distinction between constraint and preference depends on the semantics of a particular property operator (assertive operators are used for constraints, contraint-relaxing for preferences).

Property Operator (`cz.cuni.versatile.api.PropertyOperator`, also Figure 22) represents a *relational* or *functional* operator to evaluate by the resource provider when matching the property value against the artifacts' metadata annotations in the resource repository. Besides machine processable *unique identifier* and human readable *operator name* it contains two meta-attributes which help the resource provider to understand how to process a particular operator:

1. *intrinsic*/*extrinsic* flag
2. *assertive* flag

Intrinsic operators use methods of the actual property values[46] for comparison, so they rely on the existing methods of Java objects like `Object.equals()` or `Comparable.compareTo()`. *Extrinsic* operators rely on the relations externally provided by the (semantically richer) Versatile properties, for example `Equivalence` or `OrderProperty` to relate to the example given for the intrinsic operators. The *assertive* flag determines whether the operator is *assertive* (the predicate must evaluate to `true` in order to include the resource in the result set) or whether the operator allows for approximate matching (*fall-back*, *constraint relaxing*) which is significant during query evaluation as described in section 5.7.3 Query Semantics.

The Versatile framework comes pre-loaded with a set of built-in operators which act upon the pre-defined property types described in section 5.4 Versatile Properties. The built-in operators are defined in the class `cz.cuni.versatile.core.PropertyOperators` implementing the `Property-Operator` interface. Table 2 below lists all the built-in operators alongside their meta-attributes. For details on individual operators, please refer to *Versatile 1.0 API Reference* [VERSAPI], we discuss in detail only the most significant operators and their semantics throughout this section.

Operator	Intrinsic	Assertive	Applicable To
=	Yes	Yes	Property
>	Yes	Yes	Property
<	Yes	Yes	Property
>=	Yes	Yes	Property
<=	Yes	Yes	Property
assert	No	Yes	RelationalProperty
assertInv	No	Yes	RelationalProperty
assertLevel	No	Yes	Taxonomy
equivalent	No	Yes	Equivalence
comparable	No	Yes	OrderProperty
isParent	No	Yes	Taxonomy
isChild	No	Yes	Taxonomy
isAncestor	No	Yes	Taxonomy
isDescendant	No	Yes	Taxonomy
bestMatch	No	No	Taxonomy

Table 2: Built-in property operators

46 Property values are all instances of java.lang.Object

First, before we start describing the individual property operators, let us explain how property predicate evaluation actually works: when a query is created out of a query template, the query template acquires the property values for all properties used in the query template using the delivery context and assigned the values into the property predicates. When the query reaches the resource provider, each property predicate has at least the following 3 attributes:

1. property ***P***
2. property operator `po`
3. property value *PV*

When matching the property predicate against its resource repository, the resource provider compares the actual property value *(PV)* and a candidate resource property value (*RV*) using the property operator `po`. Of course, the order of *PV* and *RV* is significant for many operators, the convention used by the Versatile framework corresponds to the following infix notation:

$$RV \ \mathtt{po} \ PV$$

For example: *PV*:6, po:<=, *RV*:4 is interpreted as 4 <= 6 which evaluates to `true`.

The set of intrinsic operators (=, >, <, >=, <=) is provided mainly for compatibility with ordinary *semantically loose* properties, which correspond to the base *Property* type, such a property can be created by taking any data type and assigning it a unique identifier – without any extra work. Understandably, this kind of properties is not the main focus of this work.

The operators `assert` and `assertInv` rely on `RelationalProperty.contains(x, y)` to assert whether a property predicate holds (evaluates to `true`) or not. *Assert* corresponds to invoking `contains(`*RV*`, `*PV*`)`, `assertInv` to `InverseOf(p:RelationalProperty).contains(`*RV*`, `*PV*`)` which is equivalent to invoking `contains(`*PV*`, `*RV*`)`. The operators are applicable to all sub-types of `RelationalProperty`: using `assert` together with a taxonomy property refers to the implicit order of the taxonomy, `contains(`*RV*`, `*PV*`)` evaluates to `true` if and only if `isDescendant(`*RV*`, `*PV*`)` evaluates to `true`. The `equivalent` operator also maps to `RelationalProperty. contains(x, y)`, but is only applicable to `Equivalence` properties, it only serves as a syntax sugar and for comprehensibility of the resulting code.

The `comparable` operator maps to `OrderProperty.comparable(`*RV*`, `*PV*`)`, its purpose is to check whether the two values are comparable given a partial order property. For total order properties, it always evaluates to `true`.

The set of the four taxonomy-dependent operators (`isParent`, `isChild`, `isAncestor`, `isDescendant`) directly maps to the corresponding methods of the `Taxonomy` property type. They correspond to the four common relations which are derivable from a taxonomy. The purpose of presenting these as separate operators, is to avoid the need to derive the corresponding relational properties from a source taxonomy property every time there is a need to leverage one of these common relations.

All the property operators presented so far belong to the *relational* operators – they correspond to basic operations acting upon a binary relation. In the rest of this section we present two *functional* operators, whose behavior is more complicated from the algorithmic point of view. The semantics of the operators described in the following

paragraphs refers to the semantics of the individual operators, the semantics of an entire query is explained in section 5.7.3 Query Semantics.

`bestMatch` is extrinsic constraint-relaxing operator. It is probably the most powerful and useful Versatile operator: it starts with a given context node – the actual property value *PV* obtained from the `ValueProvider` and tries to perform the exact match, if no resource is found, it uses `Taxonomy#getAncestorIterator()` to generate a sequence of candidates in ascending order, leveraging the classification hierarchy of the taxonomy. With properly designed taxonomies in place, one can thus easily implement quite sophisticated *fall-back* strategies using hierarchical *defaulting* via property value *generalization*. The operator is used to express *preferences* rather than strict constraints.

Let us reuse the *ScreenSize* taxonomy on page .

Let us assume the property value *PV* = QVGA,

the following search sequence will be generated:

1. QVGA
2. PDA, SmartPhone
3. CompactDevice, ScreenPhone
4. "/" (root = property value not set = the universal concept)

Example 10: bestMatch example using ScreenSize taxonomy

The beauty of combining the `bestMatch` operator with hierarchical classification of a taxonomy is, that if there is only one variant of a particular resource, it does not need to be annotated at all: no matter what property values come in, at the end the default version can always be fetched; only those resource variants intended for a particular sub-class of property values need to be annotated. This allows to start application development and complete an end-to-end prototype with the default set of resources and then incrementally refine selected resources by providing multiple variants as needed to improve the user's experience.

Another example using the UNSPSC taxonomy (http://www.unspsc.org/),

given *PV* = `43232203` (File versioning software)

the following search sequence will be generated:

1. `43232203` (File versioning software)
2. `43232200` (Content management software)
3. `43230000` (Software)
4. `43000000` (Information Technology Broadcasting and Telecommunications)
5. "/" (root = property value not set = the universal concept)

Example 11: bestMatch example using the UNSPSC taxonomy

While the `bestMatch` is suitable for most situations, sometimes it may not be desirable to generalize all the way up to the default, unannotated resource variant. The `assertLevel` operator has been introduced as a way to constrain the `bestMatch` by providing an upper-bound. Because at the time of designing a particular query template the actual property value is not known, we can not specify the upper-bound using a constant – we do not know which branch of the taxonomy will be effective during the query evaluation. Therefore the `assertLevel` operator introduces the `level` attribute to specify the distance (in the taxonomy hierarchy) between the property value *PV* and the allowed resource property value *RV* as follows:

- `level = 0` - equivalent to the `assert` operator
- `level > 0` - absolute distance from the root of the taxonomy
- `level < 0` - relative distance from the context node (*PV*)

The Example 12 below taken from the *Versatile 1.0 API Reference* [VERSAPI] demonstrates the effect of assertLevel applied to bestMatch, please compare to the Example 11 above.

PV = `43232203` (File versioning software), `bestMatch` **`and`** `assertLevel(2)`,

the following search sequence will be generated:

1. `43232203` (File versioning software) [level 4, relative 0]
2. `43232200` (Content management software) [level 3, relative -1]
3. `43230000` (Software) [level 2, relative -2]

The entry `43000000` (Information Technology Broadcasting and Telecommunications) wont' match because its taxonomy level is equal to 1.

