Bind & Adapt
an Approach and Framework for
Information Visualization

by
Benoît Pointet

Thesis for the Master of Science in Computer Science
Supervised by Denis Lalanne

Document, Image and Voice Analysis (DIVA) Research Group
Department of Informatics - Faculty of Science - University of Fribourg

January 3, 2010
Abstract

This master thesis proposes to reconsider the visualization process by focusing on the relationships between data and its visual representations.

A novel ‘Bind&Adapt’ approach to visualization is presented, which considers data and visuals as partners in a structured process of bindings and adaptations, operation the visualization and supporting interaction upon it.

Reflecting this approach, a declarative language and ad hoc interpreter have been developed and integrated into the Metaphorea framework, a prototypal web-native declarative framework for visualization, which bases on the principles of declarative expression of visual mapping, reuse of visual forms, clear distinction and parity of the data and visual realms.

Through the lens of four use cases, this thesis discusses the approach and the framework, highlighting strengths and weaknesses. Key achievements are a unified and flexible model for both data and its representations, and a unified declarative language to express key features of a visualization such as data manipulation (aggregation, filtering, partitioning), visual mapping, and interaction coordination.

Further developments of the Metaphorea framework are proposed such as direct manipulation interface, view composition and generation of guides and scales, so to complete the fulfillment of the Bind&Adapt approach.
“Objects are relatively stupid. They do a few things well, as do lobsters. Intelligence resides in the system, not in objects.”


Acknowledgements

First, thanks to the supervisor of this thesis, Denis Lalanne, for his guidance, warnings, support, inputs and faith in the project. Thanks to the other members of the DIVA group, especially Enrico Bertini who the first expressed enthusiasm for the approach, and Ilya Boyandin for having accepted the role of ‘first-adopter’ of the approach.

Thanks to Michael Luggen and to my brother Abram for reviewing this thesis and providing many advices and hints. Although our discussions on the topic were too rare, I always enjoyed them.

Thanks to my parents who always encouraged me to go, with care and insight, my own way. And thanks to my brother Pierrick, for being so much to me.

Thanks to David, ‘companion in photography’, for the path we walked together the last five years. I am rich of all the images, people, and challenges we met together.

To Rachel, the one of each and every moment, love and gratitude.

Without them nothing; and them benevolent souls are so many. I do believe intelligence resides in their humanity.
# Contents

Abstract 2

Acknowledgements 3

List of Figures 7

List of Tables 7

1 Introduction 9
   1.1 Motivations ............................................. 9
   1.2 Thesis Content ........................................... 9
   1.3 Terminology .............................................. 10

2 Visualization on the Web 12
   2.1 Introduction .............................................. 12
   2.2 Visualization on the Web ................................. 12
      2.2.1 Audiences and Purposes ............................... 12
      2.2.2 Solution Models ....................................... 14
      2.2.3 Technological Platforms .............................. 15
      2.2.4 Data Models and Formats .............................. 15
      2.2.5 Interactive Representations .......................... 16
      2.2.6 Visual Mapping ....................................... 17
      2.2.7 Future Demands ...................................... 17
   2.3 Conclusion ................................................. 18

3 Bind&Adapt Approach 19
   3.1 Introduction .............................................. 19
   3.2 Vision ................................................... 19
   3.3 Inspirations : the Metaphor as Mechanism ............... 20
   3.4 Expected Benefits ........................................ 21
   3.5 Similar Approaches and Solutions ........................ 22
   3.6 Conclusion ................................................. 23

4 Visualization Framework 24
   4.1 Introduction .............................................. 24
6.5.1 Expressing Mapping: Bind to Represent ......................... 66
6.5.2 Expressing Manipulation: Bind to Aggregate, Filter, etc. .... 67
6.5.3 Expressing Coordination: Bind to Coordinate .................. 67
6.5.4 Expressing Animation ........................................... 68
6.5.5 Providing Guides & Scales ...................................... 68
6.5.6 Controlling Details .............................................. 69

6.6 Focus on Visualization Practices .................................. 69
6.6.1 Bind to Visualize and Explore through Direct Manipulation .. 69
6.6.2 SVG-Viz Adapter .................................................. 70
6.6.3 The Approach and the Market ................................... 70

7 Conclusion .............................................................. 71

A Specifications of the Binding Language ............................ 73
A.1 Introduction ......................................................... 73
A.2 BL Elements ......................................................... 73
A.2.1 <bind> elements ............................................... 73
A.2.2 <node> elements ............................................... 74
A.2.3 <range> elements ............................................... 76
A.3 Binding Language DTD ............................................. 77
A.4 NV script Specifications .......................................... 77
A.5 Scripting Attributes in BL Expressions ............................ 78

B Use Case Bindings ..................................................... 79
B.1 City Weather Binding .............................................. 79
B.2 Africa Stats Binding ............................................... 81
B.3 Bertin Hotel Binding .............................................. 83
B.4 Migration Graph Binding ......................................... 84

C Metaphorea Framework, v0.2 ........................................ 87
C.1 Framework Content ............................................... 87
C.2 Framework Usage .................................................. 88

Bibliography .............................................................. 89
List of Figures

3.1 Vision of representation .................................................. 20
4.1 Overview of the Metaphorea framework ............................ 25
4.2 The two axes of the binding process for the City Weather example . 27
4.3 NV Model : NV Metastructure Examples ............................ 29
4.4 City Weather visualization, from building blocks to end result .......... 36
4.5 City Weather, operational binding, partial schema ................. 37
4.6 Binding and matching countries to territories, step by step ............ 40
5.1 City Weather : resulting visualization solution ................... 46
5.2 Africa Stats : from readily-available documents to visualization .... 47
5.3 Bertin Hotel : bind to aggregate ......................................... 51
5.4 Migration Graph : when GraphML ‘pilots’ SVG ...................... 55
List of Tables

5.1 Use Case Performance Tests, Safari/Firefox Browser .................. 60

6.1 Advantages, Shortcomings and Possible Enhancements of the Bind&Adapt Approach and the Metaphorea Framework .................. 62
Chapter 1

Introduction

“Patterns are formed where ideas meet the evidence of the world.”

Colin Ware, Visual thinking for design [Ware, 2008]

1.1 Motivations

Data visualization is often considered in terms of two problems: how to represent data so to nurture understanding and insight through an enlighting bright image, and how to build this image, through an opaque code. Opacity of the code is due to its transformative and procedural nature, which, moreover, tends to break the semantic bounds that a representation has to the underlying fact, the bounds that relates a view to data.

The author believes that a relationship-based model should prevail in the implementation of visualization solutions, one that “never breaks the connection between the variables and the graphics that represent them” [Wilkinson, 2005], and allow end users to “see how their raw data is magically turned into colorful images” [Chen, 2005]. This would also narrow the gap that exists between the roles of producer and consumer of visualization, and thus support ‘mass education to visualization’.

Thus the desire for a declarative form to express the construction of a visualization, as well as interaction coordination and data exploration in it.

1.2 Thesis Content

After a brief overview of ‘Visualization on the Web’ (Chapter 2) which contextualizes it, this master thesis proposes a novel approach that bases on the bounds between
data and its representations (Chapter 3). This approach is implemented in a visualization framework composed of a descriptive language to express such relationships, an engine to interpret them and a visualization layer to interface it with users and context (Chapter 4). Several use cases exemplify usage of the framework (Chapter 5) and form a concrete substrate for discussion and precision of further development directions (Chapter 6).

The Binding Language (BL) specifications compose Appendix A. Appendix B lists source code excerpts pertaining to the use cases. Appendix C details the version of the framework which is discussed in this thesis.

1.3 Terminology

A few concepts that may appear in this thesis and which may not possess a consensual meaning in the community, need to be precised.

The term data is used to describe any information encoded in any format, although most of the time in an XML grammar, whereas the terms visual, visual forms or views describe data with an inherent visual meaning/embodiment.

The concept visualization, spelled à l’américaine, is to be understood as the active process of relating data to – a compound of – visual forms. It is in this sense very close to the one of representation. Were it to react over time to its context and human actions, a representation/visualization is said to be interactive.

In order to avoid the use of the generic term user, we shall precise the specific roles involved in visualization. A visualization consumer benefits from a visualization by observing, browsing and more generally exploring data through this visualization, which has been constructed and deployed by a visualization publisher, based on some data from a data provider, on some views designed by a graphic designer, as well as some logic framework from a developer.

Some technological terms and acronyms shall also be introduced. XML refers to the eXtensible Markup Language\(^1\), version 1.0. HTML refers to the most frequent version of the HyperText Markup Language: HTML4\(^2\) and HTML5\(^3\) to its promising successor. SVG refers to the most used version of Scalable Vector Graphics format: SVG1.1\(^4\).

\(^1\)http://www.w3.org/XML/
\(^2\)http://www.w3.org/TR/html4/
\(^3\)http://www.w3.org/TR/html5/
\(^4\)http://www.w3.org/TR/SVG11/
ecmascript\textsuperscript{5} refers to the scripting language used in browsers, of which JavaScript\textsuperscript{6} and JScript\textsuperscript{7} are ‘community’ variants. JSON\textsuperscript{8} refers to the Javascript Object Notation format.

\textsuperscript{5}http://www.ecmascript.org/
\textsuperscript{6}https://developer.mozilla.org/en/JavaScript
\textsuperscript{7}http://msdn.microsoft.com/en-us/library/yek4tbz0(VS.85).aspx
\textsuperscript{8}http://www.w3.org/TR/SVG11/
Chapter 2

Visualization on the Web

“I keep saying the sexy job in the next ten years will be statisticians. People think I’m joking, but who would’ve guessed that computer engineers would’ve been the sexy job of the 1990s? The ability to take data — to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it — that’s going to be a hugely important skill in the next decades, not only at the professional level but even at the educational level for elementary school kids, for high school kids, for college kids.”

Hal Varian, Hal Varian on how the Web challenges managers [Varian, 2009]

2.1 Introduction

This chapter surveys the practices and tools for web visualization, which forms the field of application for our framework.

2.2 Visualization on the Web

2.2.1 Audiences and Purposes

Visualizing Public Data  Ten years ago, administrations & institutions were reluctant to publish any data on the web. Most of them were lacking the necessary legal and technological infrastructures to do so. Nowadays, not only do the legal aspects and technological aspects take place, but administrations now also have the new mission to make data available and intelligible, not to say digestible, to the mass. The visual channel being the most effective one when it comes to express certain types of informations, it is no wonder that administrations tend to look for effective visual solutions.
The recent initiative of the US government to ‘open data’\(^1\) gave birth to many online solutions to share, visualize and discuss ‘public data’, partly governmentaly operated, such as Data.gov\(^2\), partly private, such as Socrata\(^3\).

**Visualizing Business Data** Available and intelligible data has long become a necessity for businesses, which need to capitalize on the data they generate by making it internally available and explorable. So-called *business dashboard* or *cockpit* solutions are integrated to business management systems (e.g. SAP BusinessObjects Dashboard\(^4\)). Visualizations are often published by collaborators in charge of reporting, and consumed by decision takers.

**Visualizing News Data** Visualization needs in the fast-paced world of news possesses the urgency of the business visualization needs, but the broad audience of the administrations. Issues bound to journalistic ethics and impact on the audience are particularly sensitive in the news business, where a visualization is often a visual statement replacing a thousand ‘facts’. Such visualizations are published by infographics teams attached or external to news agencies, and consumed by the masses.

**Visualizing Personal Data**

*The Age of Also means that on a daily basis I get faxes, I also get emails, I also get snail mail, I also receive many phone calls, I also receive CD-ROMS and DVDs... We live in the Age of Also. For the next decades, there will not be a single best way to receive information, rather a choice.* [Wurman et al., 2001]

Personal data is data relevant to a person or a small community, rather than the sole data owned and generated by them. Few individuals and groups have time and skills to publish visualization solutions to integrate their data, which often composes an heterogeneous corpus of numbers, words, sentences, tables and documents from a great variety of sources. Personal visualization needs are relatively recent in respect to business and administrative needs.

---

\(^1\)See [http://www.wired.com/politics/onlinerights/magazine/17-07/mf_cio](http://www.wired.com/politics/onlinerights/magazine/17-07/mf_cio) for an interesting introduction to the subject.

\(^2\)http://data.gov/

\(^3\)http://www.socrata.com/

2.2.2 Solution Models

To answer the early needs for visualization solution on the web, two types of visualization tools have emerged: libraries and online hosted tools. These types do not pertain to the sole visualization field but correspond to two diametrically opposed models of web applications: the solution-centric vs problem-centric models.

Solution-centric Systems are hosted solutions (aka online tools, web apps) which provide a complete infrastructure to manage data visualization and exploration. Examples are Daytum\(^5\) for personal data, ManyEyes\(^6\) and Tableau\(^7\), targeted at larger corpuses of (business) data. They have the undeniable advantage to integrate representation, exploration, annotation and publishing of visualizations as a coherent project, and hide many maintenance problematicstics, but the disadvantage to capture one’s data and one’s business processes into a proprietary generic framework, system and methodology.

Problem-centric Model: Ad-Hoc Solutions Ad-hoc solutions are designed, deployed and heavily customized around a project. They either are based on Service, client-server or client-side architectures. Serviced solutions propose the generation of visual and interactive diagrams as web services (in a broad sense). The client inputs a number of parameters to the service, which generally returns an image representation, or a snippet of interactive document (e.g. Google Visualization API\(^8\), Google Charts\(^9\), Wordle\(^10\)) Client-server solutions combine the power of the server for computationally intensive tasks like queries, statistics, ... with the agility of the browser for the interface. They often are fully owned by the customer. Client-side solutions maintain all computation on the browser, although they may be combined with prior data server-side manipulation. Example libraries to build such solutions are ProcessingJS\(^11\), CakeJS\(^12\) and Flare\(^13\).

It is this latter type, clientside solutions, that we shall focus on, since it is free of any server constraints.

\(^5\)http://daytum.com/
\(^6\)http://manyeyes.alphaworks.ibm.com/manyeyes/
\(^7\)http://www.tableausoftware.com/
\(^8\)http://code.google.com/apis/visualization/
\(^9\)http://code.google.com/apis/chart/
\(^10\)http://www.wordle.net/
\(^11\)http://processingjs.org/
\(^12\)http://code.google.com/p/cakejs/
\(^13\)http://flare.prefuse.org/
2.2.3 Technological Platforms

The browser is the platform for the web, but it also is the place of many disparities. Visualization Solutions generally base upon one of the following technological platforms:

- Server side generation of static charts and graphs as raster or vector graphics,
- Microsoft Silverlight\(^{14}\),
- Adobe Flash Platform + Flex\(^{15}\)
- HTML5 + ecmascript + canvas\(^{16}\) + SVG

It is this latter technological bundle that we shall focus on in this project, since it uses solely non-proprietary technologies.

