Prototyping with Data

Opportunistic Development of Data-Driven Interactive Applications

FILIP KIS

Doctoral Thesis (No. 29, 2016)
KTH Royal Institute of Technology
Computer Science and Communication
Media Technology and Interaction Design
SE-100 44 Stockholm, Sweden
Akademisk avhandling som med tillstånd av KTH i Stockholm framlägges till offentlig granskning för avläggande av teknisk doktorsexamen fredag den 16 december kl. 13:00 i sal F3, KTH, Lindstedtsvägen 26, Stockholm.
Do. Or do not. There is no try.
Abstract

There is a growing amount of digital information available from Open-Data initiatives, Internet-of-Things technologies, and web APIs in general. At the same time, an increasing amount of technology in our lives is creating a desire to take advantage of the generated data for personal or professional interests. Building interactive applications that would address this desire is challenging since it requires advanced engineering skills that are normally reserved for professional software developers. However, more and more interactive applications are prototyped outside of enterprise environments, in more opportunistic settings. For example, knowledge workers apply end-user development techniques to solve their tasks, or groups of friends get together for a weekend hackathon in the hope of becoming the next big startup. This thesis focuses on how to design prototyping tools that support opportunistic development of interactive applications that take advantage of the growing amount of available data.

In particular, the goal of this thesis is to understand what are the current challenges of prototyping with data and to identify important qualities of tools addressing these challenges. To accomplish this, declarative development tools were explored, while keeping focus on what data and interaction the application should afford rather than on how they should be implemented (programmed). The work presented in this thesis was carried out as an iterative process which started with a design exploration of Model-based UI Development, followed by observations of prototyping practices through a series of hackathon events and an iterative design of Endev – a prototyping tool for data-driven web applications. Formative evaluations of Endev were conducted with programmers and interaction designers.

The main results of this thesis are the identified challenges for prototyping with data and the key qualities required of prototyping tools that aim to address these challenges. The identified key qualities that lower the threshold for prototyping with data are: declarative prototyping, familiar and setup-free environment, and support tools. Qualities that raise the ceiling for what can be prototyped are: support for heterogeneous data and for advanced look and feel.
Sammanfattning


Målet med arbetet som presenteras i den här avhandlingen har varit att förstå utmaningarna som det innebär att använda data i prototyparbete och att identifiera viktiga kvalitéer för de verktyg som ska kunna hantera detta. För att uppnå detta mål har verktyg för deklarativ programmering utforskats med ett fokus kring vilken data och interaktion en applikationen ska erbjuda snarare än hur den ska implementeras. Arbetet som presenteras i den här avhandlingen har genomförts som en iterativ process, med en startpunkt i en utforskning av modellbaserad gränssnittsutveckling, vilket sedan följdes av observationer av prototyparbete i praktiken genom en serie hackathon och en iterativ design av Endev, som är ett prototypverktyg för att skapa datadrivna webbapplikationer. Formativa utvärderingar av Endev har utförts med programmerare och interaktionsdesigners.

De viktigaste resultaten av den här avhandlingen är de utmaningar som har identifierats kring hur man skapar prototyper och de kvalitéer som krävs av prototypverktyg som ska adressera dessa utmaningar. De identifierade kvalitéerna som sänker trösklarna för att inkludera data i prototyper är: deklarativt prototyparbete, välbekanta och installationsfria miljöer, och supportverktyg. Kvalitéer som höjer taket för vad som kan göras i en prototyp är: stöd för olika typer av data och för avancerad “look and feel”.

Acknowledgments

When I first moved to Sweden in 2011, I could not have imagined how much this experience would influence my life. I know it is not possible to name everybody who has made an impact on me in the past years, so I first want to give my gratitude to all of you who have been beside me throughout this adventure.

I would like to thank Fabio Paternó for accepting the challenge of acting as an opponent to my thesis. I also want to extend my gratitude to Margaret Burnett, Hallvard Trætteberg, Felienne Hermans, and Anne Håkansson for taking the time to read my work and come to KTH as my grading committee. In addition, I would like to thank Alan Blackwell for reviewing an early draft of my thesis. Your thoughtful comments have helped me find my position in the world of end-users and will influence my work in the years to come. I also want to give my gratitude to Gaëlle Calvary for reviewing my work halfway through my PhD and inspiring me to take a new direction by letting go of the models that have shaped the early days of my work. And to Roberto Bresin for reading my nearly finished thesis and providing insightful comments.

I would like to express my deepest gratitude to my main supervisor Cristian Bogdan for bringing me to Sweden, guiding me through the academic jungle, encouraging me when I thought I was lost and eventually letting me find my own path. This thesis would truly not be possible without you. I also want to thank Olle Bälter, my co-supervisor, who with his wise comments and sincere care stayed true to his username.