PV = `43232203` (File versioning software), `bestMatch` **`and`** `assertLevel(-1)`,

the following search sequence will be generated:

1. `43232203` (File versioning software) [level 4, relative 0]
2. `43232200` (Content management software) [level 3, relative -1]

Example 12: assertLevel and bestMatch example using the UNSPSC taxonomy

5.7.2 Result Set and Resource Entry

Result Set is an ordered collection of Resource Entries. The primary order of the collection is determined by the result *score* of individual resource entries in the collection. Score measures how closely a particular resource entry matches the *query*

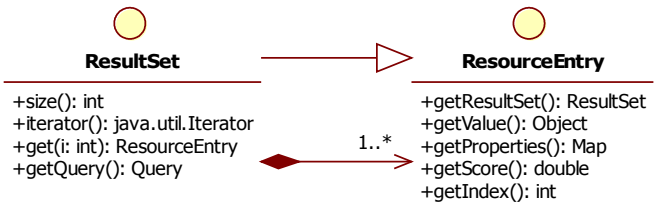


Figure 23: Result Set and Resource Entry

The query template interface allows to set the N-best size attribute which instructs the resource provider to return multiple results corresponding to a particular query in cases there is more than one entry in the resource repository which matches the query. In addition to that, because the framework supports the constraint-relaxing feature, the results may not exactly match the corresponding query; sometimes, a need to inspect the results may arise in order to debug or fine-tune the application, therefore there is a need to capture the actual metadata attributes of the resources retrieved by the resource provider. The Versatile framework uses *result set* and *result entry* to represent implement these requirements.

The `cz.cuni.versatile.api.ResultSet` interface represents the N-best result list produced by a `ResourceProvider` in a response to a particular `Query`. The `ResultSet` is ordered in descending order with the respect to the *score* of individual `ResourceEntry` items. `ResultSet` itself implements the interface `ResourceEntry` and thus exposes two facets to its users:

1. an ordered collection of `ResourceEntry` items
2. a shortcut accessor to the first (0-index) `ResourceEntry`

This approach has been chosen because of the default N-best size is equal to 1, and at the same time, a `ResultSet` always contains at least one item. In a typical situation, the user does not need (and does not want) to deal with a collection of result items and just wants to pick the first item.

The `cz.cuni.versatile.api.ResourceEntry` interface represents an individual item of the N-best result list. The `ResourceEntry` object contains not only the resource itself but also its metadata annotations (property/value pairs) and the *score* of the `ResourceEntry` with the respect to the corresponding `Query`.

5.7.3 Query Semantics

Now, when we have defined and discussed all the prerequisites, we can finally step ahead and describe the *formal semantics of Versatile query language*. When describing the semantics, we follow top-down approach: in this section: we describe the overall query semantics while avoiding to go into details regarding an important part of it: the *result entry score* and the *Versatile Scoring Function*, which is discussed separately in the following chapter. For now, we assume there is a scoring function, which calculates score for each resource entry in the result set and measures the quality of the individual entries with the respect to the query – how closely a given entry matches the query.

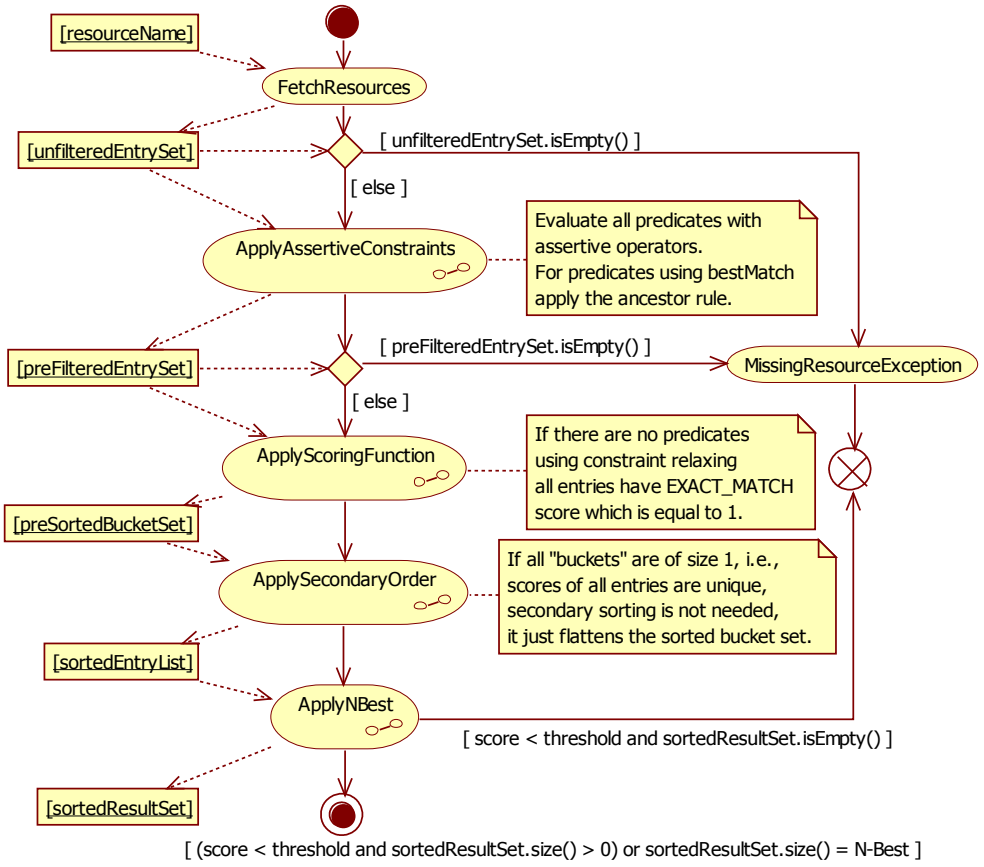


Figure 24: Query Operational Semantics

The query evaluation algorithm presented on Figure 24 above represents the *operational semantics* of the query, it should not be taken as a prescription for actually implementing the query evaluation in practice – all kinds of heuristics and optimization strategies can be put in place to speed-up query evaluation and make it less resource-intensive – as long as the alternative implementation gives the same results as the algorithm described in this section.

Step 1. Fetch Resources: given a resource name, all variants of the resource are retrieved

Step 2. Apply Assertive Constraints: all predicates with assertive operators (5.7.1) are applied, only those resource variants where all predicates evaluate to `true` are kept in the result set. In addition to that, for all predicates using the constraint-relaxing `bestMatch` operator, assert that `(`*RV* `=` *PV* `or Taxonomy.isAncestor(`*RV*, *PV*`)) = true`, i.e., for each resource entry annotation value RV and property value PV, make sure that either RV = PV (exact match) or RV is a *generalization* of PV – reachable by constraint relaxing. Only those resource variants where the above condition evaluates to `true` are kept in the result set.

Step 3. Apply Scoring Function: if there are no predicates using constraint-relaxing operators like `bestMatch`, the step can be skipped as all resource variants have score equal to 1 (exact match); otherwise the scoring function (5.7.4) is applied which ranks the resource variants with the respect to the preference predicates.

Step 4. Apply Secondary Order: if the scoring function is not *injective*, i.e., two or more entries attain the same score, the secondary order needs to be applied to the entries with the identical score. The secondary order applies *lexical ordering* following the order in which the properties appear in the predicate list (order of significance). To compare values of individual properties, the following fall-back strategy is used:

1. for an `OrderProperty`, the order the property is applied, in cases of a *partial order*, the `RelationalOperatorsRegistry.getTotalOrder()` is applied to avoid ambiguity.
2. for other properties, sorting leverages the standard Java library routines for sorting.

Step 5. Apply Score Threshold & N-best Filtering: the filter receives resource entries one by one in descending order and stops processing by firing `MissingResourceException` if the score does not pass threshold and the number of collected items is equal to zero; it stops processing by returning results as soon as one of the conditions is met:

1. the score of the incoming entry is lower than the score threshold and the number of collected items is greater than zero
2. the number of collected items reaches the N-best setting

The two special data structures *preference bag* and *preference chain* (see page 61) are not specifically mentioned in the steps above due to the brevity reasons. The table below summarizes their impact on individual phases of the query evaluation. While the preference bag only brings in more acceptable values, the preference chain overrides the property-defined order. Even more importantly, preference chain overrides the taxonomy in case of the `bestMatch` operator and the *preference chain itself acts as a taxonomy*: the first entry being a leaf node, the last entry being the root node and together with the intermediate nodes forming an upward chain without any branches.

	Preference Bag		Preference Chain	
	Assertive	Best Match	Assertive	Best Match
Step 1.	no impact	no impact	no impact	no impact
Step 2.	logical OR	logical OR	logical OR	overrides taxonomy
Step 3.	no impact	no impact	no impact	overrides taxonomy
Step 4.	no impact	no impact	overrides property order	overrides property order
Step 5.	no impact	no impact	no impact	no impact

Table 3: Semantics of PreferenceBag and PreferenceChain

5.7.4 The Scoring Function

> **Score** measures how closely a particular resource entry matches the query: Due to the framework's capability to execute approximate matching in addition to standard exact matching, we need to measure the *quality* of results with the respect to the original query. Higher score represents a better quality result.