2.2.4 Data Models and Formats

Data Inputs Two formats of data are broadly used in client-side solutions: XML and JSON, the latter being quite recent but with a quick-growing community, recent libraries like JsViz\(^{17}\) and CakeJS rely on JSON for data input. Although the two formats have, for the average uses, the same expressiveness power, no canonical mapping between the two exists\(^ {18}\), thus any conversion between the two has to decide upon some conventional mapping. These datas are fetched through ajax queries (HTTP Data Requests) from the browser.

Internal Data Model It is generally required that input data be in a format related to the computational data model. Most data models follow well-known structural archetypes: relations (tables), trees (hierarchies) and graphs (networks). The relational model prevails, many years after the introduction of document-based (hierarchical) data models.

Data Manipulations The online tools and libraries differ strongly in the possibilities of data manipulations (sorting, filtering, clustering, etc.) provided. Most of the libraries take as granted that input data is ‘pure data’ and that it does not need manipulations,

\(^{14}\)http://www.microsoft.com/SILVERLIGHT/
\(^{15}\)http://www.adobe.com/products/flex/
\(^{16}\)http://en.wikipedia.org/wiki/Canvas_(HTML_element)
\(^{17}\)http://www.jsviz.org/blog/
\(^{18}\)see http://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html
or they provide the visualization publisher programmatical hooks to manipulate data. Larger integrative tools like Tableau consider data manipulation an important part of visualization and provide efficient and refined solutions to it. Providing data manipulation is as much a question of complexity, as it is of computational power, browser-based manipulation are very costly, or of marketing, clients stay with the solution holding their data as long as new features follow up key needs.

2.2.5 Interactive Representations

Most solutions provide mapping to many common visual forms such as dot/line plots, bar/pie charts, ... The support for recent forms is very unequal: most of them will include only some of the more recent visual forms such as treemaps, word clouds and data flows. This limited palette of forms has negative aspects: the visualization publisher is reluctant to use/buy a library that does not include his favorite visual forms, and the publisher often has no non-programmatical means to extend the range of visual forms. Some libraries (Google Visualization API) do provide means to extend the visual palette, others to combine existing forms into complex views like Protovis$^{19}$.

**Animation** Since visuals are drawn/generated by functions and procedures, their animation is most of the time implemented through a *drawing loop*, also said to be *frame-based*, meaning that the visual is rendered as a (fast) sequence of static images. Animation is the most disregarded aspect of visualization, probably because of its complexity, and thus certainly an interesting field of investigation.

**Customization** The degree of customization of the representation is directly proportional to the publisher’s effort to deploy the visualization: lighter solutions will only provide some CSS styling, while heavier tools tend to be more customizable. The problem lies in the generative model: in the simpler cases, decision of visual features are made just about anywhere in the generative process.

**Interaction** Some libraries provide means of interaction with visuals, mostly selection, zoom and pan, and detail type (as with Infovis Toolkit$^{20}$). The customization of interaction is often made possible by an event model, where programmatical hooks for each event are left at publisher’s disposal. The interaction is however always performed upon the visual object, and do not reflect directly on the underlying data structure.

---

$^{19}$http://vis.stanford.edu/protovis/

$^{20}$http://thejit.org/
**Exploration**  Exploration of data is the main process that a visualization supports. Exploration happens through observation, visual query, dataset refinements, extension of visualization, and various interactions and manipulations of views and their elements. Few solutions address the issue of exploration by extension of a visualization.

### 2.2.6 Visual Mapping

Mapping – establishing the relationship between the data and its visual counterparts – is of key importance for a visualization solution. In the broadest sense, writing the code that is responsible for the mapping between the data and its representation constitute the core part of the deployment of a visualization solution. It is, in general terms, the most time-consuming, the most frustrating, and the most cryptic part. The author believes it is a direct consequence of the gap between the publishing/construction models and user conceptual model of a visualization (i.e. mental model of how a visualization ‘works’).

**Generative Approach to Mapping**  The idea behind the generative approach is that the visual be ‘drawn by data’ or that the ‘data is boiled into visuals’: programmatical statements take data as input and generate visual forms as output. This approach has the undeniable advantage to be often computationally effective, and capable of drawing any form. It however requires strong programming skills and since its product is a program, changes and maintenance costs are high.

**Component-Based Approach to Mapping**  The component-based approach advocates that a composite visual system be ‘composed and fed by data’: visual blocks, with internal logic and interaction facilities, are composited and linked so to act in a coordinated way onto input data. These components are also computationally effective, and facilitate reuse of visual forms. The component-based approach is often the best choice for rapid deployment of highly-customized visualization solutions. However, adding new components is cumbersome, and linking the components together requires programmatical skills and good understandings of the components architecture.

### 2.2.7 Future Demands

Businesses and administrations now consider visualization as an important analytical and communicational mean, and integrate it in their culture. They mostly use online
hosted tools, desktop apps and large ad-hoc solutions. Individuals and groups of individuals start to demand visualization solutions as amount and heterogeneity of relevant data augments. This new demand requires the visualization field to “cross the chasm” and enter mass consumption, culture, and production. This may imply, non-exhaustively:

- implementation models that are closer to the mental model of representation,
- coherent set of concepts for the field,
- abandon of programmation in favor of declaration or direct manipulation to express visualization systems [Shneiderman, 1997],
- focus on heterogeneous and middle-sized and semi-structured data sets,
- unified models and interfaces to them, leveraging technological differences,

Some tools like Tableau are ahead on this front, allowing to build and publish complete dashboards and complex custom visualizations. They however fully hide to the consumer how her data is turned into the bright image. Libraries are flourishing but all target code-literate visualization publishers. Even the Google Visualization API which aims at federating efforts by providing a framework extensible on the data part as much as on the visual and publishing part, fails to consider non-coders, and thus fails to consider the mass.

2.3 Conclusion

The visualization field as many other fields related to data capitalization is bursting nowadays, evenmore ‘on the Web’. This burst of innovation will see many tools and ideas come and go. It is the author belief that a strong focus should be on bringing visualization to the mass.

In order to open web visualization practices to mass consumption and production, adequate models, idioms and metaphors have to be found. This constitutes an opportunity to question the generative approach and propose alternatives.
Chapter 3

Bind&Adapt Approach

“The most important thing to design is the user’s conceptual model. Everything else should be subordinated to making that model clear, obvious, and substantial. That is almost exactly the opposite of how most software is designed.”

David Liddle, Bringing design to software [Winograd, 1996]

3.1 Introduction

One of the keys to ease up the deployment of a visualization solution is to provide descriptive means to express the solution, means related to the conceptual model common to the users of a visualization. We here express a vision of interactive representation to visualization purposes motivated by the linguistic model of the metaphor and outline the potential of such vision. Finally we review potentially similar existing approaches.

3.2 Vision

We see the process of representation as a process of iteratively refining a structure of relationships and influences between the represented and the representee, in visualization-oriented terms: a dynamic structure of bindings and adaptations between data and visual forms. We name this the Bind & Adapt approach, and schematize this vision in Figure 3.1.

The concept of binding possesses a sense of symmetry: when binding a data object to a visual object, we do not (yet) consider whether one shall influence the other. Complementary, the concept of adaptation possesses a strong notion of asymmetry: generally, the
data influences the visual form, that is, the visual is adapted to the data. The distinction between symmetrical bindings and asymmetrical adaptations is clearly pertinent to interactive representations used in graphical interfaces, in which forms often also modify data (e.g. in a graphical editor).

This approach advocates data and visual parity, it considers them on a same level, as embodiments of information. It also states a clear separation of data and visual realms (both structure and logic), and that a third realm brings them together, the realm of binding and adaptation.

### 3.3 Inspirations : the Metaphor as Mechanism

This vision resembles closely to the one of a metaphor, thus the framework code-name Metaphorea, “The essence of a metaphor is understanding and experiencing one thing in term of another experience.” [Lakoff and Johnson, 1980]. The concept seems quite adequate to summarize our vision, and the structure-mapping theory already expresses similar ideas [Zhang, 2007]: “According to structure-mapping theory (Gentner and Markman, 1997), the mapping between the source and target referent in a metaphor is a process of establishing a structural alignment between two represented referents, which consists of an explicit set of correspondences between the representational elements of the two referents, and high-order relations between the relations”. Although this theory is not directly related to the visualization field, it hints at the fact that an
important part of cognition is metaphoric, and that the metaphoric model may be a pertinent model for cognitive aids such as visualization solutions.

it also emphasizes the importance of structure within data and visual forms, introduced by J. Bertin in [Bertin, 1981]: “Information is a relationship. But this relationship can exist among elements, subsets or sets. And these three levels must be retained in the subsequent graphics...” and confirmed by latter works: “adding multiscale visual structure in graphics” enhances their power, “making search much more efficient” [Ware, 2008].

The concept of metaphor has been under high pressure in the Human Computer Interaction (HCI) community: “A metaphor is always wrong, by definition.” [Norman, 1999], “Strict adherence to metaphors ties interfaces unnecessarily to the working of the physical world.” [Cooper et al., 2007]. These criticisms are indeed thoughtful warnings: metaphorical expression is intuitive and frequent in our daily communications, and one may wrongfully consider it as the ‘ACME’ solution when comes the time of imagining a representation.

However, the Bind&Adapt approach considers metaphors as a mechanism and not as visual representation, a mechanism that it uses and applies to data and abstract visual forms, be the latter metaphorical representations or not.

This approach can be machine-aided in many ways. The machine may provide means to interface data and forms through accessible structures, so to abstract from their descriptive formats, or means to instanciate the structure of representational relationships, means to allow the consumer to extend and refine these relationships, in a malleable space between data and form.

### 3.4 Expected Benefits

This descriptive and structural approach of the data, the visual form and their relationships, has, over the procedural approach, the following presumed advantages:

+ clear distinction of the tasks of preparing the data, preparing the visual form, and building the representation,

+ clear separation of logics of the data, visual forms and representation,

+ coordination of state and actions between the data and visual forms as the heart of the system, and not as mere options,
iterative building of the visualization solution through a declarative language and/or direct manipulation.

3.5 Similar Approaches and Solutions

The idea of expressing representation through bounds tying the data to the visuals is not new; neither is the use of the metaphor as inspiring concept. However, to base the visualization onto declarative description of such bounds is relatively new. We here give a rather non-exhaustive overview of akin approaches and technological solutions, a deeper study would be necessary to write the genesis of the approach.

**Form-Semantics-Functions**  Simoff [Simoff, 2008] proposes the Form-Semantics-Functions (FSF) framework, which considers visualization as the development of a metaphor and describes a series of steps that “ensure the correctness of the development of the representation of the metaphor” [Simoff, 2008]. It also contains a same structured relationships model: “As a rule, metaphorical mappings do not occur isolated from one another. They are sometimes organized in hierarchical structures, in which lower mappings in the hierarchy inherit the structures of the higher mappings.” However, the FSF framework is to be understood as a methodology, active at conceptual and semantic level, and not at the functional level.

**Snap-Together**  North [North, 2000] designed a coordinated-view framework that makes it possible for a user to compose and coordinate multiple views of a data body in a coordinated way. The coordination happens at interaction level, based on the relational data model: user actions upon some objects in a view (select/deselect) are bound to other actions to the related objects in the connected views, leading to action synchronization. To do so, each view has to propose a set of possible actions that can be done in them as API.

**View-Coordination Architecture for Information Visualization**  Pattison et al. [Pattison and Phillips, 2001] take a generic approach to the issue of view coordination and adapt the Model-View-Controller model to integrate interaction coordination. It emphasizes the fact that “view coordination cannot be implemented solely through data-centric coordination mechanisms”, and advocates the “separation of responsibilities for the specification and implementation of the mapping from data model to view model”.

Both approaches are highly interesting, however we are interested in *coordination* as a principle that concerns not only interaction, but other aspects of visualization as well.

**eXtensible Binding Language**  Mozilla User-Interface framework also contains the concept of *bindings* in the eXtensible Binding Language \(^1\) (XBL). This however is used to detail the logic and content elements of an interface element rather than relate the interface elements to some data and is thus to be understood as the expression of compositing, rather than relating.

**Degrafa Bindings**  Degrafa\(^2\), a declarative graphics framework based on the Adobe Flex framework, possesses a very similar sense of relating data and form: degrafa *bindings* create a relationship between two properties so that when one updates the other is dynamically updated. These binding concept belongs in facts to the culture of the Flex framework. It is however by no mean symmetrical and rather to be considered as dynamic parameters to a closed components.

### 3.6 Conclusion

We propose an approach to representation as a structured process of binding and adapting data and visual elements; a structured process to be expressed in a declarative form and direct manipulation.

It contrasts with a generative approach which ‘boils’ data into visual forms, and doing so often breaks the links between them. Our approach also differs from the component-based approach in the sense that it approaches data and visual symmetrically and in an open manner, whereas the component-based approach composites a visual system and feeds data to it.

Its most radical proposition is the clear distinction between the three ‘realms’ of data and visuals, and their relations; a distinction advocating their parity.

---

\(^1\)http://developer.mozilla.org/en/XBL
\(^2\)http://degrafa.org
Chapter 4

Visualization Framework

“it is the internal mobility of the image which characterizes modern graphics. A graphic is no longer drawn once for all; it is constructed and reconstructed (manipulated) until all the relationships which lie within it have been perceived.”


4.1 Introduction

Based on the Bind&Adapt approach, a language and an interpreter for this language, as well as an upper applicative layer have been developed as a prototypal proof of concept and development playground. This chapter presents the architecture of the framework they compose, details design decisions for the Binding Language (BL), and overviews key algorithms and constructs of the process generated by expressions in the Binding Language, called the binding process.

4.2 Framework Overview

Figure 4.1 schematizes the Metaphorea framework. Based on some concrete data and visual documents, which the system abstracts through adequate adapters into respective metastructures, the binding engine put in relations and adapts the visual form and data metastructures accordingly to an operational binding structure corresponding to a binding description. Adaptations are further repercuted onto concrete data and visual objects. A user interface layer (UI Layer) provides interaction with the system to the visualization consumer\(^1\).

\(^1\)Hopefully, it is through the same UI Layer that, in a future version of the framework, the visualization publisher will be able to build the visualization solution
4.2.1 Technological Context

In coherence with the view of web visualization expressed in Chapter 2, we here restrict our development to client-side web-based visualization for small to medium size data (less than 10’000 records). Furthermore, we focus on XML-based data and visuals documents (including HTML, SVG, ...), and rely on cross-browser ecmascript for the implementation logic.

To port the problem in the realm of XML documents seems reasonable, since XML itself is a good contender for data, able to express a wide range of data structures (and semi-structures), and SVG for interactive visuals, as [Wilkinson, 2005] puts it: “the use of XML for specifying graphics is natural since charts are treelike, charts are thought of in terms of major elements such as axes and bars, and because the extensibility of XML allows us to plan for unthought of future graphics”. Moreover, since our approach is descriptive, XML provides a good syntax for expressing bindings. And since binding documents imply that they be live (as DOM objects), a web browser is currently the best environment for live documents rendering and both user and scripted manipulations.

4.2.2 Introductory Use Case : City Weather

To present the framework, we introduce a simple use case. It concerns the visualization of some weather data (namely latitude, longitude, number of days with precipitations
and average temperature) for a couple of cities from Western Europe. Our goal is to represent this data in both a wordplot graph (words positioned in a 2D space) and a table, and somehow coordinate interactions such as selection/deselection (by mouse-click) and mouse hovering in both views.

The data is expressed in raw XML, the table in HTML, the wordplot in SVG.

### 4.3 Binding Language Design

To describe the binding and adaptation of data and visual documents, we rely on an ad-hoc language, coined the *Binding Language*. No language, be it XML-based or not, seemed to fulfill all requirements, and composing with many languages would have brought a level of complexity that was not desirable at design level.

We indeed need a language to describe the two main axes of the binding process and their interplay: *metastructuration* and *binding & adaptation*.

The first axis, metastructuration, is the issue of providing uniform access to data and visual objects, which inherently have different data and logic structures. This abstraction mechanism turned out to be necessary to cope with the great heterogeneity of data and visual forms. The second axis, binding & adaptation, represents the heart of the approach: once data and visuals are accessible and manipulable, they can be put in relation and adapted meaningfully.

Figure 4.1 presents how these two axes structure the framework. In our example, those axis represent, first, the concrete problem of abstracting the data, the table and the wordplot to a uniform model where, second problem, they can bind and adapt, as shown in Figure 4.2.

Interestingly enough, the FSF methodology [Simoff, 2008] proposes a series of steps that relate to our two axes. The first steps of FSF, which are “identification of the source and target spaces of the metaphor”, “conceptual decomposition of the source and target spaces”, “identification of the dimensions through which the metaphor operates” correspond to the metastructuration axis of the binding process, while the last FSF step, “establish semantic links, relations and transformations between the concepts in both spaces” corresponds to the binding & adaptation axis.
4.3.1 Data Metastructure

The structure of a document is not intuitive: salient properties of a visual may be hidden as some styling property, data values may be encoded either as XML attributes or as content of an XML elements, ... These peculiarities, inherent to each document, need to be lifted up by some higher-level model, to provide an interface to the low-level data structure and values. In other words, to alleviate data format so to concentrate on data. Unlike other systems which transform input data into instances of the working data model, the binding engine, does not transform but interfaces data and visual sources, thus the term metastructure.

Of key importance is that this metastructure provide access by name to (sets of) elements. For example, one may need to query all dots in a given scatterplot (i.e. children of ... expressions), or all dots that are siblings of a given dot. Another important requirement upon this document metastructure is to allow the description of domains of values. Since property values are going to be adapted, it is important to known their domains.
Metastructures vs XML Typing languages  One could imagine to work with other meta descriptions of documents, like tagging document parts with structural descriptions (through special attributes), using one of the various XML typing languages, like XML Schema\(^2\), DTD\(^3\), etc. But such schematics languages 'do too much': they describe the structure of a document to its finer structural details, whereas our needs here are not only to type but also to hide structural details, keeping the salient structure. Embedding structural description in documents (i.e. tagging) is simply not an option when documents may from various uncontrolled sources.

The solution retained is to include in the Binding Language the ability to describe a high-level metastructure to any document, through structured queries of some kind (XPath\(^4\), CSS selectors\(^5\), etc.). A metastructure able to encompass most data models.

The first designs of the metastructure mimicked the document model, a hierarchy of elements and attributes. However it turned out to be inadequate: some properties being encapsulated in sub-elements, while others in attributes. Moreover, the behaviour of elements and attributes were very similar. This lead to a design where document, elements, attributes, are all considered as nodes of a directed graph.

Node-Value : a Model for Metastructures  The Node-Value (NV) Model is inspired by the Object Model\(^6\), the Entity-Relationship Model\(^7\) and the Entity-Attribute-Value Model\(^8\).

A Node-Value metastructure is a graph: a NV metastructure is made of nodes and directed edges, each node holds a value, each edge holds a name. Since the NV graph is built from hierarchical sources (data and visual documents), an implicit _parent_ edge is automatically built to keep trace of hierarchy.

An edge, with name _foo_, leading from node _A_ to node _B_, reflects the following statements:

- _B_ is the _foo_ of _A_,
- _B_ is a child/property node of _A_.

In our example, this model allows to informally express the following metastructures verbally (and schematically in Figure 4.3):

\(^2\)http://www.w3.org/XML/Schema
\(^3\)http://en.wikipedia.org/wiki/DTD
\(^4\)http://www.w3.org/TR/xpath
\(^5\)http://www.w3.org/TR/CSS2/selector.html
\(^6\)http://en.wikipedia.org/wiki/Object_model
\(^7\)http://en.wikipedia.org/wiki/Entity-relationship_model
\(^8\)http://en.wikipedia.org/wiki/Entity-attribute-value_model
Chapter 4. Visualization Framework

**Figure 4.3:** NV Model : NV Metastructure Examples

- **data.xml**: a single `data` node has many `city` nodes, each having some `longitude`, `latitude`, `average-temperature`, and `precipitation-days`, each holding some value.

- **wordplot.svg**: a `wordplot` contains `words`, each holding some textual value and having some `x` and `y` positions and a `size`, all holding numerical values.

- **table.html**: the `table` has a `head` and many `rows`, all having `cells`, which hold some textual value.

**Querying Metastructures** Such NV metastructures should support queries by chaining ids in a dot-notation fashion, starting from a node. We shall name such such a query **NV query**. example: `data` returns all its child nodes named data (i.e. bound with `data` edges), if none is found, then start the same query from the `_parent` node, `data.city.name` similary looks for all `data` nodes, then aggregates all child `name` nodes of all child `city` nodes of those `data` nodes. The result of a node query, is always a set of nodes, sometimes empty.

Wildcards should be used to express all nodes, regardless of the name on the edge that relate them, e.g. `city.*` should return not only the `name` node of the item (generally one, though), but also its `latitude` and `longitude` nodes, etc. Expression of combination should be possible through another wildcard, the vertical separator ‘—’: `city.y—city.x` should combine the results of queries `city.y` and `city.x`. Expression of node filtering should also be possible `âla` XPath: `city/name='Berlin'`. 
Value Ranges

One then can extract the set of values from queried nodes, or compute statistics upon such set. To do so, and to map values between nodes of different types, one should be able to specify in the metastructural description the value domain of a node. We here propose a simplification of such a domain by reducing it to a range (i.e. an ordered, 1-dimensional domain), either empirical (gathering a set of values from the meta-structure and ordering them) or theoretical (parametrized, functionally defined ordered set). This decision aims to simply the expression of value adaptation and mapping.

Tying Metastructure to Document

Each document (or more generally, data source) possesses a certain format, and thus needs appropriate logic to load the document, get and set values, and perform queries. This should be outsourced to some adapter modules, following an adapter interface, so to allow the possibility to add new adapters to the framework, to handle new data and visual formats and/or features.

4.3.2 Binding Structure

The binding structure acts upon the metastructures and not the data structures themselves. Bindings should happen on node sets. Bindings should allow composition, in order to express the hierarchy of relationships: bindings should be able to contain bindings. Moreover, parent/global bindings shall provide a context for children/local bindings, so that adaptation benefits both from local nodes as from general global sets of nodes.

Bindings are temptatives. They may success, and be called complete bindings, and allow nested bindings to operate, concerned nodes to adapt, or may fail, and stop deeper propagation of the binding process.

4.3.3 Adaptation

Adaptation of documents parts happen in the context of a binding. Two types of adaptation are necessary: structural and value adaptation.

Structural Adaptation

Structural adaptation is necessary when the structures of bound documents don’t fit; e.g. our table doesn’t have enough rows to match all cities in our data set. Often some visual nodes are missing, ... This missing node problem shall be solved by generation or cloning of the right document part, appended at the right place.
To avoid lengthy procedural description of structural adaptations, and because there seems to be classical versions of the problem, a strategy approach is taken. Describing structural adaptation thus becomes choosing the right adaptation strategy. Note that only structure and not value is adapted at this level.

**Value Adaptation**  Value adaptation is often considered as the mapping of data properties onto visual properties. Values are adapted through mapping of values, copying of values or setting of computed value. To express the dynamic adaptation, we choose a functional scripting notation to manipulate node values, numbers, strings and sets of those types. The ecmascript language is chosen for the expression of value adaptations, its simple syntax and the fact that the framework already uses it as implementation language.

In our example of weather visualization, both the set of rows in the table and the set of words in the wordplot have to be structurally adapted to the set of cities. Value adaptation happens when the name of a city, is passed as the value of its corresponding word in the wordplot.

### 4.3.4 Imbrication of Bindings and NV Metastructures

The bindings structure and the NV metastructure compose two distinct tree-like structures which together perform the binding in an intertwined manner. It should be described in a single tree-like (XML) structure. Two strategies exist so far to have them coexist: either bind and extend metastructures locally to a binding, or describe metastructures and reference them later at various binding levels.

The first strategy injects the metastructure of a same document onto the various levels of a binding hierarchy. Thus the metastructure of the document is not clearly understandable, and poorly reusable. On the opposite, the second strategy, to first describe metastructures and then bind upon them, allows the metastructures to be reused and manipulated in a clearer way, independent of how the documents will be bound together. It is this second strategy that has been chosen in this project.

### 4.4 Binding Language Grammar

We here give an overview on the grammar of the BL language, its various components and their imbrications. XML has been choses as the format of choice for binding descriptions
over JSON and other alternatives. Since the language main purpose is to describe a process upon documents, it seemed quite a good option.

4.4.1 Elements

A few element types compose the grammar of BL, reflecting its design:

- `<bind>`, to represent the action of binding nodes from node sets,
- `<node>`, to represent the query/metastructuration of node sets,
- `<range>`, to represent value range of nodes of a given set,

We hereafter provide a concise description. The complete specifications are to be found in the Appendix A.

`<bind>` A BL `<bind>` element represents the definition of a binding of NV nodes; it may contain any number of BL elements, either `<bind>`, `<node>` or `<range>` elements. It does not express a sole binding, but rather the process of creating several operational binding between sets of NV nodes, according to some criteria expressing in their corresponding `<node>` element.

`<node>` A BL `<node>` element represents a set of NV nodes; it may contain any number of `<node>` elements, and must contain a single `<range>` if the nodes is to have a value, it cannot contain `<bind>` elements. A BL `<node>` element contains attributes that parametrize its various actions:

- meta-structural attributes such as `query`, `id`, which query and abstract data elements and, or the `ref` attribute that expresses a NV query (query by reference) at metastructural level.
- binding-processing attributes such as `match`, `miss`, `adapt`, `min` or `max`, to respectively express matching patterns, structural adaptation and value adaptation, as well as minima and maxima of nodes of a same type in each operational binding.
<range> A BL <range> element types the values of NV nodes; it may not contain other elements. Its attributes precise its meanings:

- id and ref to name a range and be able to reuse it by reference,
- value computing and mapping parameters, like min, max, step, ...
- the value attribute specifies that the range span be dependent on a set of nodes.

4.4.2 Example

Data Metastructure In the City Weather example, we express the metastructures of the data, the wordplot and the table through the following BL descriptions.

```xml
<node id="data" src-type="xml" src="data.xml">
  <node id="city" query-type="xpath" query="./city">
    <node id="name" query="@name">
      <range values="data.city.name" />
    </node>
    <node id="average-temperature" query="@avgt">
      <range values="data.city.average-temperature" />
    </node>
    <node id="precipitation-days" query="@precd">
      <range values="data.city.precipitation-days" />
    </node>
    <node id="latitude" query="@lat">
      <range values="data.city.latitude" />
    </node>
    <node id="longitude" query="@lng">
      <range values="data.city.longitude" />
    </node>
    <node id="selected" query-type="interaction" query="selected">
      <range />
    </node>
    <node id="hovered" query-type="interaction" query="hovered">
      <range />
    </node>
  </node>
</node>
```

Note that both the query and the source (src) are typed through query-type and src-type attributes, defining respectively what kind of query to perform, and how to get the source. They correspond to specific adapters\(^\text{10}\). Those types are inherited by nested nodes, making the description more concise.

The values attribute of a range defines the set of existing nodes whose values will compose the range. In this case, data.city.latitude represents (for the specific range

---

\(^\text{10}\)See Section 4.3 and Figure 4.1 for the role adapters play in the metastructuration process.
element) the set of values of all nodes named \textit{latitude} in all nodes named \textit{city} in the node named \textit{data}.

Interestingly, interaction properties like \textit{selected} and \textit{hovered} are also expressed as child nodes. Of course, the \texttt{query-type} of such property is not xpath anymore, since xpath does not cover DOM events, but another type called \textit{interaction}, to which corresponds an adequate adapter.

\textbf{Wordplot Metastructure} The metastructures of the wordplot and the table are built similarly, we here present only the one of the wordplot:

```xml
<node id="wordplot" src-type="svg" src="wordplot.svg" query-type="xpath">
  <node id="word" query="/svg:svg[@id='words']/svg:text">
    <range min="10" max="90" unit="\%" step="1" />
  </node>
  <node id="y" query="@y">
    <range min="10" max="90" unit="\%" step="1" />
  </node>
  <node id="size" query="@font-size">
    <range min="8" max="20" step="2" />
  </node>
  <node id="selected" query-type="interaction" query="selected">
    <range />
  </node>
  <node id="hovered" query-type="interaction" query="hovered">
    <range />
  </node>
</node>
```

Note that \texttt{range} nodes are here not specified by values queries, but by their \texttt{min}, \texttt{max}, and \texttt{step} attributes which together suffice to describe the span of all possible values. A \texttt{unit} is often necessary to express explicit dimensions in visual documents.