The scoring function is designed in the way, so that the score of a particular resource entry is on the scale `(0.0, 1.0>`. The value `1.0` is called the *exact match score* and it means that no *constraint relaxing* (fall-back, generalization) activity took place. For queries containing only predicates with assertive operators all resource entries in the result set have *exact match score,* because the assertive operators have strict binary behavior: they either evaluate to `true` and the item is kept in the result set or they evaluate to `false` and the entry is removed from the result set, i.e., the assertive operators have no impact on the resource entry's score.

If a query contains one or more predicates with constraint relaxing operator (e.g. `bestMatch`), only those resource entries in the result set which *exactly match* the query have the exact match score. Exactly matching a query means, that for all predicates P_0 ... P_{n-1} using `bestMatch`, the query property value PV_i and resource entry property value RV_i are equal. As soon as constraint relaxing takes place, the score is lower than `1.0` and more a particular resource entry diverges from the query, the score is lower and converges towards `0.0`. The score effectively measures how far is a particular resource entry from the ideal candidate described by the query.

Another aspect to take into account is the relative significance of the individual predicates with the respect to the expected score: sometimes we consider all predicates more or less equally important while in other cases, some *soft constraints* (*preferences*) are much more important than other. We already stated earlier, that when building a query template, the predicates are added to the predicate list in order of their significance. This is the way how to tell which *preferences* are more important than other. In addition to that, there is a way to express the ratio of relative significance between subsequently added predicates by adjusting the *scoring factor* of the query template object.

All the above can be summarized in a single mathematical formula, which represents the *Versatile Scoring Function* applied in Step 3 of the algorithm described in the previous section. Let:

- $\vec{P}=(P_{0,}\dots,P_{n-1})$ be a vector of property predicates representing a query.
- $\vec{PV}=(PV_{0,}\dots,PV_{n-1})$ be a vector of property values of the vector $\vec{P}$
- $\vec{RV}=(RV_{0,}\dots,RV_{n-1})$ be a vector of property values (annotations) of a resource entry
- $\Delta(PV_i,RV_i)$ represent a distance between PV_i and RV_i in the taxonomy
- Φ be the scoring factor in the interval `(0, 1>`

then the Versatile Scoring Function is defined as:

$$score\left(\vec{PV},\vec{RV},\Phi\right)=\frac{1}{1+\sqrt{\sum_{i=0}^{n-1}\left(\Delta(PV_i,RV_i)\cdot\Phi^i\right)^2}}$$

Figure 25: The Versatile Scoring Function

To better understand how the scoring function has been constructed, let us look at a special case; let $\Phi=1.0$ (it is called the *neutral scoring factor*):

$$score\left(\vec{PV},\vec{RV},1\right)=\frac{1}{1+\sqrt{\sum_{i=0}^{n-1}\left(\Delta(PV_i,RV_i)\cdot 1^i\right)^2}}=\frac{1}{1+\sqrt{\sum_{i=0}^{n-1}\left(\Delta(PV_i,RV_i)\right)^2}}=\frac{1}{1+\|\vec{PV}-\vec{RV}\|}$$

Using the neutral scoring factor $\Phi=1.0$ all the property predicates become equally significant and $\|\vec{PV}-\vec{RV}\|=\sqrt{\sum_{i=0}^{n-1}\left(\Delta(PV_i,RV_i)\right)^2}$ represents the distance[47] of the result entry represented by $\vec{RV}$ from the original query represented by $\vec{PV}$ in an N-dimensional *Euclidean* space, each property corresponding to one dimension. It is obvious, that in case of the exact match, the score will be equal to 1 as the distance between the query and the resource entry is 0. On the other hand, when the distance between the query and the resource entry grows, the score converges to 0.

Now let us have the second look at the scoring function using slightly different notation:

$$score\left(\vec{PV},\vec{RV},\Phi\right)=\frac{1}{1+\sqrt{\Delta(PV_{0,}RV_0)^2+\left(\Delta(PV_{1,}RV_1)\cdot\Phi\right)^2+\dots+\left(\Delta(PV_{n-1},RV_{n-1})\cdot\Phi^{n-1}\right)^2}}$$

Figure 26: The Versatile Scoring Function (expanded syntax)

The effect of the scoring factor becomes more apparent: its impact multiplies with each dimension and for scoring factor in the interval (0, 1) it makes each subsequent predicate less significant than its predecessor. For scoring factor greater than 1, the

47 Classical formula to calculate the magnitude of a vector Euclidean vector space

effect would be exactly opposite, as it would gauge distances in each subsequent dimension[48]. The framework comes with three pre-defined scoring factors listed in the table below:

NEUTRAL_SCORING_FACTOR	1.00	all property predicates are equally significant
DEFAULT_SCORING_FACTOR	0.99	small penalty of this scoring factor effectively prevents ambiguity of the result scores
BIASED_SCORING_FACTOR	0.10	for shallow taxonomies results in lexical ordering

Table 4: Predefined scoring factors

To better demonstrate the effect of the scoring factor on the value of the scoring function, we present the following two examples (Example 13 and Example 14), the first one compares `DEFAULT_SCORING_FACTOR` to `NEUTRAL_SCORING_FACTOR`, while the second one uses `BIASED_SCORING_FACTOR`. Please note the use of `BIASED_SCORING_FACTOR` in to ensure consistency in the use of localized resources in the user interface.

Δ(PV0, RV0)	Δ(PV1, RV1)	Δ(PV2, RV2)	Φ = 1	Φ = 0.99
0	0	0	1.00000	1.00000
0	0	1	0.50000	0.50502
0	1	0	0.50000	0.50251
1	0	0	0.50000	0.50000
0	1	1	0.41421	0.41787
1	0	1	0.41421	0.41663
1	1	0	0.41421	0.41543
1	1	1	0.36603	0.36835
0	0	2	0.33333	0.33782
0	2	0	0.33333	0.33557
2	0	0	0.33333	0.33333
0	1	2	0.30902	0.31289
1	0	2	0.30902	0.31245
0	2	1	0.30902	0.31159
1	2	0	0.30902	0.31073
2	0	1	0.30902	0.30986
2	1	0	0.30902	0.30944
1	1	2	0.28990	0.29300
1	2	1	0.28990	0.29196
2	1	1	0.28990	0.29092
0	2	2	0.26120	0.26412
2	0	2	0.26120	0.26313
2	2	0	0.26120	0.26217
1	2	2	0.25000	0.25251
2	1	2	0.25000	0.25187
2	2	1	0.25000	0.25125
2	2	2	0.22401	0.22575

Example 13: Scoring Function Results sorted by Φ = 0.99

48 Scoring factors > 1 are not supported by the framework, the same effect can be achieved by adding predicates to the query template in reverse order and using a scoring factor on the scale (0, 1>

Δ(PV0, RV0)	Δ(PV1, RV1)	Δ(PV2, RV2)	Φ = 1	Φ = 0.1
0	0	0	1.00000	1.00000
0	0	1	0.50000	0.99010
0	0	2	0.33333	0.98039
0	1	0	0.50000	0.90909
0	1	1	0.41421	0.90868
0	1	2	0.30902	0.90746
0	2	0	0.33333	0.83333
0	2	1	0.30902	0.83316
0	2	2	0.26120	0.83264
1	0	0	0.50000	0.50000
1	0	1	0.41421	0.49999
1	0	2	0.30902	0.49995
1	1	0	0.41421	0.49876
1	1	1	0.36603	0.49874
1	1	2	0.28990	0.49871
1	2	0	0.30902	0.49510
1	2	1	0.28990	0.49509
1	2	2	0.25000	0.49505
2	0	0	0.33333	0.33333
2	0	1	0.30902	0.33333
2	0	2	0.26120	0.33332
2	1	0	0.30902	0.33306
2	1	1	0.28990	0.33305
2	1	2	0.25000	0.33304
2	2	0	0.26120	0.33223
2	2	1	0.25000	0.33223
2	2	2	0.22401	0.33222

Example 14: Scoring Function Results sorted by Φ = 0.1

5.7.5 Resource Provider

Resource provider consumes a query, searches its underlying repository of metadata annotated resources and returns the resource (or – depending on query settings – a list of resources) which most closely corresponds to the metadata constraints expressed in the query.