Here again, we added a \textit{selected} node as child of a \textit{word} node, making it selectable.

\textbf{Binding the Metastructures} Binding metastructures is about relating the metastructures of the data and the two visuals in a structure of relations. The metastructures described above in facts are elements of a root binding which simply binds the documents together:

```xml
<bind>
  <node id="data" src-type="xml" src="data.xml"/>
  <node id="wordplot" src-type="svg" src="wordplot.svg"/>
  <node id="table" src-type="html" src="table.html"/>
</bind>
```
This root binding contains child bindings, defining sub-relationships. Here it contains the binding of data cities (\texttt{data.city}) with wordplot words (\texttt{wordplot.word}) and rows of the table (\texttt{table.row}). However it is quite easy to understand that we will miss rows (the original \texttt{table.html} only contains two rows). This is a problem that we solve through a strategy called \texttt{clas}, for \textit{copy latest as sibling}. Subsection 4.5.4 details this solution.

Nested to the description of the binding of cities, words, and rows, many concise bindings express value adaptations. We detail them hereafter.

**Mapping longitude to x position**

```xml
<bind>
  <node ref="city.longitude" />
  <node ref="word.x" adapt="map('longitude')" />
</bind>
```

The \texttt{map('longitude')} expression describes the linear mapping of the value of the \textit{longitude} of the \textit{city} to the value of the current node, which is the \textit{x} position of the \textit{word}.

**Mapping city names onto wordplot words**

```xml
<bind>
  <node ref="city.name" />
  <node ref="word" adapt="value('name')" />
</bind>
```

This binding and adaptation replaces a \textit{word}’s textual value, by the \textit{name} of the bound \textit{city}.

**Distributing item properties onto row cells**

```xml
<bind>
  <node id="each" ref="city.name|city.average-temperature|city.precipitation-days" />
  <node ref="row.cell" adapt="value('each')" miss="clas" />
</bind>
```

We here use the | separator to express a combined NV query of property nodes of a \textit{city} node to a \textit{row cell} (in the table). Adaptation of the \textit{cell} is simply copying the value of the bound property node.
Selection coordination

The wild card * gathers together nodes bound in the parent binding (i.e. a row, a word and a city nodes) and binds their selected properties together. The value(latest()) tells the trio of selected properties to adapt to the latest which changed value, so that when a row, word or city is selected, the property change is reflected on to bound nodes.

![Diagram of selection coordination]

Figure 4.4: City Weather visualization, from building blocks to end result

Naming table columns

![Wordplot diagram with table.html and data.xml]

```xml
<bind>
  <node ref="data.city" />
  <node ref="table.head" />
  <bind>
    <node id="each" ref="city.name|city.average-temperature|city.precipitation-days" />
    <node ref="head.cell" adapt="id(‘each’)" miss="clas" />
  </bind>
</bind>
```
Similarly, we bind some city to the sole table head, and subsequently bind each city property node to a table head cell, into which we copy the id of the property (and not its value).

**Desired Result** The resulting BL expresion\(^{11}\) aims at a single process of transformation between data and the two visuals as pictured in Figure 4.4. Note how many aspects of a visualization like interaction coordination, value and structural mappings are expressed through a single construct reflecting our approach: *query nodes, bind and adapt them.*

### 4.5 Binding Language Semantics

A binding expression is nothing if we were to interprete it ‘manually’. The Binding Engine is an interpreter for the Binding Language. It interpretes BL expressions into live operational nodes and bindings. Such a binding structure, as depicted in Figure 4.5 provides a context of named nodes for expressing adaptation.

![Figure 4.5: City Weather, operational binding, partial schema](image)

\[^{11}\text{the full expression is to be found in Section B.1.}\]
4.5.1 Binding Process Example

The binding process starts when a BL expression is passed to the engine. The process never stops, in the sense that the binding is dynamic and responsive to changes. In the example, the binding description detailed in Section 4.4 is passed to the binding engine through a simple ecmascript method call:

```javascript
 councillors . binding ( "binding.xml" );
```

The binding engine then processes through the following steps, represented as grey-shaded zones in Figure 4.5.12.

1. The main element in the binding description is the binding of the various documents: `data.xml`, `wordplot.svg`, `table.html`. The engine fetches those documents and creates for each one an operational node, and creates adequate metastuctures for its content (Figure 4.5). Those three nodes are then bound into a single operational binding, which knows by id the nodes it binds. This binding has found for each `<node>` expression that its `<bind>` description contains, a single node, which corresponds to its minimal requirements to be complete.

2. Once this binding is complete, it tells the engine to perform sub-bindings, according to nested `<bind>` descriptions. The next `<bind>` description expresses the binding of `cities`, `table rows` and `wordplot words`. Here we have many `cities`, so each one will be attributed an operational binding. Each of this binding is eager to also bind a `row` and a `word`. But we have only two `rows` in the `table` and two `words` in the `wordplot`, so the strategy chosen (copy latest as sibling) tells the engine to duplicate `rows` and `words`, to fulfill the needs of these operational bindings.

3. Now that each `city` is bound to a `row` and a `word`, it is time to perform value adaptation and interpret all the subsequent bindings which:

- copy the `name` of a `city` into its bound `word`,
- map the `longitude` of a `city` to the `x` position of its bound `word`,
- invertedly map the `latitude` of a `city` to the `y` position of its bound `word`,
- map the number of `precipitation-days` for a `city` to the `size` its bound `word`,
- map the `name`, `precipitation-days` and `average-temperature` of a `city` to a series of `cells` in the bound `row`,
- coordinate selection and hovering of `table rows` and `wordplot words`.

Other bindings, that regard the map and interaction coordination are also performed.13

12For the sake of simplicity, the bindings to the table metastructure are not represented. Equally, not all bindings between the data and the wordplot are represented in Figure 4.5.
13Again, we omit parts of the process for the sake of simplicity.
4.5.2 Binding Process Key Problems and Aspects

The binding process possesses several key problems and aspects, some of which were encountered in the previous example:

- **matching and binding problem**: when binding nodes from two or more node sets, how should them be matched? which node should be bound to this or that precise node?

- **missing node problem**: when the table needs more rows, for example, how does the engine do? which one to copy? where to put it?

- **structural mismatch problems**: how to cope with data and visuals which do not possess similar structures? how to deal with hierarchical or tabular descriptions of graphs?

- **process flow**: what happens when a node changes its value? it surely means something to some other nodes, but which nodes? How does change flow throughout the binding structure?

- **expressing value adaptation**: to bind the latitude of a city to the y position of a word in the wordplot, does not tell which one should adapt, and how it should adapt. And once you answer those questions, how do you express those adaptation decisions? what variables are available for local computation of adaptation?

4.5.3 Matching and Binding Problem

When binding nodes from two or more node sets, how should them be matched? which node should be bound to this or that precise node?

To illustrate the problem and the solution implemented, we use a simplistic example that somewhat resembles the Africa Stats use case (Section 5.3): a set of countries in a data file is to be bound to a set of territories on a map. Countries like Japan span over several hundreds of islands, which all are territories; a typical example of a one-to-many relationship. the binding, which is part of a larger binding context, is expressed as follows:

```xml
<bind>
  <node ref="data.country" match="country.code" />
  <node ref="map.territory" match="territory.id" max="100000" />
</bind>
```
Informally: to bind a *country* to a *territory* their respective *country code* and *territory id* values should match, and many *territories* (up to 100000) may be bound to a same *country*.

Let’s start in the situation where *countries* and *territories* have been queried, and the two sets are available for binding (Figure 4.6, initial situation). A single operational binding is at first present, although binding no nodes.

Each node will try to join a suitable binding, in their order of creation. The first node, with *country code* CH which represents Switzerland, approaches the empty binding, gently saying ‘hello’. The binding sees that it has a match key with value CH. The binding currently has no match key. So this node matches the binding. Then the binding sees that it’s a *country*, and looks up into its description to see its minimal requirement for *countries*. Since the min attribute is not set, it requires per default a minimum of 1
The binding binds the country, glad to have reached its minimum number of country nodes (see Appendix A for specifications), and the country stops looking for a binding (Figure 4.6, step 1) glad of having found an operational binding.

Then the next country, Italy, looks for a binding and says ‘hello’ to the same operational binding. However, the match key of the binding (CH) and the one of the node (IT) do not match! So the binding refuses to bind the Italy node, and since no binding accepts this node, the engine creates a new binding and lets the node approach it. The binding will of course be glad to have found a country and binds it, as well as adapts its match key. The third country node, Germany (DE), is refused by all bindings because of the match key issue, so the engine creates a third binding for Germany. (Figure 4.6, step 2).

Since all countries have been bound, territories will look for bindings. The first territory, the ‘swiss territory’, says ‘hello’ to the first binding. Their match-keys are identical, and the binding still needs territories (its minimum requirement for territories is 1, the maximum is 100000), so the binding binds this territory. The match key coherently stays the same. Same thing happens with the second territory, ‘continental Italy’: it is refused by the first binding since they do not match keys, and accepted by the second since they match keys, and this binding still needed 1 territory (Figure 4.6, step 3).

Now comes a special case: the third territory, ‘Sardinia’, is refused by the first binding since they do not match keys. It then says ‘hello’ to the second binding, which sees that it has met its minimum requirement for territories and says no, although they were matching!. The third binding also refuses it because keys do not match. Having approached all bindings, the ‘Sardinia’ node visits them one more time, saying ‘please’. The first binding still refuses, insisting on the fact that their match keys are not the same. The second binding, which shares the same match key but has already enough territories, must loosen its requirements and check whether it has reached its maximum number of territories (100000). This is not the case, and it is forced to bind this node as a territory (Figure 4.6, step 4).

All nodes have successfully been bound and with respect to the previous BL expression. The local binding process is over.

Hello!Please! Algorithm The process described in the above example has been coined Hello!Please!, since nodes approach operational bindings in two rounds (when

Note that the order of binding is relative to the description, although it should not impact on the result.
necessary), the second of which sees nodes begging for binding, and bindings relaxing their requirements on amount of nodes of a same set.

### 4.5.4 Missing Node Problem

When the table needs more rows, for example, how does the engine do? which one to copy? where to put it?

In the previous example, the binding process finishes on a state where the third binding is not complete: it has not met its minimum number of territories (i.e. one). Being uncomplete, a binding is forbidden to adapt, and forbidden to deploy sub-bindings. A rather embarassing situation known as the missing node problem.

This problem is in fact two problems happening both at concrete and abstract levels: the problem of obtaining an extra node and its corresponding concrete data or visual object, and placing it, both concretely and in the metastructure.

In the current implementation, only one startegy has been developped, called \textit{clas}, for \textit{copy latest as sibling}. Concretely, when a node misses, it tells the system to copy-and-paste the last encountered node and concrete object next to itself. This is the strategy used to duplicate rows and words in the City Weather example, in order to match the number of cities.

The default strategy is the \textit{status quo}: when a node misses, no measure is taken, either because it so makes sense, or because no available measure makes sense\textsuperscript{15}.

### 4.5.5 Structural Mismatch problems

How to cope with data and visuals which do not possess similar structures? how to deal with hierarchical or tabular descriptions of graphs?

The NV Model allows to express solutions for both cases.

In the case of structural dissimilarities, structures may be made compatible by aggregation through binding, a pattern exemplified by the Bertin Hotel use case (Section 5.4).

When faced with serialized descriptions of a graph, say in the GraphML document format, it would be profitable to perform id-referencing and recreate the ‘broken’ structural relationships. Such a problem is presented and solved in the Migration Graph use case (Section 5.5).

\textsuperscript{15}Adding new measures strategies is not supported by the actual implementation
4.5.6 Process Flow

What happens when a node changes its value? It surely means something to some other nodes, but which nodes? How does change flow throughout the binding structure?

A binding generally loads some documents and their metastructure and binds those documents and their parts. It thus makes sense, for binding management to wait that documents / document-parts have been loaded before trying to bind upon them. Moreover, one may sense that this would benefit from a greedy approach: the update flow should stop when sub queries are not performed, and no part should be updated that has not been touched; something akin to a dirty/clean markup system.

The actual implementation possesses a simplistic management of change: any adding or removing of node tells the root binding to do one more general update once it’s done with the current one. Change of value is however managed locally in concerned bindings, to ensure interaction fluidity.

4.5.7 Expressing Value Adaptation

How do you express adaptation? What variables are available for local computation of adaptation?

A binding performs the adaptation of nodes it binds, a binding which knows by name only all nodes it binds, and which asks its parent binding for nodes matching the names it ignores. The range of the node to adapt takes part in the adaptation, mostly when mapping values.

The adaptation is expressed through functional expressions, benefiting from the full ecmaScript language. Some dedicated functions are provided. They are functions on node sets such as `map()`, `value()`, `id()`, `latest()`, ... More details regarding these functions can be found in Section A.4.

4.6 User Interface Layer

To fit the binding process to visualization purposes, the Metaphorea framework adds an extra layer to the Binding Engine (see Figure 4.1). This layer is quite thin in the actual implementation. It mainly provides display management of visual documents and implementation of interaction, through an adequate interaction adapter.
Section 6.6.1 details its possible enhancements.

4.7 Conclusion

A prototypal visualization framework has been developed, which, based on the Bind&Adapt approach presented in Chapter 3, implements document binding for visualization purposes. To do so, it relies on the Binding Language to express both the construction of intermediate metastructures to concrete data and visuals, the binding and the adaptation of such metastructures, reflected on their respective concrete structures. To interpret this language, a Binding Engine has been developed. The following chapter presents its application through various use cases.
Chapter 5

Use Cases

5.1 Introduction

To evaluate a software framework is a difficult task. We here propose to base our evaluation of the Metaphorea framework on some use cases which exemplify its possibilities and limits, strengths and weaknesses. This chapter presents such use cases in a systematic but colloquial way in order to exemplify the use of the BL language. At the end of this chapter, a synoptic table (Table 5.1) presents performance information about the use cases.

These use cases are available online at http://code.google.com/p/metaphorea/wiki/UseCases, and annexed to the official copies of this master thesis on a CD-ROM disk (Appendix C).