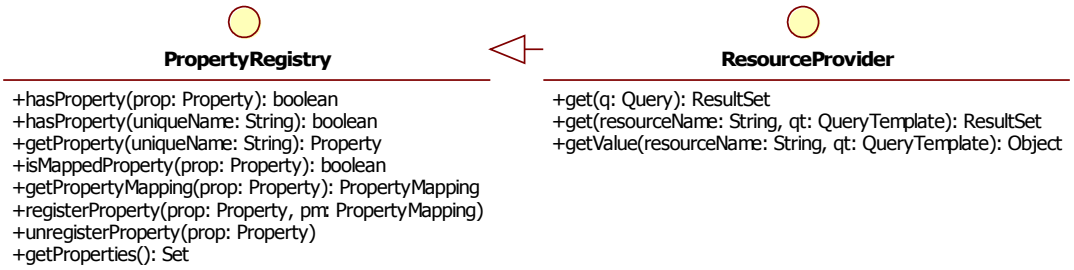


Figure 27: Resource Provider extending Property Registry

Resource provider (`cz.cuni.versatile.api.ResourceProvider`) consumes a query, searches its underlying repository of metadata annotated resources and returns the resource or – depending on the query N-best setting – a list of resources which most closely correspond to the metadata constraints expressed in the query. The framework specification does not assume any particular implementation or a data store type for the annotated resources, the only requirement is that all resource provider implementations must fully implement the query semantics as described in section 5.7.3 including the scoring function as per 5.7.4. The implementations of the interface are assumed to be specialized for a particular role or an environment. For example:

- a static resource bundle with metadata annotations attached to each variant of a resource (thus supporting many variants of the same resource label, message or graphics)
- a class or a component factory (producing pre-configured individual instances according to selected properties of the delivery context represented in the query)
- a meta-class factory (meta-component factory) (resource name being an interface name and query constraints used to look-up the most suitable class/component factory)
- an opaque content transformation/transcoding engine, which instead of searching for a pre-existing resource, finds the closets transformations it can execute and returns a result set in the form of *transformation handles* which, upon invoking `ResourceEntry.getValue()` on a particular entry execute the actual content transformation generating content of the qualities described in the metadata descriptor of the selected resource entry.

In a typical configuration, we expect multiple purpose-specific `ResourceProvider` instances to correspond to a single general-purpose `DeliveryContext` instance within a particular application scope (e.g. a module or a component). On the other hand, resource providers are more-likely to be re-used across multiple application scopes.

The careful reader has probably noticed (Figure 27, above) that the `ResourceProvider` interface extends `PropertyRegistry` and it is in fact a sibling of the `DeliveryContext` interface. The reasoning behind that, is to let the user to enumerate all built-in properties known to a particular `ResourceProvider`, as well as to register new *custom properties* via property mappings. The main purpose of using the property mappings in the `ResourceProvider` is to allow for a kind of *reverse* mapping: The mappings in the `DeliveryContext` transform raw domain-specific metadata into cleansed application-centric data structures. In a perfect world, the resource repositories are tagged using the application-centric annotations, but this may not be always possible in practice. The `ResourceProvider` may need to translate the queries it consumes to the original raw metadata or a third-party metadata vocabulary: in the example given in section 2.4.2, Figure 8 and Figure 9, we canonicalize *MIDP 1.0*, *MIDP/1.0*, *MIDP-1.0* and *Profile/MIDP-1.0* values into *MIDP 1.0*. In case the resource repository is tagged by raw data, we need to map MIDP 1.0 back to the original set of entries by constructing a reverse mapping *MIDP 1.0* -> `PreferenceBag(`*MIDP 1.0*, *MIDP/1.0*, *MIDP-1.0*, *Profile/MIDP-1.0*`)`

An important consequence of the above is, that when registering a mapped property in the `ResourceProvider`, the semantics of the `PropertyRegistry .registerProperty(Property, PropertyMapping)` is exactly the opposite to the `DeliveryContext`: we are not transforming **to** the mapped property but instead **from** the mapped property to the built-in properties of the `ResourceProvider`.

6 Conclusion

6.1 Overview

Let us start the conclusion of the thesis by re-iterating the key take-away (section 5.1):

The main idea behind the Versatile framework is describing device capabilities (*requirements*) and application artifacts (*provisions*) using semantically rich properties – mostly *hierarchical classifications* (taxonomies) – and employing the semantical information captured in the properties for implementing a best-effort (approximate) requirements/provisions matching algorithm. Thanks to the application of hierarchical classifications, the best-effort algorithm can incrementally generalize the requirements while searching for the artifacts most closely corresponding to device capabilities. This ability of *constraint relaxing* via *generalization*, allows for extremely efficient metadata annotation of application artifacts: using generic property values for shared resources while using more specialized property values for resources intended for specific device clusters or even individual devices. In addition to the above, the framework provides services for flexible definition of *priorities* and *resolution rules* for property value acquisition from multiple sources and services for property *transformations* including *canonicalization*, *information extraction* and *information synthesis*.

6.2 Goals Evaluation

6.2.1 Functional Aspects Evaluation

In chapter 3 (Setting the Goals) we listed the key issues we were aiming to address:

1. *Metadata Consolidation* (multiple overlapping metadata sources issue)
2. *Metadata Canonicalization* (inconsistent metadata issue)
3. *Level of Abstraction Gap* (knowledge representation issue)
4. *Domain Expertise Issue* (learning curve issue)
5. *Best Practices Enforcement* (separation of concerns, modularity)

In section 4.1.1 (Functional Considerations) we further detailed the functional aspects of these requirements. Let us now look in more detail whether and at what extent these issues are actually being addressed by the Versatile framework presented in this thesis.

1. Metadata Consolidation is dealt with by using the concepts of central *delivery contex*t and chained *value providers*. The delivery context serves as a one-stop-shop for retrieving all metadata used for versioning and configuration purposes. The *value provider chain* concept allows to setup the metadata resolution rules and policies in a high-level yet flexible way. The rules are easily modifiable and if needed, they can be setup uniquely on an individual metadata entity level.

2. Metadata Canonicalization is dealt with by applying the metadata transformations (*property mappings*) as a service embedded in both the *delivery context* and the *resource provider*. The transformations, once setup and configured, are completely

opaque to the application developer and the transformed metadata entities are accessible in the very same way as the original raw metadata. Moreover, the mapping functions can evolve and improve without having any disruptive effect on the application code base.

3. Level of Abstraction Gap is addressed by promoting semantically rich metadata entities, relational properties, order properties and especially by emphasizing the hierarchical classifications – *taxonomies*. The semantical metadata enrichment is technically implemented using the property mapping mentioned in the paragraph above. The framework has its unique query apparatus allowing to express the metadata constraints and preferences in a straightforward and comprehensible way. The key construct of the query language is the operator for approximate matching which allows to implement sophisticated constraint-relaxing strategies without tedious coding – only by employing properly designed metadata taxonomies.

4. Domain Expertise Issue was kept in mind while designing all the aspects of the framework. The framework is specified in terms of object-oriented API. Its users do not need to posses knowledge from the domain of meta-modeling and ontologies or more specifically the knowledge of the *Semantic Web* technology stack. Besides familiarity with object-oriented programming, the framework only requires some rudimentary knowledge of high-school algebra (binary relations) and a grasp of the principle of hierarchical classification, which is natural to almost all typed object-oriented languages like C++, Java or C#.

5. *Best Practices Enforcement:* The framework is designed so that it encourages best programming practices by applying separation of concerns and emphasizing modularity: the property value acquisition rules are isolated to value providers, data transformations are performed by property mappings and constraints are expressed using templates without specifying the actual property values which helps to raise the level of abstraction and allows to isolate the versioning code from the application logic and modularization of versioning rules into individual property mappings.

6.2.2 Technical Aspects Evaluation

In section 4.1.2 (Technical Considerations) we discussed the technical aspects the framework must address in order to become practically applicable in the problem domain. This section attempts to evaluate the framework against the technical considerations – the performance aspects.

Some of the performance considerations are natively built into Versatile:

- Query is an immutable object which lets the Resource Provider to implement efficient result caching, as long as the content of the resource repository itself is also immutable.
- Property Mappings contain One2ManyMapping which allows to implement information extraction maps (parsing and interpreting composite literal values, see section2.1.1, *User-Agent* HTTP header example) in an efficient way by making sure such parsing and interpretation for multiple derived values can happen in one pass.

The end-to-end process of the Versatile Framework covering both the query creation from a pre-existing template and query evaluation consists of the following steps:

1. property value acquisition (value providers)
2. property mappings evaluation (delivery context)
3. property mappings evaluation (resource provider)
4. search & match execution (resource provider)

Property value acquisition performance directly depends on the performance of the value providers for the individual metadata sources and the number of metadata sources in use. There is no way the Versatile Framework can improve or control performance on this level – it is the responsibility of individual value provider implementors.

Property mappings evaluation depends the computational complexity of individual property mappings as well as depth and breadth of the property mappings dependency tree. However, there are two possible approaches to improve performance:

1. individual property mappings can implement caching to calculate subsequent evaluation requests faster – this can be implemented by direct mapping of input values to output values. The disadvantage of this method is that its success depends on the implementors of the individual property mappings.

2. (2 &3) delivery context / resource provider (property registry) caching: the direct map of inputs and outputs is build for entire property mapping dependency trees. This approach avoids the dependency on the implementors of the individual property mappings, on the other hand, the number of combinations can be significantly higher, which may negatively affect memory footprint and speed of cache searches

Search & match execution can be sped up on the level of entire queries as mentioned above, thanks to the fact, that Query is an immutable object. In general case, In other cases, the search & match task can be improved by using algorithm optimized for a particular constrained Resource Provider:

For example, consider a specific Resource Provider, which only understands a pre-defined set or properties and none of them is a taxonomy: the only way to pass a taxonomy value to such a resource provider is to register a property mapping to map the taxonomy to one of the build-in properties, so the taxonomy value never reaches the actual search & match phase. It means in turn, that the Resource Provider needs to implement only a subset of the Versatile operators (for example, there is no need to implement bestMatch, assertLevel, isChild, isParent, isDescendant, isAncestor, no need to care about scoring function, etc.). Such a constrained Resource Provider may be justified in some specific cases and its search & match algorithm can be certainly more efficient than a generic algorithm with the complete semantics as per section 5.7.3.

In the general case, there are two major tasks in the evaluation of the query (search & match):

1. “search” the resource variants for a given resource name (Fetch Resources), this step is completely Resource Provider dependent, assuming a relational database with an index on the resource name field, the complexity is O(log(n))

2. “match”: filter the set of variants using the constraints and preferences (Apply Assertive Constraints, Apply Scoring Function, Apply Secondary Order, Apply N-Best):

1) Apply Assertive Constraints – O(n x m_1), where n is the number of variants, m_1 is the number of assertive predicates
2) Apply Scoring Function – O(n x m_2), where n is the number of variants, m_2 is the number of constraint relaxing predicates
3) Apply Secondary Order – O (n.log(n) x m) where n is the number of variants, m is the number of all predicates
4) Apply N-Best – O(n)

As a conclusion of this section, we claim, that the Versatile Framework carefully addresses the performance aspects, however, there are too many unknown variables – dependencies on the external (user provided and/or third party) components whose impact can not be evaluated in a general case without knowing the detailed specifications of these external parts.

6.3 Related Work Evaluation

We choose to compare the related work in tabular way. We start with an overview of all solutions included in the comparison and then we present side-by-side comparison for pairs of solutions. We include the framework presented in this thesis into the tabular comparison in order to make it easier to directly compare to other solutions.

Identifier	Title	Location	Focus Features
Versatile	The Versatile Framework	chapter 5 p. 51	delivery context, variant selection, generic versioning framework
S1	Related Standards	sec. 2.2 p. 18	delivery context
O1	WURFL	sec. 2.3.1 p. 24	delivery context
O2	DELI + Capability Classes	sec. 2.3.1 p. 25	delivery context
C1	Volantis Mobile Content Framework	sec. 2.3.2 p.26	an end-to-end multimodal framework
C2	MobileAware Interaction Server	sec. 2.3.2 p.27	an end-to-end multimodal framework
R1	Adapting multimedia Internet content for universal access	sec. 2.3.3 p.28	content adaptation, delivery context
R2	An End–End Approach to WirelessWeb Access	sec. 2.3.3 p.28	delivery context, content adaptation
R3	Enhancing pervasive Web accessibility with rule-based adaptation strategy	sec. 2.3.3 p.29	content adaptation, delivery context
R4	Device-independent web browsing based on CC/PP and annotation	sec. 2.3.3 p.30	content adaptation, delivery context
R5	Graceful Degradation: a Method for Designing Multiplatform Graphical ...	sec. 2.3.3 p.30	model-based adaptation cascaded
R6	Tool-supported single authoring for device independence and multimodality	sec. 2.3.3 p.31	abstract UI-based adaptation
R7	Context-Aware Adaptation for Mobile Devices	sec. 2.3.3 p.31	delivery context, variant selection, content adaptation
R8	Experiences in Using CC/PP in Context-Aware Systems	sec. 2.3.3 p.32	delivery context

Table 5: Related Work Overview (focus features in order of significance)

Criteria	Versatile	S1 (standards)
Metadata Consolidation	**Good**: allows to retrieve metadata from multiple sources, user-defined resolution rules for individual properties (value provider chains)	**Out of scope**: the framework assumes everybody is using CC/PP for every purpose, DELI supports legacy devices via static device repository
Metadata Canonicalization	**Good**: property mappings let the developer to implement canonicalization rules in Java or another language embeddable in Java	**Out of scope:** raw metadata accessible
Level of Abstraction Gap	**Good**: semantically rich properties can be designed and their values derived from raw metadata using property mappings; constraints and preferences expressed using a high-level query language	**Out of scope:** raw metadata accessible
Domain Expertise Issue	**Moderate**: in case one of the raw metadata sources is CC/PP, the developer in charge of implementing the value providers has to be familiar with CC/PP and UAProf	**Moderate:** requires knowledge CC/PP and UAProf
Best Practices Enforcement	**Good**: separation of concerns (value provider chains, properties, property mappings, constraints and preferences), modularization of inference rules into individual property mappings, separation of actual variant selection from the application code (resource providers)	**Out of scope:** provides raw metadata retrieved from CC/PP profile, consolidation with other sources, canonicalization and device clustering is up to the application developer

Table 6: Versatile versus Related Standards

Criteria	O1 (WURFL)	O2 (Capability Classes)
Metadata Consolidation	**Moderate:** the device repository can be extended with attributes not included in CC/PP (UAProf), however, as the repository is static, the profiles can not be updated to reflect the at runtime changes (e.g. the user pressing mute button, or changing screen orientation	**Out of scope**: the framework uses CC/PP as the exclusive source of metadata, DELI supports legacy devices via static device repository
Metadata Canonicalization	**Good**: the profiles are reviewed before loading them into the device repository and therefore the most common mistakes (typos, value inconsistencies) are manually corrected	**Moderate**: the rules used to define capability classes can partially mitigate the canonicalization issue
Level of Abstraction Gap	**Moderate**: in general (using WURFL API) raw metadata are accessible, as opposed to CC/PP (UAProf), whenever possible, the UAProf attributes are converted to boolean to simplify the conditions; When using WALL tag library, the level of abstraction is raised significantly by using WALL as an abstract user interface language	**Good**: carefully designed capability classes raise the level of abstraction significantly
Domain Expertise Issue	**Good**: the users do not need to be familiar with CC/PP, UAProf and other Semantic Web technologies, but needs to learn WALL tag library instead	**Moderate:** requires knowledge CC/PP and UAProf
Best Practices Enforcement	**Moderate**: when using WALL tag library, the users can focus on single authoring the application in WALL and avoid mixing of the versioning related code with the application; there is no best practices enforcement when using WURFL API directly	**Moderate:** due to raising the level of abstraction, the amount of versioning code can be much lower comparing to plain DELI, on the other hand the framework is not aiming at best practices enforcement

Table 7: WURFL and DELI (with Capability Classes extension)

Criteria	C1 (Volantis)	C2 (MobileAware)
Metadata Consolidation	*Moderate*(?): the device repository can be extended with attributes not included in CC/PP (UAProf), the repository seems to be static (unable to reflect the runtime changes), the company provides repository update subscription to its customers	*Good*(?): the device repository can be extended with attributes not included in CC/PP (UAProf); according to Figure 5 on page 27, the framework seems to support runtime updates of the delivery context via CC/PP and HTTP request/session APIs
Metadata Canonicalization	*Good*(?): the data stored in the device repository are reviewed manually cleaned up	*Good*(?): the data stored in the device repository are reviewed manually cleaned up
Level of Abstraction Gap	*Good*(?): the framework provides XDIME (XHTML Device-Independent Mark-Up Extensions) abstract user interface language	*Good*(?): the framework uses a custom extension of XHTML with *mobility tags* to let the designer annotate content with rendering hints and alternatives for mobile devices
Domain Expertise Issue	*Good*(?) When using the abstract UI authoring language, the designer does not need to be familiar with CC/PP and UAProf, on the other hand, needs to learn XDIME	*Good*(?): When using the abstract UI authoring language, the designer does not need to be familiar with CC/PP and UAProf, on the other hand, needs to customized XHTML
Best Practices Enforcement	N/A – not enough information	N/A – not enough information