Visualization Process All use cases follow a same sequence of steps, which characterizes visualization in the Bind&Adapt approach:

1. **prepare** data and visual documents,
2. **describe metastructure** : for each document, to access and manipulate it,
3. **describe bindings** : of metastructures, and thus of documents,
4. **describe adaptation** : of both structure and value,
5. **deploy visualization** : by integrating engine in a web page and passing it the binding description resulting from the four previous steps.
5.2 City Weather, an Introductory Use Case

The introductory use case is the one of weather visualization for some European cities. Chapter 4 detailed step by step its metastructuration, bindings and adaptation. The resulting visualization is the apposition of two views: a wordplot and a table (Figure 5.1), with Brush&Link interaction between them.

The table allows sorting of rows according to the values in one of the columns. This logic pertains to the table solely (ecmascript code embedded in the original `table.html` file) and not to the binding process logic.

The coordination of selection and focus is specified through the interaction adapter, provided by the visualization layer. This layer makes interaction nodes sensitive to user actions on their corresponding concrete visual objects and changes selected and hovered property nodes accordingly. A binding is responsible for coordinating the state of such interaction nodes.

The complete BL expression of this use case can be found in Section B.1.
5.3 Africa Stats, a Mashup Use Case

In this second use case both a map, and a dataset are readily available as static documents from the net. After a slight adaptation, they are combined to provide a geographical view of some statistics concerning Africa. An extra view provides detailed statistics when the user hovers a country on the map, a typical example of ‘overview and detail’ coordination.

A complete BL description of the binding can be found in Section B.2.

![Diagram of the Africa Stats use case]

**Figure 5.2:** Africa Stats : from readily-available documents to visualization

**Document Preparation** Metaphorea is here also used to bind together data and visual pieces, and create a visualization through this binding (Figure 5.2). Data comes from the MONDIAL Database\(^1\), and the map of Africa from Wikipedia/Wikimedia\(^2\). This map has 2-letter country code to identify the shapes forming the territories. There’s however no single standard coding for countries and the MONDIAL database does it another way. We manually adapted the codes on the map to reflect the ones of the

\(^1\)[http://www.dbis.informatik.uni-goettingen.de/Mondial/]

\(^2\)[http://commons.wikimedia.org/wiki/File:Blank_Map-Africa.svg]
database. The detail view – a simple HTML document scribbled in an HTML editor – seconds the map by detailing the statistics of an hovered country.

Metastructures  The data document contains many countries each having a country code and containing some country statistics such as population-growth, inflation and infant-mortality. This unformal structure is expressed through the following BL expression:

```xml
<node id="data" src-type="xml" src="mondial.xml">
  <node id="country" query-type="xpath" query="./country">
    <node id="code" query="@car_code">
      <range />
    </node>
    <node id="name" query="./name">
      <range values="data.country.name" />
    </node>
    <node id="population" query="./population">
      <range values="data.country.population" />
    </node>
    <node id="population-growth" query="./population_growth">
      <range values="data.country.population-growth" />
    </node>
    <node id="infant-mortality" query="./infant_mortality">
      <range values="data.country.infant-mortality" />
    </node>
    <node id="total-gdp" query="./gdp_total">
      <range values="data.country.total-gdp" />
    </node>
    <node id="inflation" query="./inflation">
      <range values="data.country.inflation" />
    </node>
  </node>
</node>
```

Note that the mondial.xml contains many more statistics about countries, we however only use a handful of them. Note also that some properties like the country code are coded as XML attributes, others as XML elements in the mondial.xml file. At the metastructural level, those differences are leveraged: they all are property nodes of a country.

The metastructure of the map is much shorter, and says that the map, contains territory shapes, which have an id and an opacity as interesting properties. It also contains some heterogeneity: the territory id is either attached to a group of shapes (i.e. SVG paths), or to a shape itself. Such heterogeneity emphasizes once again the necessity of a metastructure:

```xml
<node id="map" src-type="svg" src="africa.svg">
  <node id="territory" query-type="xpath" query="//svg:g[@id='territories']/svg:path | //svg:g[@id='territories']/svg:g">
  </node>
</node>
```
The metastructure of the detailview is also concise: the *detailview* contains *detail* blocks, which have a *title* and many *properties*, each of which has a *name* and a *value*. Like *territories*, details can be hovered.

```xml
<node id="detailview" src-type="html" src="detail.html" query-type="xpath">
  <node id="detail" query="//div">
    <node id="title" query="./h3">
      <range def="text"/>
    </node>
    <node id="property" query="./p">
      <node id="name" query="./span[@class='name']">
        <range/>
      </node>
      <node id="value" query="./span[@class='value']">
        <range/>
      </node>
    </node>
    <node id="hovered" query-type="interaction" query="hovered">
      <range/>
    </node>
  </node>
</node>
```

**Binding Metastructures** The metastructure described above are in facts elements of a root binding which simply binds the data file and the map together:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bind>
  <node id="data" src-type="xml" src="mondial.xml"/>
  <node id="map" src-type="svg" src="africa.svg"/>
  <node id="detailview" src-type="html" src="detail.html" query-type="xpath">
    ...
  </node>
<bind>
  <node ref="data.country" match="value('country.code')"/>
  <node ref="map.territory" match="value('territory.id')"/>
  <node ref="detailview.detail" miss="clas"/>
  ...
</bind>
```
Each country of the data file is bound to its corresponding territory on the map and to a detail in the detailview. Note that it’s a matching problem that is solved by binding together countries and territories showing a same match-key, the country code and the territory id. Details do not have a match expression, since it only matters that each such binding include a unique detail node, not a specific one.

Note that the data file concerns all countries in the world. However, the map restricts to the african continent. Thus if no matching territory is found for a country, the country will be bound to a detail, but no territory, and this binding will never be complete.

**Mapping Statistic**  Subsequently, we map a country statistics, e.g. the infant mortality, by binding and adapting the opacity of the territory to it. The mapping is curved toward high values (to the fourth power) to accentuate visual differences.

```xml
<bind>
  <node ref="country.infant-mortality" />
  <node ref="territory.fill-opacity" adapt="map('infant-mortality','pow4')"/>
</bind>
```

The template detail view is then ‘filled’ with country data: the country name used as detail title, and each child node of the country (expressed by use of the wildcard country.*) is presented as a name-value pair:

```xml
<bind>
  <node ref="country.name" />
  <node ref="detail.title" adapt="value('name')" />
</bind>

<bind>
  <node id="attribute" ref="country.*" />
  <node ref="detail.property" miss="clas" />
  <bind>
    <node ref="property.name" adapt="id('attribute')" />
    <node ref="property.value" adapt="value('attribute')" />
  </bind>
</bind>
```

Finally, mouse hovering events are coordinated within each territory-detail binding.

```xml
<bind>
  <node ref="*.hovered" max="10000" adapt="value(latest())" />
</bind>
```
Chapter 5. Use Cases

5.4 Bertin’s Hotel, a Comparative Use Case

The third use case bases on a famous example by J. Bertin. Since it is also a use example of the Protovis Toolkit\(^3\), it allows a comparison of approach and implementation of the visualization. Protovis has been chosen as comparison for its elegant API, which may also be considered as declarative code.

A complete BL expression of the binding can be found in Section B.3.

Data & Template Visual

Visualization

![Diagram of data and template visualization]

Figure 5.3: Bertin Hotel: bind to aggregate

Document Preparation In this usecase, we used JSON formatted data from the Protovis example\(^4\) and translated it in a XML document. One more preparation was to order the data by category and month. On the visual side, a quick sketch of some bars and legend in an SVG editor provided a bar chart\(^5\) template (Figure 5.3).

Metastructures The data is a set of statistical counts for an hotel, over a single year period. Its structure is as follow: the data document contains many items each contains a count, which is a statistics of a certain category (like percentage of clients of a certain age, prices of rooms, ...) for a certain month. The metastructure for this data set naturally reflects this description:

\(^3\)http://vis.stanford.edu/protovis/  
\(^4\)http://vis.stanford.edu/protovis/ex/hotel.html  
\(^5\)We use here the term ‘barchart’ to name a ‘row of bars’ as in the Protovis example. A better name for it, according to [Harris, 1999] would be ‘column chart’, since the bars are vertical, and no scales is provided (with scales it would become a ‘column graph’, in Harris’ taxonomy).
Note the range elements, which express the value domain for category, month and count nodes. For counts, we precised a minimum value of 0, to respect the fact that count values are of the ‘ratio’ type (distances to zero).

The metrastructure for the barchart is, this time, quite heavy:
In other words: a barchart contains many rows, each having a y position and height, and containing a set of bars and a legend, with each bar having properties such x, y, height, width and class.

Note how we once define what a percent range is for the y position of a row, and later reuse this definition to describe the range of other properties of rows and bars.

**Binding Metastructures**  As usual, we bind those two documents together in a first binding expression:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bind>
  <node id="data" src-type="xml" src="data.xml">
    ...
  </node>
  <node id="barchart" src-type="svg" src="barchart.svg">
    ...
  </node>
</bind>
```

The difference in size of the metastructures of the barchart and the data hints at a new problem: the structures do not match: the data contains items with properties, the barchart contains rows which contain bars with properties, and a legend. If it weren’t for the legend, the visual form still has one more structural level than the data. This is because each row represents a category of items, whereas the data only has one set of items, categorized through the value of the cat attribute.

Inside the context of the root binding, we thus need to bundle (aggregate) data items into sets of items with same category, and bind each such set to a row:

```xml
<bind id="bundle">
  <node ref="data.item" match="value('item.category')" max="12" />
  <node ref="barchart.row" miss="clas" />
  <bind>
    <node ref="item.category" />
    <node ref="row.legend" adapt="value('category')" />
  </bind>
  <bind>
    <node ref="row.y" adapt="map(index('row','barchart.row')/size('barchart.row'))" />
    <node ref="row.height" adapt="map(1/size('barchart.row'))" />
  </bind>
  ...
</bind>
```

Note how the first line expresses aggregation by binding together between 1 and 12 items and a single row, under the constraint that items have a same category property (match attribute). Note also that missing rows are cloned on demand. Each aggregate (in facts an operational binding) is here named by the term bundle, which we will reuse later.
Subsequently, the *category* name is mapped onto the *legend* of the *row*, and the *row* *y* position and *height* are adapted to allow them to stack nicely in the available display space.

This last binding, responsible for the positioning and sizing of *rows*, pertains solely to the *barchart*. It would be pertinent to implement it in the *barchart* document itself as an embedded script, rather than through binding logic.

Now that each categorical *bundle* of *items* is bound to a *row*, we still need to perform the mapping that distributes an *item* per *bar* in a *row*, and adapts properties.

```xml
<bind>
  <node id="each-item" ref="bundle.item" />
  <node ref="row.bar" miss="clas" />
  <bind>
    <node ref="bar.width" adapt="map(1/size('row.bar'))" />
    <node ref="bar.x" adapt="map(index('bar','row.bar')*nvalue('bar.width'))" />
  </bind>
  <bind>
    <node ref="each-item.count">
      <range values="bundle.item.count" min="0" />
    </node>
    <node ref="bar.height" adapt="map('count')" />
    <node ref="bar.y" adapt="map('height','inv')" />
    <node ref="bar.class" adapt="'above'+map(lt('count',mean('bundle.item.count')))" />
  </bind>
</bind>
```

The last binding is the most interesting: it binds an *item count* (a statistics of unknown nature), the *height* of a *bar*, etc. together. The count represents sometimes percentages, sometimes 0/1 values, etc. Thus the range of values to consider for a *count* node, is not the range of all *count* values, which corresponds to the NV query `data.item.count`, but only the range of values within a category, which luckily, we just bound into a *bundle* and query by the expression `bundle.item.count`.

**Comparison to Protovis Implementation** The core of the Protovis solution is the following method-chained generative code. Generation of graphics, computation of statistics and structuration of data are mixed in a single statement:

```javascript
vis.add(pv.Panel)
  .data(pv.permute(type.entries(), order))
  .top(function() ch * (this.index + 1))
  .height(h)
  .add(pv.Label)
  .font("13px Georgia")
  .left(cw * 24 + 8)
  .bottom(0)
```
The most computationally efficient of those approaches is certainly the generative approach of Protovis. However it implies that the visual be ‘designed by numbers’ through some cryptic procedural code – where is the part concerning rows? what does the 11th line of code mean? – whereas the bind&adapt approach requires that visualization publishers prepare visuals the way they prepare data: as documents edited with ad hoc applications, and that the same publishers express the logic that relates data and visuals as a metaphor-like structure.

5.5 Migration Graph Use Case

In this use case, we represent migration flows between US counties, based on data taken from the Census 2000.\footnote{http://www.census.gov/population/www/cen2000/ctytoctyflow/index.html}

A complete BL expression of the binding can be found in Section B.4.
Document Preparation  Migration data was previously formatted in a GraphML document \(^7\) by C. Weiwei\(^8\). This use case only uses a subgraph of the original graph. On the visual side, as for the Bertin Hotel use case, a quick sketch of a dot (circle) and a line in an SVG editor provided an almost good enough template: the graphic.svg is refined to contain a red-to-green gradient that is used to color the line and express direction of a line. Gradients in SVG are ill-conceived, with respect to our needs, and dot not follow rotation/orientation of an object. We thus add a small script to the graphic that adapts the gradient according to a line orientation.

Metastructures  The migrations GraphML document contains US counties (with name and \(x\) and \(y\) coordinates) and flows of migrants from a source county to a target county (referenced by \(id\)) and a certain weight (number of migrants).

\[
\begin{align*}
\text{<node id="migrations" query-type="xpath" src-type="xml" src="migrations-5000.xml" >} \\
\text{<node id="county" query="./graph/node" >} \\
\text{<node id="id" query="@id" >} \\
\text{<range />} \\
\text{</node>} \\
\text{<node id="x" query="./data[@key='x']" >} \\
\text{<range values="migrations.county.x" />} \\
\text{</node>} \\
\text{<node id="y" query="./data[@key='y']" >} \\
\text{<range values="migrations.county.y" />} \\
\text{</node>} \\
\text{<node id="name" query="./data[@key='tooltip']" >} \\
\text{<range />} \\
\text{</node>} \\
\text{</node>} \\
\text{<node id="flow" query="./graph/edge" >} \\
\text{<node id="id" query="@id" >} \\
\text{<range />} \\
\text{</node>} \\
\text{<node id="weight" query="./data[@key='value']" >} \\
\text{<range values="migrations.flow.weight" />} \\
\text{</node>} \\
\text{<node id="source-id" query="@source" >} \\
\text{<range />} \\
\text{</node>} \\
\text{<node id="target-id" query="@target" >} \\
\text{<range />} \\
\text{</node>} \\
\text{<node id="source" ref="migrations.county[id={value('source-id')}]">} \\
\text{<node id="target" ref="migrations.county[id={value('target-id')}]">} \\
\text{</node>} \\
\text{</node>}
\end{align*}
\]
Note the last lines, which link source and target counties of a flow as property nodes of this flow through a filtered NV query which performs the deserialization of id-ref cross-references.