Table 8: Volantis Mobile Content Framework and MobileAware MIS[49]

Criteria	R1 (Adapting multimedia ...)	R2 (An End-End Approach ...)
Metadata Consolidation	**Out of scope:** as a pre-CC/PP framework, it does define its own delivery context and does not specify what inputs are needed to construct instances of delivery context at runtime	**Out of scope**: the framework proposes to use a pre-defined set of device classes identified by URI stored in the CC/PP extension header, the URI itself is used to identify the device class
Metadata Canonicalization	**Out of scope:** the framework does specify how the delivery context is constructed	**Out od scope**: not needed due to the above
Level of Abstraction Gap	**Good**: the adaptation is driven by *InfoPyramid* and the corresponding content *fidelity* function – without the need to write the adaptation (transformation) rules manually	**Moderate**: the level of abstraction is raised, the problem is that it seems to be raised too much: if a new device class is to be supported by an application, the designer needs to consume the implicit knowledge hidden behind the profile URI to understand the semantics of the device class (device capabilities)
Domain Expertise Issue	**Out od scope:** as a pre CC/PP framework, it does not depend on Semantic Web, on the other hand, the proposed delivery context is not sufficiently rich (from the today's perspective)	**Moderate**: the user does not need to fully understand CC/PP and UAProf, only a need to know CCPP transport layer (HTTP extensions)
Best Practices Enforcement	**Good:** separation of concerns and modularity is implied by the framework design	**Good**: separation of concerns and modularity is implied by the framework design (fixed adaptation strategy and content adaptation using XSLT templates)

Table 9: Adapting multimedia Internet content for universal access and An End–End Approach to WirelessWeb Access

49 The grades for the two commercial frameworks are *estimates* only, because the information publicly available on company websites and in W3C position documents is not sufficient to *objectively* evaluate the frameworks.

Criteria	R3 (Enhancing pervasive Web …)	R4 (Device-independent web …)
Metadata Consolidation	**Out of scope:** this work represents a delivery context as a collection of attributes, each attribute being a controlled vocabulary (an enumeration of possible values); the article does not discuss how such a profile is instantiated and what sources are needed	**Out of scope**: the framework uses CC/PP as the exclusive source of metadata, DELI supports legacy devices via static device repository
Metadata Canonicalization	**Out of scope:** this framework does not require any canonicalization as its delivery context is canonicalized by its definition	**Moderate**: the constraints used to individual resource variants can partially mitigate the canonicalization issue
Level of Abstraction Gap	**Good:** the enumerated values of the delivery context are specified using high-level abstractions, the resource annotations are using similar approach as the delivery context, the matching and adaptation is driven by a declarative rule engine	**Good**: using structured and annotated content together with sophisticated adaptation algorithm sufficiently raises the level of abstraction
Domain Expertise Issue	**Moderate**: the user does not need to know Semantic Web stack, but does need to be fairly familiar Jess rule language	**Moderate:** requires knowledge CC/PP and UAProf in order to annotate resources
Best Practices Enforcement	**Moderate**: the separation of concerns is ensured, on the other hand, the modularity is somewhat compromised by using a single global rule-base which can grow extensively in real world applications featuring more refined delivery context (with more attributes and attribute values)	**Good:** separation of versioning and application code is achieved by using declarative resource annotations, modularity is ensured by using annotations on page level – separate set of structured definitions for each individual "screen".

Table 10: Enhancing pervasive Web accessibility with rule-based adaptation strategy and Device-independent web browsing based on CC/PP and annotation

Criteria	R5 (Graceful Degradation …)	R6 (Tool-supported single authoring …)
Metadata Consolidation	**Out of scope:** the author introduces *Platform Model* which is based on UAProf vocabulary with a few modifications; the article does not discuss how such a model is instantiated and what sources are needed	**Out of scope:** the framework uses a local device profile repository which is used during the adaptation process; however, I was not able to find other examples than those directly referring to pre-defined device classes
Metadata Canonicalization	**Out of scope:** due to the above, the work does not discuss the need for canonicalization	**Out of scope:** the to the above, it is assumed that canonicalized data are already stored in the repository
Level of Abstraction Gap	**Moderate:** the UAProf vocabulary was extended with arbitrary attributes (e.g. Category of the device) which raise the level of abstraction for pre-selected features	**Moderate:** the applications are first prototyped for selected *device classes* and once the designs are approved, the final version is implemented in the UIML metalanguage and a set of style-sheets used to generate concrete user interface at runtime
Domain Expertise Issue	**Moderate:** the work requires a good knowledge of UML and Model Driven Architecture, requirement for CC/PP and UAProf knowledge is marginal, as the Platform Model is presented in the form of an object model (UML class diagram) rather than RDF model of triplets	**Moderate:** the users need to familiar with the UIML language and the MONA development methodology using a series of prototypes
Best Practices Enforcement	**Good:** the work presents a methodology driven by a cascade of model transformations powered by graceful degradation (GD) rules	**Good:** the framework proposes a realistic methodology of designing a series of prototypes and then developing an abstract user interface and separately a set of transformational style-sheets for individual device classes

Table 11: Graceful Degradation: a Method for Designing Multiplatform Graphical User Interfaces and Tool-supported single authoring for device independence and multimodality

Criteria	R7 (Context-Aware Adaptation ...)	R8 (Experiences in Using CC/PP ...)
Metadata Consolidation	**Out of scope:** the framework introduces a custom enriched delivery context inspired by CC/PP, but does not mention how the delivery context instances are actually created	**Out of scope:** the framework introduces a custom enriched delivery context based on CC/PP and UAProf, but does not mention how the individual components of the extended delivery context instances are created
Metadata Canonicalization	**Out of scope**: due to the above, it is assumed the delivery context data are already canonicalized	**Out of scope**: due to the above, it is assumed the delivery context data are already canonicalized
Level of Abstraction Gap	**Moderate**: XQuery is used to query the delivery context, which allows to raise the level of abstraction modestly, the resource annotation part and content negotiation/adaptation parts contribute to the favorable grade	**Out of scope**: extended set of raw metadata is accessible
Domain Expertise Issue	**Moderate**: the users need to be familiar with CC/PP and partially UAProf	**Moderate**: requires knowledge CC/PP and UAProf
Best Practices Enforcement	**Good:** separation of concerns and modularization are enforced by the framework	**Out of scope:** provides raw metadata retrieved from extended delivery context, device clustering is up to the application developer

Table 12: Context-Aware Adaptation for Mobile Devices and Experiences in Using CC/PP in Context-Aware Systems

6.3.1 Related Work Conclusion

The set of evaluation tables above attempts to evaluate the related work against the goals set for the Versatile framework. The evaluation does not attempts to judge the individual frameworks from the perspective of practical usability: some research work focused on multimodal applications authoring regards the goals we use as the evaluation criteria as marginal and simply presume some kind of delivery context to exist, sometimes in quite a simplified form (a set of device classes). This is quite understandable and some of the former work of the author suffers from the same issues (2.1.2). On the contrary the work presented in this thesis is fully focused on practically implementable solution for the acquisition, representation and manipulation of the delivery context, as well as efficient resource variant retrieval implemented using high-level declarative constraints and preferences. On the other hand, this work is touching the idea of the authoring methodology only marginally (1.3).

When comparing this work to selected commercial frameworks, it is important to note the scope of this thesis is limited versioning domain and does not introduce a concrete end-to-end multimodal application framework. On the other hand, this work is trying to define a generic broadly applicable framework in the form of platform agnostic concepts – contrasted to the presented commercial frameworks, which approach the domain of interest with solutions which are similar in many aspects, yet different enough so that code and application portability represents a significant issue: there is nothing like J2EE standard in the domain of multimodal computing, encouraging application portability between the platforms of different vendors.

6.4 Current Status

As a part of the framework design and evaluation, all the concepts of the framework were implemented as Java interfaces and accompanied by detailed API documentation [VERSAPI]. The design of individual interfaces goes down to the level of detailing all error conditions and error handling. Implementation considerations are also a part of the API documentation. The API is provided in the form of a pre-compiled API Java library and is available for download on the thesis web site[50]. The API library was actually used while authoring the examples used in this thesis to enforce syntax and type consistency of throughout the examples. Implementation of the entire framework – providing concrete implementations of all the abstract interfaces – was out of scope of the thesis authoring effort: the API specification alone, converted to PDF, is over 100 pages and describes in detail 45 artifacts (classes and interfaces)

50 http://dsrg.mff.cuni.cz/~gergic/versatile/

6.5 Alternative Applications

The case study presented at the beginning and consistently followed through the entire thesis is the domain of multimodal web applications. Nevertheless, the framework's principles are generic and the framework can be applied in other application domains, whenever there is a need to semi-dynamically match and bind requirements to provisions (e.g. a web-services runtime binding to one of a set of pre-defined – pre-approved - services). The term semi-dynamically is used for a good reason: as explained in section 2.4.2 Metadata Canonicalization, it is practically impossible to trust readily available unsupervised metadata. The need to review the metadata, develop canonicalization mappings and design hierarchical classifications effectively disables fully automated metadata-driven service discovery and binding. Nevertheless, a framework like Versatile becomes highly effective once the metadata landscape has been mapped and we are dealing with the problem of sorting out the best choice from the set of of available options.

Appendices

Index of Figures

Index of Tables

Index of Examples

References

[VXML03] *Voice Extensible Markup Language (VoiceXML) Version 2.0*, W3C Candidate Recommendation, W3C, 20 February 2003, http://www.w3c.org/TR/voicexml20/
[ICSM2001] V. Demesticha, J.Gergic, J.Kleindienst, M. Mast, L.Polymenakos,H.Schulz, L.Seredi: *Aspects of design and implementation of multi-channel and multi-modal information system*, Conference proceedings: ICSM 2001, Italy, 2001
[JG99] J Gergic: *A Versioning Model for SOFA/DCUP Architecture*, Master's Thesis, Charles University, 1999
[JG03] J Gergic: *Towards a Versioning Model for Component-based Software Assembly*, Conference proceedings: ICSM 2003, Amsterdam, The Netherlands, 22-26 September 2003
[JG99] J Gergic: *A Versioning Model for SOFA/DCUP Architecture*, , Charles University, 1999
[SA02] S. H. Maes: *A "single authoring" programming model: the interaction logic*, Conference proceedings: 2002 Symposium on Applications and the Internet, Nara City, Nara, Japan, 2002
[VERSAPI] *Versatile 1.0 API Reference*, API Reference, Charles University, 2007, http:/dsrg.mff.cuni.cz/~gergic/versatile/
[MIMEMT] *MIME Media Types*, Specification, IANA, 2007, http://www.iana.org/assignments/media-types/
[LC142] *Locale (Java 2 Platform SE v1.4.2)*, Java 2 API Reference, Sun Microsystems, 2003, http://java.sun.com/j2se/1.4.2/docs/api/java/util/Locale.html
[HTTP11] *Hypertext Transfer Protocol -- HTTP/1.1*, Specification, IETF, 1999, http://www.w3.org/Protocols/rfc2616/rfc2616.html
[CATCH2004] CORDIS RTD-PROJECTS: *Converse in AThens Cologne and Helsinki 2004*, Project Record, IST-1999-11103, © European Communities., 2002, http://cordis.europa.eu/
[IIWAS2001] Yannis Despotopoulos, George Patikis, John Soldatos, LazarosPolymenakos, Jan Kleidienst, Jaroslav Gergic: *Accessing and Transforming Dynamic Content based on XML: Alternative Techniques anda Practical Implementation*, Conference proceedings: IIWAS 2001, Linz, Sep 2001
[SEKE02] J. Gergic, J. Kleindienst, Y. Despotopoulos, J. Soldatos, G. Patikis, A. Anagnostou, L. Polymenakos: *An Approach to Lightweight Deployment of Web Services*, Conference proceedings: the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE '02), Ischia (Italy), 2002
[ICMI02] J. Kleindienst, L. Seredi, P. Kapanen, J. Bergman: *CATCH-2004 multi-modal browser: overview description with usability analysis*, Conference proceedings: Fourth IEEE International Conference on Multimodal Interfaces, Pittsburg, USA, 2002
[IWANLIS01] M. Mast, T Ross, H. Schulz, H. Harrikari, V. Demesticha, L. Polymenakos, Y. Vamvakoulas, J. Stadermann: *A Conversational Natural Language Understanding Information System for Multiple Languages*, Conference proceedings: 6th International Workshop on Applications of Natural Language to Information Systems, Madrid, Spain, 2001
[WAP20] *The WAP 2.0 conformance release*, WAP Forum conformance release, Open Mobile Alliance, 2001, http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[RB142] *ResourceBundle (Java 2 Platform SE v1.4.2)*, Java 2 API Reference, Sun Microsystems, 2003, http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html
[SEMWEB] *Semantic Web Activity*, Web Site, W3C, 2007, http://www.w3.org/2001/sw/
[SEMWEBVIS] *Berners-Lee and the Semantic Web Vision*, Journal Article, XML.com, 2000, http://www.xml.com/pub/a/2000/12/xml2000/timbl.html
[RDF04] *RDF Primer*, W3C Recommendation, W3C, 10 February 2004, http://www.w3.org/TR/rdf-primer/
[CCPP04] *Composite Capability/Preference Profiles (CC/PP)*, W3C Recommendation, W3C, 15 January 2004, http://www.w3.org/TR/CCPP-struct-vocab/
[UAP06] *OMA User Agent Profile V2.0*, Approved Enabler, OMA, 6 February 2006, http://www.openmobilealliance.org/release_program/uap_v2_0.html
[DELI] Mark H Butler: *DELI: A delivery context library for CC/PP and UAProf (Revised)*, Research Report, HPL-2001-260, HP Labs, 2002, http://www.hpl.hp.com/techreports/
[JSR188] *Composite Capability/Preference Profiles(CC/PP) Processing Specification*, Java Specification Request, Sun Microsystems, Inc., 2003, http://jcp.org/en/jsr/detail?id=188
[SPARQL] *SPARQL Query Language for RDF*, W3C Candidate Recommendation, W3C, 2007, http://www.w3.org/TR/rdf-sparql-query/
[XQUERY] *XQuery 1.0: An XML Query Language*, W3C Recommendation, W3C, 2007, http://www.w3.org/TR/xquery/
[CCPPIIFD] Mark H. Butler: *CC/PP and UAProf: Issues, Improvements and Future Directions*, Research Report, HPL-2002-35, HP Labs, 2002, http://www.hpl.hp.com/techreports/2002/HPL-2002-35.html
[SEMHYPE] Mark H Butler: *Is the Semantic Web Hype?*, Talk / Presentation, , HP Labs, 2003, http://web.archive.org/web/20040427115352/ http://www.hpl.hp.com/personal/marbut/isTheSemanticWebHype.pdf
[BARRIERS] Mark H Butler: *Barriers to the real world adoption of Semantic Web technologies*, Research Report, HPL-2002-333, HP Labs, 2002, http://www.hpl.hp.com/techreports/
[IDDWG05] *Input to Device Description Working Group*, Position Paper, W3C, 2005, http://lists.w3.org/Archives/Public/public-ddwg/2005Aug/att-0005/ddwgPositionPaper.htm
[OWL04] *OWL Web Ontology Language*, W3C Recommendation, W3C, 10 February 2004, http://www.w3.org/TR/owl-features/
[DIDCO06] *Delivery Context Overview for Device Independence*, W3C Working Group Note, W3C, 2006, http://www.w3.org/TR/di-dco/
[WURFL] *Wireless Universal Resource File*, Open Source Project, L Passani, A Trasatti, 2007, http://wurfl.sourceforge.net/
[CAPCLASS] Mark H. Butler: *Using capability classes to classify and match CC/PPAnd UAProf profiles*, Research Report, HPL-2002-89, HP Labs, 2002, http://www.hpl.hp.com/techreports/
[CAPPROF] Mark H. Butler: *Using Capability Profiles For Appliance Aggregation*, Research Report, HPL-2002-173 (R.1), HP Labs, 2002, http://www.hpl.hp.com/techreports/
[VOLANTIS02] *A Device-Independent Method for Web Site Authoring*, Position Paper, Volantis Systems Ltd., 2002, http://www.w3.org/2002/07/DIAT/posn/volantis/VolantisPosition.html