The metastructure of the graphic document strongly reflect the one of the data: the graphic contains dots and lines. Each dot possesses planar x and y coordinates (as well as a certain size and a title). A line is expressed by its start (x1, y1) and end (x2, y2) coordinates, as well as its width.

```
<node id="graphic" src-type="svg" src="graphic.svg" query-type="xpath">
    <node id="dot" query="/svg:svg[@id='dots']/svg:circle">
        <range id="percent" min="10" max="90" unit="%" step="0.01"/>
    </node>
    <node id="x" query="@cx">
        <range def="percent"/>
    </node>
    <node id="y" query="@cy">
        <range def="percent"/>
    </node>
    <node id="size" query="@r">
        <range min="8" max="20" unit="" step="2"/>
    </node>
    <node id="title" query="/svg:title">
        <range/>
    </node>
</node>

<node id="line" query="/svg:svg[@id='lines']/svg:line">
    <node id="x1" query="@x1">
        <range def="percent"/>
    </node>
    <node id="y1" query="@y1">
        <range def="percent"/>
    </node>
    <node id="x2" query="@x2">
        <range def="percent"/>
    </node>
    <node id="y2" query="@y2">
        <range def="percent"/>
    </node>
    <node id="width" query="@stroke-width">
        <range min="1" max="16" step="1"/>
    </node>
</node>
```

**Binding Metastructures** The metastructures described above in facts are elements of a root binding which simply binds the migrations and the graphic files together:

```
<?xml version="1.0" encoding="UTF-8"?>
<bind>
    <node id="graph" query-type="xpath" src-type="xml" src="migrations-5000.xml">
        ...
    </node>
    <node id="graphic" src-type="svg" src="graphic.svg" query-type="xpath">
        ...
    </node>
</bind>
```
This root binding, has two nested bindings. the first binds to each flow a line on the graphic. The second binds to each county a dot on the graphic.

Mapping Properties  We still need to adapt properties locally to these bindings. For counties and dots, we map the name of the county onto the title of the dot, and the x and y positions of the county onto the ones of the dot:

Regarding flows and lines, we map the flow weight onto the line width, and source and target x and y coordinates of a flow, onto, respectively, a line's x1, y1, x2, y2.

Note that we did not bind line width to flow weight before adapting it: we only included in this binding the nodes that needed to adapt and slightly modified the adaptation expression. If we really had wanted to bind all pairs of visual and data nodes before adapting them, we would have expressed it through the more verbous BL expression:
This ‘bloated’ BL expression performs the same as the previous one, but may be more robust in some occasions: one of the five bindings may fail without blocking the others. In the more concise expression, if an edge were to not contain weight information, the whole binding would not complete and thus, not adapt (see ‘missing node problem’, Subsection 4.5.4).

### 5.6 Performance

The Table 5.1 presents performance measures regarding these use cases.

The tests were done on a MacBookPro with an Intel Core 2 Duo 2.5Ghz processor, 4GB of random access memory, with Mac Os X 10.6.2 operating system, and the following browsers: Firefox 3.5.6 and Safari 4.0.4. The files were served from a MAMP server running on the same machine. Each test was reconducted 10 times.

The following measures are presented:

- **dataset size**: amount of data items, also visual items when relevant.

- **load**: average load time for the loading of the binding document as well as data and visual sources. The framework load time has not been included since it is constant: about 90ms for a total file size of about 120KB, uncompressed.

- **binding process**: average time of process until first stabilization before any interaction. This includes the load time.
### Table 5.1: Use Case Performance Tests, Safari/Firefox Browser

<table>
<thead>
<tr>
<th>use case</th>
<th>dataset size</th>
<th>load</th>
<th>binding process</th>
<th>interaction response</th>
<th>memory footprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>City Weather</td>
<td>20 cities</td>
<td>0.072s/0.11s</td>
<td>0.21s/0.90s</td>
<td>&lt; 0.2s (both)</td>
<td>14MB/8MB</td>
</tr>
<tr>
<td>Africa Stats</td>
<td>238 countries, 50 territories</td>
<td>3.5s/0.17s</td>
<td>3.5s/15s</td>
<td>&lt; 0.2s (both)</td>
<td>190MB/210MB</td>
</tr>
<tr>
<td>Bertin Hotel</td>
<td>144 items</td>
<td>1.2s/0.11s</td>
<td>1.7s/7.3s</td>
<td>no interaction</td>
<td>86MB/130MB</td>
</tr>
<tr>
<td>Migration Graph</td>
<td>251 nodes, 428 edges</td>
<td>6.7s/0.13s</td>
<td>15s/76s</td>
<td>no interaction</td>
<td>230MB/180MB</td>
</tr>
</tbody>
</table>

- **interaction response**: average time for the coordination of interactive actions. 0.2s is here considered as the threshold between instataneous and noticeable interaction time\(^{14}\).

- **memory footprint**: average amount of memory the browser dedicates to this visualization (base memory needs of the browser application are not included).

### 5.7 Conclusion

Although the presented use cases compose a limited set, they provide a good overview of practical uses of the Bind&Approach and the Metaphorea Framework and they various problems that arise, as well as the solutions that the approach and the framework provide for them.

\(^{14}\)Studies such as [Pattison and Phillips, 2001] recommend the stricter threshold of 0.1s.
Chapter 6

Discussion and Improvements

“A graphic is never an end in itself, it is a moment in the process of decision making.”

6.1 Discussion

Based on the previous use cases, we here propose to discuss the framework as well as the approach it bases upon, through the analysis of the BL language, the conceptual, data and visual models developed, and the ability of the language to address visualization aspects, such as aggregation, representation, interaction and coordination. Performance and complexity issues are further outlined. We pursue with a discussion of the impacts on visualization practices, its tasks and actors.

The last section presents how some improvements will be essential to the fulfillment of the approach.

Table 6.1 summarizes the advantages, shortcomings and possible enhancements of the Bind&Adapt Approach and the Metaphorea framework.

6.2 Language

6.2.1 Metastructural Description

The BL grammar provides means to describe structures of data and visual objects of various formats. Although it is a verbous task, implying the knowledge of some query
Table 6.1: Advantages, Shortcomings and Possible Enhancements of the Bind&Adapt Approach and the Metaphorea Framework

<table>
<thead>
<tr>
<th>aspect</th>
<th>(+) advantages, (–) shortcomings and (!) needed improvements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binding Language</td>
<td>+ powerful and progressive abstraction of data and visuals,</td>
</tr>
<tr>
<td></td>
<td>+ concise reference by NV query language,</td>
</tr>
<tr>
<td></td>
<td>+ local scope for adaptation,</td>
</tr>
<tr>
<td></td>
<td>+ NV query language able of aggregation, filtering, union, ...</td>
</tr>
<tr>
<td></td>
<td>– manual metastructuration,</td>
</tr>
<tr>
<td></td>
<td>– heteroclite typing (range) system,</td>
</tr>
<tr>
<td></td>
<td>! adapters to assist metastucture building, adapter to propose computational nodes.</td>
</tr>
<tr>
<td>Models</td>
<td>+ unified and flexible data and visual model,</td>
</tr>
<tr>
<td></td>
<td>+ leverages disparities at concrete level, unified manipulation scheme,</td>
</tr>
<tr>
<td></td>
<td>– homonymy and synonymy problems,</td>
</tr>
<tr>
<td></td>
<td>! develop a model of change flow,</td>
</tr>
<tr>
<td></td>
<td>! address failed binding problem,</td>
</tr>
<tr>
<td></td>
<td>! extend the model for (visual) composition,</td>
</tr>
<tr>
<td></td>
<td>! evaluate acception of conceptual model,</td>
</tr>
<tr>
<td></td>
<td>! strengthen the models by deeper study.</td>
</tr>
<tr>
<td>Performance</td>
<td>+ the system works ...</td>
</tr>
<tr>
<td></td>
<td>– sluggishly: too few internal optimization,</td>
</tr>
<tr>
<td></td>
<td>– some redundancy in implementation,</td>
</tr>
<tr>
<td></td>
<td>! optimize engine based on change flow model,</td>
</tr>
<tr>
<td></td>
<td>! implement memoization/caching and simplification of constructs.</td>
</tr>
<tr>
<td>Expressiveness</td>
<td>+ same clear-case construct to express visualization aspects as mapping, aggregation, and interaction coordination,</td>
</tr>
<tr>
<td></td>
<td>– guides and scales are not natural to the approach,</td>
</tr>
<tr>
<td></td>
<td>! research view composition and recursive mapping,</td>
</tr>
<tr>
<td></td>
<td>! clarify interaction coordination,</td>
</tr>
<tr>
<td></td>
<td>! address animation aspects,</td>
</tr>
<tr>
<td></td>
<td>! develop an efficient solution for guides and scales.</td>
</tr>
<tr>
<td>Visualization</td>
<td>+ based on refining relationships,</td>
</tr>
<tr>
<td>Practices</td>
<td>+ iterative refinement at the heart of the system,</td>
</tr>
<tr>
<td></td>
<td>+ clear separation of data preparation and visual design,</td>
</tr>
<tr>
<td></td>
<td>! direct manipulation tools,</td>
</tr>
<tr>
<td></td>
<td>! visualization-oriented SVG and HTML adapter,</td>
</tr>
<tr>
<td></td>
<td>! usability testing at language, framework and interface level.</td>
</tr>
</tbody>
</table>

language in regard to the document format (e.g. XPath to query XML), it needs to be done only once for documents presenting a same structure and type, as is typically the case for subsets of a same data corpus).

The metastructures it describes are able to deal with the three data archetypes that are relations, graphs and hierarchies, as hinted by the use cases.

The adapter construct (Figure 4.1) relieves query tasks from the core binding process. This will also permit to add adapters to new document formats (KML, VML), or even non-XML formats like CSV, JSON, etc.
The meta-structural description is still a manual task, one may however imagine that it can be aided by direct manipulation, in the user interface, of existing bindings and meta-structures representations (further discussed in Section 6.6.1).

6.2.2 Binding Description

The BL grammar provides means to relate sets of metastructure nodes, and thus relate document parts.

These bindings are easy to express thanks to the fact that the nodes they target are queryable through NV query notation. This query language, although young, possesses features like filtering and union of queries. It however could benefit from further developments, to reach the expressiveness of the XPath language.

The canonical issues that are the matching and grouping and the missing node problems that arise when relating structures have been identified and addressed through efficient strategies and algorithms. Both problems certainly need a deeper examination, the missing-node problem being at the frontier between the binding and the generative approaches, and the matching and grouping problem certainly hides other necessities, that the use cases did not cover so far.

6.2.3 Adaptation Description

Value mapping is described in functional terms following the ecmascript scripting grammar. One may regret this design decision, which implies some programmatical knowledge on the part of the person describing the value mapping. However, it is a level of programmatical fluency comparable to the one needed to use statistical functions in a spreadsheet editor, and which be aided by the system. Moreover, value mapping often implies computation of counts, ratios, means, and many other statistics; describing them in an XML grammar rather than through a functional expression would by no mean be an advantage.

It is an open question whether these statistics should be represented as special nodes, i.e. local ‘variable nodes’ with computed values, something that surely would increase efficiency and simplify value adaptation expressions. The implementation of such a construct would in facts be trivial thanks to the adapter construct; only time restrained us from exploring this idea.

This once more shows the extent of the approach, which emphasizes on describing the structures and bindings and doing so eases up the expression of value computation.
The value-typing decision that has come to the definition of the construct of value range, more specifically implying that values are considered in BL as zero- or one-dimensional data is also questionable. The use cases however emphasize the fact that multi-dimensional values are not omnipresent and belong mainly to the domains of color and time. One should also consider that a data corpus containing property types with multi-dimensional values is ill-defined and should be refined. To cope with such multi-dimensional values, one may provide an adequate adapter (e.g. targeting saturation, hue and brightness as distinct property nodes of an SVG element).

Moreover, a range targets both typing and domain mapping needs in a mixed way and at a weird position: attached to a node to type it, it influences other nodes, elsewhere in the binding structure. The two features that are typing and domain mapping may well need to be expressed separately. On the other side, ranges are quite powerful, comprehensible and flexible constructs.

6.3 Models

6.3.1 Conceptual Model

The conceptual model, close to the one of a metaphor, is, according to author’s beliefs, one of the main quality of the framework. This assertion is yet to be proven. How, is yet another question.

6.3.2 Data Model

Metaphorea is data flexible (according to the term in [North, 2000]): it can adapt to the main models of data. However, its working model is a directed graph, a quite uncommon model for general-purpose visualization toolkits which tend to have relational data model as working model.

Although our working model is of a complex and reticular nature, it is, when focusing on a binding and its bound nodes, or a node and its connections, very simple to grasp, and emphasizes node relationships thanks to named edges.

Local synonymy and homonymy are problems that may happen and that have been considered, and avoided by attention to it. In the Bertin Hotel use case (Section 5.4), categorical sets by items categories have been named bundle and not category, to avoid homonymy; and items are renamed (through specific ids) through the binding levels, implying synonymy.
6.3.3 Visual Model

Metaphorea’s visual model is the same as the data model. This particularity makes Metaphorea strongly flexible in terms of visual forms usage and creation. Encouraging at all levels simple structures of the visual form and expressing this structure in the metastructural definition facilitates the understanding of the visual structure, and may well act as a learning factor.

An area that has regretfully not been approached in this project, is the compositing of views coming from various sources. They are, in the actual implementation, simply posed one next to the other, and not laid out in a proper interface design. It is an open question wether layout and embedding of visuals is expressible through extension of the BL language, or wether it will conflict with it.

6.3.4 Process Model

The actual development presents lackings in the rigorous definition of the semantics of a binding expression. And, although it enforces the rules of binding repeatedly and until the system stabilizes, it fails to provide an optimized model for events and actions of the global process, one that could keep the computational costs in a linear order, something that would allow the framework to scale up gracefully.

6.4 Framework Performances and Complexity

6.4.1 Data Complexity

In a visualization system, it is very unlikely that the number of visual elements in a single view outpasses the number of data elements that this view represents. We may thus consider that the number of elements in a visualization is proportional to the number of data elements factored by to the number of views plus one (for data). Wether the data and the views are strongly hierarchal or not does not change this figure.

Moreover, since it is unlikely that a same node be included in more than one binding, and very unlikely that it is bound more than a couple of times, we may consider this maximum amount of bindings per node to be finite and below 3 in the great majority of cases. So the complexity of the data structure is linear in regard to the number of elements, the number of views and the number of nodes per binding.
6.4.2 Process Complexity

Process complexity, however, seems to be of a different order. The binding process, although naturally ordered, tends to favor local scope. The implemented Binding Engine runs costly updates until the process stabilizes, and many operations are yet unoptimized. For example, at each global update iteration, values are recollected, statistics and ranges recomputed, although most of them remain the same after a few iterations.

This cost can typically be avoided by memoizing/caching values, value sets and their statistics, with an appropriate mechanism to decide when to update the cache, in connection to a change flow model to come.

6.4.3 Performances

The behaviour resulting of these costly recomputations, coupled with the poor response of ecmaScript engines to heavy computations, results in browser crashes and framework sluggishness. However, better performances can be expected, the overall binding process showing no sign of intractability.

6.5 Expressiveness

6.5.1 Expressing Mapping: Bind to Represent

Generative toolkits which base on procedural, object-oriented, or functional programming languages are of course able of expressing any computation. Their usage however does not reflect much of the corresponding operations at the level of the functional model\(^1\). In terms of describing the mapping, they are black boxes. The binding model, on the opposite, is of limited expression power but is a transparent box, in which the expression of the relationships between data and form are clearly described.

The BL expression of a mapping is most elegant, reduced to declaring which node’s value (in the context of its range) is to be mapped into which range, and set as the value of which variable. Example:

```
<bind>
  <node ref="item.longitude" />
  <node ref="word.x" adapt="map('longitude')" />
</bind>
```

\(^1\)see the Protovis code in Bertin Hotel use case (Section 5.4)
Recursive Mapping  Recursive mapping is a problem that one faces when using visual forms with internal recursive structure such as treemaps. In BL terms, it implies recursive descriptions of the data and visual metastructures and of the binding itself.

In this regard, expressing a recursive mapping is more cumbersome in BL than it is in procedural terms, where generating the visual form and mapping values compose a single recursion loop.

The Failed Binding Problem and Other Phantoms  The Africa Stats use case (Section 5.3) hints at the fact that some bindings will never be completed: only african countries will find a matching territory on the map of Africa. This non-completion of the binding means something to the country, something like ‘get lost!’, ‘quiet down...’ or other such subtilities. We here coin this as the failed binding problem. A problem which needs deeper study, and which shows the importance of stronger study of the NV Model and the need for new use cases to face new such problems.

6.5.2  Expressing Manipulation: Bind to Aggregate, Filter, etc.

The unified data and visual model, the NV Model, implies a unified language to query data and visual structures (NV query), and compute upon them (NV script).

The binding sets do not necessarily contain single elements of each type, but rather sets of nodes of a same type. This allows for aggregation/partition both on the visual and the data side, and for the mapping of data statistics onto visual properties.

Aggregation is a natural function of the binding construct. The Bertin Hotel use case exemplifies how the partioning of a data set is performed through a single binding (Section 5.4).

Filtering is also addressed and supported by the NV query language, as demonstrated in the Migration Graph use case (Section 5.5), as is the union of queries, demonstrated in the City Weather use case (Section 5.2).

6.5.3  Expressing Coordination: Bind to Coordinate

A strong advantage of the binding paradigm relies on the maintenance of live bounds and contexts (operational bindings), on which coordination and adaptation of data and visual parts are operated. In the City Weather use case (Section 5.2), selection and mouse hovering actions on the table and the wordplot are coordinated two expression types:
Embedded into the metastructural description of a table row, a city in the data or a word on the wordplot, the selected node renders them selectable. The second expression which binds the selected nodes of bound city, row and word together expresses interaction coordination in an effective way:

```
<bind>
  <node ref="*.selected" max="3" adapt="value(latest())" />
</bind>
```

The logic behind is a simple interaction adapter, suitable for any XML document, that renders the NV nodes sensitive to interaction on the document elements they abstract. This implementation of the coordination of views, so effective that it is, is somewhat of a tradeoff: it bases on an attribute, shared by many but not all web document types, the class attribute. This model is still to be developed and the question remains whether interaction should be adapted format-wise and added as an extra requirement on adapters, or not.

### 6.5.4 Expressing Animation

In the perspective that animation belongs to the visual realm, animation is the sole responsibility of the visual form in our approach and should not be implemented through BL expressions. However, the visual form has to allow one to parametrize animation through ‘adapting some values of some nodes’, the same way one sets a selected or a size value.

Animation seems a difficult task in the Bind&Adapt approach, as it is with the others, although no use case has been developed to study it.

### 6.5.5 Providing Guides & Scales

Our use cases present almost no legends or scales. Providing visual guides is tedious through the Bind&Adapt approach: one is willing to describe the relation between data and visual forms, but expects that visual hints providing the context of this relation and this relation itself be automatically added. Interestingly, a binding expression contains quite a few informations pertinent to legends and guides, as value mapping happens in the context of value ranges, nodes are named, ... This is yet another area of development.
6.5.6 Controling Details

The amount of detail in a Bind&Adapt visualization can be controlled through various mechanisms. The step parameter of a range can be large so to allow value ‘rounding’ (aggregation) or small for precise value depiction and comparison. For example, the first of the following ranges allows the six values 0%, 20%, 40%, 60%, 80%, and 100%, while the latter ranges from 0% to 100% by increments of 1.

\[
\begin{align*}
&\text{<range min="0" max="100" step="20" unit="\%" />} \\
&\text{<range min="0" max="100" step="1" unit="\%" />} \\
\end{align*}
\]

Another mean of adding detail, is to refine a binding by subsequent bindings which express the action of drilling down to smaller node sets, thus allocating more of the visual channel for their exploration.

6.6 Focus on Visualization Practices

The Bind&Adapt approach has been developed with visualization purposes as main target, but with the parallel interest of making the Binding Language a language for document transformation. Although this surely is an interesting research lead per se, the Metaphorea Framework needs to be refocused on the sole purpose of visualization, now that the Binding Engine and process are modeled and implemented. The following issues deserve, in this regard, particular attention.

6.6.1 Bind to Visualize and Explore through Direct Manipulation

The Bind&Adapt approach changes the visualization process, particularly its mapping part. However, in its actual incarnation, it empowers the publisher and not the consumer. We here express how further improvements of the approach and the framework may shift the building of visualizations through our approach on the consumer side, and blur the roles of publisher and consumer.

The visualization publisher stands in the middle of data and visual forms, an ideal posture to facilitate the understanding and control of the functional model. This stance should also be the one of the visualization consumer. Although the approach bases on simple concepts, the framework lacks a tool to build and extend visualization solution through direct manipulation rather than through the manually writing of BL expressions, in two words: a graphical publisher interface.
Binding and adapting by direct manipulation would allow easier refinement of the visualization by the publisher and the consumer – merging the two practices – to add a view, change and refine mappings, ... and thus extend the visualization to further explore the data, without asking the publisher for such extensions. Previous approaches exist, such as VisTree [Rodrigues et al., 2005], which shall be of great guidance in this research direction.

View compositions is also highly desirable, in a component-based manner. However, unlike the component-based approach, the composited view should remain open structures and their logics should only target visual issues and follow the separation of logic principle.

6.6.2 SVG-Viz Adapter

The combination of the NV Model and the adapter construct constitutes a powerful pair that can be oriented toward visualization. The actual XML adapter is in facts responsible for handling any XML-based document type, including SVG. The possibility to target color space through separation of the hue, saturation and brightness ranges and nodes has been presented as a feasible construct of the NV Model (Section 6.2.3). Extending this view, a complete SVG-Viz adapter could unify manipulation of SVG visuals through a model which would base on the constructs of visual cognition, such as retinal variables [Bertin, 1981, Ware, 2008].

6.6.3 The Approach and the Market

Based on our experience on the use cases we presented, the approach and framework seem adequate for rapid prototyping purposes: much data preparation can be overlooked (the system can cope with dirty semi-structured data), visuals can be edited in vector-graphics or hypertext editors, no programming is required, making the framework adequate to web mashup practices.

The proposed extensions should open it to a broader range of practioners. A particularly interesting population is graphic designers and web publishers, most of which are fluent with declarative languages, manipulating XML-based layered visual documents, etc. and which often lack the programming skills to deploy other visualization solutions. These populations versed in the visual and hyperlinked culture should be particularly receptive to the approach. The Metaphorea framework has good chances answer their needs for a highly-customizable and compositable, refinable, stylable visualization framework.
Chapter 7

Conclusion

The field of visualization as many other fields related to information manipulation, cognition and capitalization is bursting nowadays, even more ‘on the Web’. This burst of innovation will see many tools and ideas come and go. Strong emphasis should be put on bringing visualization to mass consumption and practice, both requiring vernacular conceptual models and idioms and focus on heterogeneous semi-structured data. This situation constitutes an opportunity to question the generative approach and propose alternatives.

This thesis proposes an approach to visualization as a structured process of binding and adapting data and visual elements; a structured process to be expressed in a declarative form and/or direct manipulation of data and visual structures and their relationships.

It contrasts with a generative approach which ‘boils’ data into visual forms, and doing so often breaks the links between them. Our approach also differs from a component-based approach in the sense that it considers data and visuals symmetrically and in an open manner, whereas the component-based approach composites a visual system and feeds data to it.

It bases on the principle of clear distinction between the three ‘realms’ of data, visual forms, and the data-visual relation, and advocates their parity, allowing a unified model of access and transformation.

A prototypal visualization framework has been developed, based on the Bind&Adapt approach, which applies document binding for visualization purposes. To do so, it relies on the Binding Language to express both the construction of intermediate metastructures (following the NV Model) to concrete data and visuals, the binding and the adaptation of such metastructures (referenced with NV queries, value computation through NV
script), reflected on their respective concrete structures. To interpret these languages, a Binding Engine has been implemented.

Four use cases have been developed, which provide a good overview of practical uses of the Bind&Approach, the Metaphorea framework and the various problems that arise, as well as the solutions that the approach and the framework provide for them. These use cases also compose a practical introduction to the approach.

Although both the approach and the framework lack maturity (which would at least require optimization, thorough study of models and a refined type system), they provide a conceptual and technical framework addressing contemporary issues in visualization like data manipulation (filtering, aggregation, partitioning), interaction coordination, and exploration in an innovative manner. The expression of visual mappings through the BL language turns out to be quite elegant, an elegance which most of the time is reflected in the simple value adaptation expressions, as well as the conciseness of the binding description. This elegance, which reflects onto NV query expressions for access to both data and visual structures, may well be the lead to follow to implement direct manipulation of bindings and structures.

Some unexplored issues like the generation of guides and scales, recursive structure mapping or expressing animation will reveal new problems, new strengths and weaknesses of the approach.

Stepping away from the technique, the author must admit that little evaluation has been done, except for use case analysis and performance tests. Usability evaluation of the approach, its conceptual models and the framework are strongly advocated.
Appendix A

Specifications of the Binding Language

A.1 Introduction

The Binding Language (BL) is used to describe high-level metastructure for documents and describe the bindings between those documents/document parts in terms of bindings between those metastructures as pictured in Figure 4.2.

A few concepts are present in the Bind&Adapt approach, namely binding, node and range. Each of those concept is represented in BL by an eponymous XML element type: <bind>, <node>, <range>.

A BL description is the expression of a binding, written in the BL language, and generally saved as an XML document. The BL document type definition (DTD) is to be found in Section A.3.

The Binding Language is still young and subject to change. These specifications represent the language interpreted by the framework as in its version 0.2, we will thus coin it BL 0.2.

A.2 BL Elements

A.2.1 <bind> elements

A BL <bind> element defines the action of binding nodes from node sets. It does not express a single binding, but rather the process of creating several operational bindings
between nodes of given sets. The semantics of such `<bind>` element is precised by its child `node` elements. Its attributes allow the reuse of bind constructs, either defined in the same BL document or externally.

**Attributes:**

**id** the id/alias of a bind element
- `optional`, `default value: ''`, `value type: name token`

**def** the definition of a bind element
- `optional`, `default value: ''`, `value type: name token`

**hdef** the URI of the definition of a bind element
- `optional`, `default value: ''`, `value type: URI`

**Content Model:** any number of `<bind>` or `<node>` elements, regardless of their order.

**Additional Constraints:**

- the value of the `def` attribute must correspond to the value of the `id` attribute of another bind element in the BL document, which is not a descendant of this bind element.

- the `def` and `hdef` attributes are exclusive: a `<bind>` element may not contain both attributes.

### A.2.2 `<node>` elements

A BL `<node>` element defines a set of nodes. This set is either built by metatstructuration (query on the concrete documents), or by reference (NV queries). Its attributes either concern definition expression and reuse: `id`, `def` and `hdef`, or metastructuration: `id`, `ref`, `query`, `query-type`, `src`, `src-type`, or the binding process: `match`, `min`, `max`, `miss`, `adapt`.

**Attributes:**

**id** the id/alias of the queried nodes
- `required`, `value type: name token`

**def** the definition of a node element
- `optional`, `default value: ''`, `value type: name token`
Appendix A. Specifications of the Binding Language

**hdef** the URI of the definition of a node element
  
  *optional, default value: ‘’, value type: URL*

**ref** the dot-notation query of NV nodes
  
  *optional, default value: ‘’, value type: scriptable NV query*

**query** the query expression, a lookup for concrete document elements
  
  *optional, default value: ‘’, value type: scriptable string

**query-type** the type of the query expression
  
  *cascading, default value: ‘xpath’, value type: string*

**src** the source document location
  
  *optional, default value: ‘’, value type: URL*

**query-type** the type of the source document
  
  *optional, default value: ‘xml’, value type: string*

**match** the matching key
  
  *optional, default value: ‘’, value type: ecmascript expression*

**miss** the strategy to handle missing nodes
  
  *optional, default value: ‘’, value type: name token*

**min** the minimum number of nodes of this set for a binding to succeed
  
  *optional, default value: 1, value type: integer*

**max** the maximum number of nodes of this set for a binding to succeed
  
  *optional, default value: 1, value type: integer*

**adapt** the adaptation value for nodes of this set
  
  *optional, default value: ‘’, value type: NV script*

**Content Model:** any number of `<node>` elements, regardless of their order; one `<range>` element.

**Additional Constraints:**

- the value of the **def** attribute must correspond to the value of the **id** attribute of another node element in the BL document, which is not a descendant of this node element.