[VOLANTIS07] *Volantis Mobile Content Framework*, Product Overview, Volantis Systems Ltd., 2007, http://www.volantis.com/products/framework.php
[W3CMWI] *Mobile Web Initiative*, Web Site, W3C, 2007, http://www.w3.org/Mobile/
[W3CDIWG] *Device Independence Working Group*, Web Site, W3C, 2007, http://www.w3.org/2001/di/
[W3CUWA] *Ubiquitous Web Applications Working Group*, Web Site, W3C, 2007, http://www.w3.org/2007/uwa/
[MAWARE] *Mobile Interaction Servier / Device Recognition*, Company Web Site, MobileAware, 2007, http://www.mobileaware.com/device_rec.jsp
[MOHAN99] Rakesh Mohan and John R. Smith and Chung-Sheng Li, *Adapting Multimedia Internet Content for Universal Access*, Journal Article: IEEE Transactions on Multimedia, Volume (1), Issue (1), Pages (104-114), 1999
[MOHAN98] Chung-Sheng Li Mohan, R. Smith, J.R.: *Multimedia content description in the InfoPyramid*, Conference proceedings: 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, Seattle, WA, USA, 1998
[ESWA07] Stephen J. H. Yang, Norman W. Y. Shao, *Enhancing pervasive Web accessibility with rule-based adaptation strategy*, Journal Article: Expert Systems with Applications, Volume (32), Issue (4), Pages (1154-1167), 2007
[SCCPP01] Vladimir Korolev, Anupam Joshi: *An End-End Approach to Wireless Web Access*, Conference proceedings: 21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01), Los Alamitos, CA, USA, 2001
[JESS07] *Jess(R), the Rule Engine for the Java(TM) Platform*, Web Site, Sandia National Laboratories, 2007, http://www.jessrules.com/
[JSR94] *Java(TM) Rule Engine API Specification 1.0*, Java Specification Request, Sun Microsystems, Inc., 2002, http://jcp.org/en/jsr/detail?id=94
[HK06] Hwe-Mo Kim, Kyong-Ho Lee, *Device-independent web browsing based on CC/PP and annotation*, Journal Article: Interacting with Computers, Volume (18), Issue (2), Pages (283-303), 2006
[FLORINS06] Murielle Florins: *Graceful Degradation: a Method for Designing Multiplatform Graphical User Interfaces*, PhD Thesis, Université catholique de Louvain, 2006
[FLORINS04] Murielle Florins and Jean Vanderdonckt: *Graceful degradation of user interfaces as a design method for multiplatform systems*, Conference proceedings: 9th international conference on Intelligent user interfaces, Funchal, Madeira,Portugal., 2004
[USIXML] *UsiXML - User Interface eXtensible Markup Language*, Project Web Site, Université catholique de Louvain, Belgium, 2004-2007, http://www.usixml.org/
[TERESA04] Berti, S., Correani, F., Mori, G., Paternò, F., Santoro, C.: *TERESA: a transformation-based environment for designing and developing multi-device interfaces*, Conference proceedings: Conference on Human Factors in Computing Systems, Vienna, Austria, 2004
[CTT00] Springer, *Model-Based Design and Evaluation of Interactive Applications*, Springer, 2000
[SIMON05] R. Simon, F. Wegscheider, K. Tolar: *Tool-supported single authoring for device independence and multimodality*, Conference proceedings: 7th international conference on Human computer interaction with mobile devices & services, Salzburg, Austria, 2005
[MONA05] L. Baillie, R. Schatz, R. Simon, H. Anegg, F. Wegscheider: *Designing Mona: User Interactions with Multimodal Mobile Applications*, Conference proceedings: 11th International Conference on Human-Computer Interaction, Las Vegas, Nevada, USA, 2005

[MONA] *MONA - Mobile Multimodal Next Generation Applications:*, Project Web Site, ftw., 2007, http://mona.ftw.at/
[XFORMS] *XForms 1.0*, W3C Recommendation, W3C, 2006, http://www.w3.org/TR/xforms/
[SCHATZ05] R. Schatz, R. Simon, H. Anegg, F. Wegscheider, G. Niklfeld: *Developing Mobile Multimodal Applications*, Conference proceedings: HCI 2005, Edinburgh, Scotland, 2005
[BAILLIE05] L. Baillie, R. Schatz: *Exploring multimodality in the laboratory and the field*, Conference proceedings: 7th international conference on Multimodal interfaces, Torento, Italy, 2005
[TMN04] Traetteberg, H., Molina, P. J., Nunes, N. J.: *Making Model-Based UI Design Practical: Usable and Open Methods andTools*, Conference proceedings: 9th international conference on Intelligent user interfaces, Funchal, Madeira,Portugal, 2004
[MDM04] T. Lemlouma, N. Layaïda: *Context-Aware Adaptation for Mobile Devices*, Conference proceedings: 2004 IEEE International Conference on Mobile Data Management, Berkeley, CA, USA, 2004
[SAINT03] T. Lemlouma, N. Layaïda: *Adapted Content Delivery for Different Contexts*, Conference proceedings: 2003 Symposium on Applications and the Internet, Washington, DC, USA, 2003
[UPS02] T. Lemlouma, N. Layaïda: *Universal Profiling for Content Negotiation and Adaptation in Heterogeneous Environments*, Conference proceedings: W3C/INRIA: W3C Workshop on Delivery Context, Sophia-Antipolis, France, 2002
[XSLT] *XSL Transformations (XSLT)*, W3C Recommendation, W3C, 1999, http://www.w3.org/TR/xslt
[Indulska03] J. Indulska and R. Robinson and A. Rakotonirainy and K. Henricksen: *Experiences in Using CC/PP in Context-Aware Systems*, Conference proceedings: 4th International Conference on Mobile Data Management (MDM), Melbourne, Australia, 2003
[UAREP1] *UAProf Repository*, WWW Repository, W3Development.de, 2005, http://w3development.de/rdf/uaprof_repository/
[UAREP2] *UAProf Profiles*, WWW Repository, Nokia Corporation, 2007, http://nds.nokia.com/uaprof/
[CTOS] *Trading Object Service*, Specification, OMG, 2000, http://www.omg.org/
[JINI] *Jini Network Technology*, Java Technology, Sun Microsystems, 2007, http://www.sun.com/software/jini/
[UDDI] *UDDI Version 3.0.2*, Oasis Committee Draft, OASIS, 2004, http://uddi.org/pubs/uddi-v3.0.2-20041019.htm
[JSF] *JavaServer Faces Technology*, Java Technology, Sun Microsystems, Inc., 2007, http://java.sun.com/javaee/javaserverfaces/
[ODM06] *Ontology Definition Metamodel*, Recommended for Adoption, OMG, 5 June 2006, http://www.omg.org/ontology/
[FCA] *Formal concept analysis*, Encyclopedia entry, Wikipedia, The Free Encyclopedia, 2007, http://en.wikipedia.org/wiki/Formal_concept_analysis
[LATORD] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, 1990
[CONAL] G. Snelting: *Concept analysis-a new framework for program understanding*, Conference proceedings: SIGPLAN/SIGSOFT Workshop on Program Analysis For SoftwareTools and Engineering, Montreal, Canada, July 1998

[LATEO] F. J. Oles., *An application of lattice theory to knowledge representation*, Journal Article: Theoretical Computer Science, Volume (249), Issue (1), Pages (163-196), 2000
[PA20020198719] Gergic Jaroslav, Hosn Rafah A., Kleindienst Jan, Maes Stephane H., Raman Thiruvilwamalai V., Sedivy Jan, Seredi Ladislav, *Reusable voiceXML dialog components, subdialogs and beans*, United States Patent Application #20020198719, U.S. Patent & Trademark Office, 2002
[PA20030046316] Gergic Jaroslav, Kleindienst Jan, Maes Stephane H., Raman Thiruvilwamalai V., Sedivy Jan, *Systems and methods for providing conversational computing via javaserver pages and javabeans*, United States Patent Application #20030046316, U.S. Patent & Trademark Office, 2003
[PA20060036770] Hosn Rafah A., Gergic Jaroslav, Ling Nai Keung Thomas, Wiecha Charles, *System for factoring synchronization strategies from multimodal programming model runtimes*, United States Patent Application #20060036770, U.S. Patent & Trademark Office, 2006
[DMSP] *Distributed Multimodal Synchronization Protocol*, Internet-Draft, Internet Engineering Task Force, 2005-2007, http://www.ietf.org/internet-drafts/draft-engelsma-dmsp-04.txt
[CMAP06] J. D Novak & A. J. Cañas: *The Theory Underlying Concept Maps and How to Construct Them*, Technical Report, 2006-01, Florida Institute for Human and Machine Cognition, 2006, http://cmap.ihmc.us/Publications/ResearchPapers/TheoryCmaps/
[CMAP04] A. J. Cañas, G. Hill, R. Carff, N. Suri, J. Lott, T. Eskridge, G. Gómez, M. Arroyo, R. Carvajal: *CmapTools: A Knowledge Modeling and Sharing Environment*, Conference proceedings: First International Conference on Concept Mapping, Pamplona, Spain, 2004