- the **def** and **hdef** attributes are exclusive: a node element may not contain both attributes.

- the **min** attribute value must be smaller or equal to the **max** attribute value.

---

1 see Section A.4 for more on scripting attribute values
A.2.3  <range> elements

A BL <range> element precises the value type of the nodes corresponding to its parent
<node> element. It may contain optional attributes to precise its meaning. A range
is a finite ordered set of textual or numerical values. Implicit orders are alphabetical,
respectively real numbers’ order.

Attributes:

id  the id/alias of the range element
    required, value type: name token

def  the definition of a range element
    optional, default value:'', value type: name token

hdef  the URI of the definition of a range element
    optional, default value:'', value type: URL

values  the dot-notation query of NV node values
    optional, default value:'', value type: NV query

order  the ordering type
    optional, default value:’’ — ‘num’ — ‘alpha’

min  the minimum value of the range
    optional, default value:0, value type: numerical

max  the maximum value of the range
    optional, default value:1, value type: numerical

step  the step size for the range
    optional, default value:1, value type: numerical

unit  the unit of the range values
    optional, default value:’’, value type: string

Content Model:  may not contain other elements.

Constraints:

• the value of the def attribute must correspond to the value of the id attribute
  of another range element in the BL document, which is not a descendant of this
  range element.

• the def and hdef attributes are exclusive: a node element may not contain both
  attributes.

• the min attribute value must be smaller or equal to the max attribute value.
A.3 Binding Language DTD

```xml
<!DOCTYPE bl SYSTEM "bl-0.2.dtd">
<!ELEMENT bind (bind|node)* >
<!ATTLIST bind
  id CDATA #IMPLIED
  def CDATA #IMPLIED
  hdef CDATA #IMPLIED
>
<!ELEMENT node (node*, range)>  
<!ATTLIST node
  id CDATA #IMPLIED
  def CDATA #IMPLIED
  hdef CDATA #IMPLIED
  query CDATA #IMPLIED
  query-type CDATA #IMPLIED
  src CDATA #IMPLIED
  src-type CDATA #IMPLIED
  ref CDATA #IMPLIED
  match CDATA #IMPLIED
  miss CDATA #IMPLIED
  min CDATA #IMPLIED
  max CDATA #IMPLIED
  adapt CDATA #IMPLIED
>
<!ELEMENT range EMPTY>
<!ATTLIST node
  id CDATA #IMPLIED
  def CDATA #IMPLIED
  hdef CDATA #IMPLIED
  values CDATA #IMPLIED
  min CDATA #IMPLIED
  max CDATA #IMPLIED
  adapt CDATA #IMPLIED
  step CDATA #IMPLIED
  unit CDATA #IMPLIED
>
```

A.4 NV script Specifications

BL descriptions need script expressions to express adaptation and allow dynamic parameters (see Section 4.4).

Complete ecmaScript language is available for such scripting needs. However most of the times, a handful of functions suffices. Those are the functions that compose a library
Appendix A. Specifications of the Binding Language

named NV script. Most of them take as first (and often sole) parameter a NV query expressing a set of NV nodes (setQuery), or the first element of this set (nodeQuery)².

**size(setQuery)** returns the number of nodes in queried set.

**id(setQuery)** returns the id of the first node in queried set.

**index(nodeQuery,setQuery)** returns the position of the queried single node in the queried set.

**latest()** returns the latest node that was added or had its value changed in the current scope.

**lt(v1,v2)** ‘less than’ value comparison of values v1 and v2.

**map(nodeQuery,options)** maps the value of first node in the queried set onto the range of the node to adapt. Options inv, log, exp, pow2, pow3, pow4, pow6, pow8, transforms de facto linear mapping, respectively as logarithmic, exponential, or some power mapping.

**mean(setQuery)** returns the mean value of nodes in queried set.

**nvalue(nodeQuery)** returns the normed value (between 0 and 1) of the first node in queried set.

**value(nodeQuery)** returns the value (between 0 and 1) of the first node in queried set.

### A.5 Scripting Attributes in BL Expressions

The necessity to dynamically define parameters of the binding process has appeared in a few use cases. This was solutionned by allowing some attributes to be scriptable, like the ref attribute of the <node> element. A dynamical part can be added to the string value of such attributes by inserting it in brackets. Example³:

```xml
<nodex id="source" ref="migrations.county[id={value('source-id')}]"/>
```

In the context of a node which has a bound node named source-id with value 15, the value of the ref attribute will interpret to: migrations.county[id='15'].

---

²the distinction between queries of a single node and queries of many nodes is purely on the user level, any query will always return a set of nodes, and some functions consider only the first of those nodes

³from the Migration Graph use case, Section 5.5.
Appendix B

Use Case Bindings

The full BL expressions for each use case are listed here, for reference.

B.1 City Weather Binding

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bind>
  <!-- metastructures -->
  <node id="data" src-type="xml" src="data.xml">
    <node id="city" query-type="xpath" query="/city">
      <node id="name" query="@name">
        <range values="data.city.name"/>
      </node>
      <node id="average-temperature" query="@avgt">
        <range values="data.city.average-temperature"/>
      </node>
      <node id="precipitation-days" query="@precd">
        <range values="data.city.precipitation-days"/>
      </node>
      <node id="latitude" query="@lat">
        <range values="data.city.latitude"/>
      </node>
      <node id="longitude" query="@lng">
        <range values="data.city.longitude"/>
      </node>
      <node id="selected" query-type="interaction" query="selected">
        <range/>
      </node>
      <node id="hovered" query-type="interaction" query="hovered">
        <range/>
      </node>
    </node>
  </node>
  <node id="wordplot" src-type="svg" src="wordplot.svg" query-type="xpath">
    <node id="word" query="/svg:svg[@id='words']/svg:text">
      <node id="x" query="@x"/>
    </node>
  </node>
</bind>
```
Appendix B. Use Case Bindings

B.2 Africa Stats Binding

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bind>
<!-- metastructures -->
<node id="data" src-type="xml" src="mondial.xml">
  <node id="country" query-type="xpath" query="./country">
    <node id="code" query="@car_code">
      <range />
    </node>
    <node id="name" query="/name">
      <range values="data.country.name" />
    </node>
    <node id="population" query="/population">
      <range values="data.country.population" />
    </node>
    <node id="population-growth" query="/population_growth">
      <range values="data.country.population-growth" />
    </node>
    <node id="infant-mortality" query="/infant_mortality">
      <range values="data.country.infant-mortality" />
    </node>
    <node id="total-gdp" query="/gdp_total">
      <range values="data.country.total-gdp" />
    </node>
    <node id="inflation" query="/inflation">
      <range values="data.country.inflation" />
    </node>
  </node>
</node>
</bind>
```
Appendix B. Use Case Bindings

27 </node>
28 </node>
29 <node id="map" src-type="svg" src="africa.svg">
30  <node id="territory" query-type="xpath" query="/svg:g[@id='territories']/svg:path | /svg:g[@id='territories']/svg:g">
31   <node id="id" query="@id">
32     <range />
33   </node>
34   <node id="fill-opacity" query="@fill-opacity">
35     <range min="0.05" max="1" unit="" step="0.05" />
36   </node>
37   <node id="hovered" query-type="interaction" query="hovered">
38     <range />
39   </node>
40 </node>
41 </node>
42 <node id='detailview' src-type='html' src='detail.html' query-type='xpath'>
43  <node id='detail' query="/div">
44   <node id='title' query="/h3">
45     <range def="text" />
46   </node>
47   <node id='property' query="/p">
48     <node id='name' query="/span[@class='name']">
49       <range />
50     </node>
51     <node id='value' query="/span[@class='value']">
52       <range />
53     </node>
54   </node>
55   <node id='hovered' query-type='interaction' query='hovered'>
56     <range />
57   </node>
58 </node>
59 </node>
60 <!-- bind & adapt -->
61 <bind>
62  <node ref='data.country' match='value('country.code')'/>
63  <node ref='map.territory' match='value(territory.id)'/>
64  <node ref='detailview.detail' miss="clas"/>
65  <bind>
66    <node ref='country.infant-mortality' />
67    <node ref='territory.fill-opacity' adapt="map( infant-mortality, \pow4 ')"/>
68  </bind>
69 <bind>
70  <node ref='country.name' />
71  <node ref='detail.title' adapt='value('name')'/>
72 </bind>
73 <bind>
74  <node id='attribute' ref='country.*'/>
75  <node ref='detail.property' miss="clas"/>
76  <bind>
77   <node ref='property.name' adapt='id('attribute')'/>
78   <node ref='property.value' adapt='value('attribute')'/>
79  </bind>
B.3 Bertin Hotel Binding

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bind>
    <!-- metastructures -->
    <node id="data" src-type="xml" src="data.xml">
        <node id="item" query-type="xpath" query="./item">
            <node id="count" query="@count">
                <range values="data.item.count" min="0" />
            </node>
            <node id="category" query="@cat">
                <range values="data.item.category" />
            </node>
            <node id="month" query="@month">
                <range values="data.item.month" />
            </node>
        </node>
    </node>
    <node id="barchart" src-type="svg" src="barchart.svg">
        <node id="row" query-type="xpath" query="./svg:svg">
            <node id="y" query="@y">
                <range id="percent" min="0" max="100" unit="%" step="0.1" />
            </node>
            <node id="height" query="@height">
                <range def="percent" />
            </node>
            <node id="bar" query="./svg:rect">
                <node id="x" query="@x">
                    <range def="percent" />
                </node>
                <node id="y" query="@y">
                    <range def="percent" />
                </node>
                <node id="height" query="@height">
                    <range def="percent" />
                </node>
                <node id="width" query="@width">
                    <range def="percent" />
                </node>
                <node id="class" query="@class">
                    <range />
                </node>
            </node>
        </node>
    </node>
    <node id="legend" query="./svg:text">
        <range />
    </node>
</bind>
```
Appendix B. Use Case Bindings

B.4 Migration Graph Binding

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bind>
<!-- metastructures -->
<node id="migrations" query-type="xpath" src-type="xml" src="migrations-5000.xml">
  <node id="county" query="../graph/node">
    <node id="id" query="@id">
      <range />
    </node>
    <node id="x" query="../data[@key='x']">
      <range values="migrations.county.x" />
    </node>
    <node id="y" query="../data[@key='y']">
      <range values="migrations.county.y" />
    </node>
    <node id="name" query="../data[@key='tooltip']">
      <range />
    </node>
  </node>
</bind>
```
<node id="flow" query="./graph/edge">
  <node id="id" query="@id">
    <range />
  </node>
  <node id="weight" query="./data[@key='value']">
    <range values="migrations.flow.weight" />
  </node>
  <node id="source-id" query="@source">
    <range />
  </node>
  <node id="target-id" query="@target">
    <range />
  </node>
  <node id="source" ref="migrations.county[id={value('source-id')}]"><node id="target" ref="migrations.county[id={value('target-id')}]"></node>
</node>

<node id="graphic" src-type="svg" src="graphic.svg" query-type="xpath">
  <node id="dot" query="./svg:svg[@id='dots']/svg:circle">
    <node id="x" query="@cx">
      <range id="percent" min="10" max="90" unit="%" step="0.01" />
    </node>
    <node id="y" query="@cy">
      <range def="percent" />
    </node>
    <node id="size" query="@r">
      <range min="8" max="20" unit="" step="2" />
    </node>
    <node id="title" query="./svg:title">
      <range />
    </node>
  </node>
  <node id="line" query="./svg:svg[@id='lines']/svg:line">
    <node id="x1" query="@x1">
      <range def="percent" />
    </node>
    <node id="y1" query="@y1">
      <range def="percent" />
    </node>
    <node id="x2" query="@x2">
      <range def="percent" />
    </node>
    <node id="y2" query="@y2">
      <range def="percent" />
    </node>
    <node id="width" query="@stroke-width">
      <range min="1" max="16" step="1" />
    </node>
  </node>
</node>

<!-- bind & adapt -->
<bind>
    <node ref="migrations.flow" />
    <node ref="graphic.line" miss="clas" />
    <bind>
        <node ref="line.width" adapt="map('flow.weight')" />
        <node ref="line.x1" adapt="map('flow.source.x')" />
        <node ref="line.y1" adapt="map('flow.source.y')" />
        <node ref="line.x2" adapt="map('flow.target.x')" />
        <node ref="line.y2" adapt="map('flow.target.y')" />
    </bind>
</bind>

<bind>
    <node ref="migrations.county" />
    <node ref="graphic.dot" miss="clas" />
    <bind>
        <node ref="dot.title" adapt="value('county.name')" />
        <node id="dx" ref="dot.x" adapt="map('county.x')" />
        <node id="dy" ref="dot.y" adapt="map('county.y')" />
    </bind>
</bind>
</bind>
Appendix C

Metaphorea Framework, v0.2

The development of the visualization framework is hosted at http://code.google.com/p/metaphorea/. It is licensed under LGPL terms. The version of the framework discussed in this thesis is the version 0.2, downloadable at the same address, and annexed to the official copies of this master thesis on a CD-ROM disk.

C.1 Framework Content

The framework release is composed of several folders and files. We detail them here.

/metaphorea-0.2/

- /libs/ libraries the framework depends upon
  - javascript-xpath.js library, required for to manipulate XML documents.
- /src/ core of the framework
  - db.js core binding engine.
  - adapters.js adapters for XML and CSS manipulations.
  - metaphoreajs.js visualization layer, wrapper to the binding engine for visualization purposes.
- /usecases/ example use cases
  - /africa_stats/ see Section 5.3.
  - /bertin_hotel_reviewed/ see Section 5.4.
  - /city_weather/ see Section 5.2.
  - /migration_graph/ see Section 5.5. for visualization purposes.
C.2 Framework Usage

Deployment in an HTML page requires a few easy steps.

1. prepare data and visual sources and express the binding (see Section 5.3),

2. include needed ecmscript sources in the web page:

   1. `<script src="libs/javascript-xpath-latest-cmp.js" type="text/javascript"></script>`
   2. `<script src="src/dbe.js" type="text/javascript"></script>`
   3. `<script src="src/adapters.js" type="text/javascript"></script>`
   4. `<script src="src/metaphorea.js" type="text/javascript"></script>`

3. set target view space, set view frames size, finally load binding:

   1. `$metaphorea.setTarget("targetViewsSpace");`
   2. `$metaphorea.setFrameDimensions(300,300);`
   3. `$metaphorea.binding("binding.xml");`

Note that all other document sources (data, wordplot, ...) are specified in the binding itself, with path relative to including page.
Bibliography