The PhD work can, at times, be lonely. At other times, my dear colleagues at Media Technology and Interaction Design department were there to discuss anything and everything, and for this they will always hold a special place in my heart. My roommates Rebecka Cuppitt, Henrik Åhman, Susanna Heyman, Cecilia Teljas, Åke Walldius, it was truly a pleasure sharing rooms with you and having somebody who can listen to you after a long meeting or just a usual Monday morning. Anders Lundström and Fredrik Enoksson, I had the pleasure to work with you and to learn from you. To Elina Eriksson I want to give my warmest embrace for the positive attitude and joy she brought to the 6th floor and although you are now only one floor below, know that you are missed. A special thank you to Marcus Nilsson and Pernilla Josefsson.
for making me feel very welcome in the new environment. I will not try to name all the colleagues at the department that I have had the pleasure of working next to and sharing fun moments with, but you should know I have appreciated every moment, no matter how big or small, that we have shared together and for that I sincerely thank you.

At KTH I also had a pleasure of working with people from other schools of which I especially want to thank Björn Palm, Omar Shafqat, and Jaime Arias Hurtado for the time and dedication they put into our collaboration.

During my PhD I have had the opportunity to visit other universities and research groups and I am grateful to all the people I met along the way for their support, feedback and advice. I would especially like to thank the wonderful people at TU Delft, starting with Michel Oey for arranging my stay and Martijn Warnier for serving as a mentor while I was there. Then there are my roommates (who are really spies), Yilin Huang and Nina Voulis. I have never had as much laughter in a work environment as with the two of you.

A collaboration with the ICT group at TU Vienna has had a profound impact on the early work in this thesis. I have learned a lot about discourse modeling and writing from Hermann Kaindl, whom I also want to thank for making this collaboration possible. Additionally, I want to thank David Raneburger and Roman Popp, who provided me an invaluable insight into their tools and code so I could build upon it for my own work.

As part of the PhD education I have also taken part in the EIT Digital’s Doctoral School program and I would like to thank Martin Vendel from KTH, who has been a great support as well as an amazing teacher throughout the program. It is inspiring to see that in the increasingly complex systems there are still people who apply the common sense principle. Through the EIT program I also had the pleasure of working with Sanja Šćepanović, with whom I enjoyed many interesting discussions about our shared passion for entrepreneurship.

Outside of my academic work, during my time in Sweden I have managed to build another work environment within my startup, TrainedOn. Doing that has helped me stay in touch with “the real world” and evaluate my research against what I have seen in the industry. I was not alone in that endeavor and I want to thank Robert Cserti, my business partner and friend, who had
patience and understanding when I would disappear for days on end, chasing paper deadlines.

I have always considered myself lucky to have dear friends all around the world. And this was true for Sweden even before I moved here. Which is why I want to wholeheartedly thank Maria Håkansson and Philip Talani, who have taken me in when I experienced the harshness of Stockholm’s housing market. Whether dancing salsa, enjoying Swedish traditions or having dinner parties, I have never felt alone and for this I thank you Vasia Kalavri, Paula Yadranka Žitinski Elías, Veronica Gustafsson, Christophe Van der Kelen, and many, many others. I do hope to see all of you more often now that this chapter of my life is coming to a close. And to all of my friends, whom I haven’t seen as much as I would have wanted, I want you to know that you were always somewhere close in my mind, bringing me strength and joy when I needed them, through the memories of the great times we had and through hope of the new adventures to come.

While it is not easy to maintain close relationships over distance, I am thankful that the bond of friendship with Tassos Natsakis only grew stronger in these years. It was probably our pursuit of knowledge that has brought us closer and I want to sincerely thank you for all the reflective moments we have shared.

To my closest friends Andrej Ficnar, Marina Ulemek, Ivana Štulić, Manuel Bernhardt, Rudolf and Diana Mayer, I want to thank you for the support you have given me over the years and the distance. I am truly privileged to have you in my life.

I owe my sincere gratitude to my whole family, who have always been there for me and who I miss the most in my new home. Thank you for all the homemade desserts, emails, pictures, and calls that made me feel I was back in Croatia, and for your advice, encouragement and love that always keep me going forward. I want to thank Silva Tomanić Kiš for giving me invaluable support by proofreading my work.

And finally, the greatest result of the past years is finding my love and partner Hanna Hasselqvist. You have been my inspiration, support and a source of endless love. You are a hope for a wonderful future.

Filip Kiš
Stockholm, November 2016
## Abreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AUI</td>
<td>Abstract User Interface</td>
</tr>
<tr>
<td>CA</td>
<td>Communicative Acts</td>
</tr>
<tr>
<td>CD</td>
<td>Cognitive Dimensions</td>
</tr>
<tr>
<td>CRF</td>
<td>Cameleon Reference Framework</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create-Read-Update-Delete</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>CTT</td>
<td>ConcureTaskTree</td>
</tr>
<tr>
<td>CUI</td>
<td>Concrete User Interface</td>
</tr>
<tr>
<td>EUD</td>
<td>End-User Development</td>
</tr>
<tr>
<td>EUSE</td>
<td>End-User Software Engineering</td>
</tr>
<tr>
<td>FUI</td>
<td>Final User Interface</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HCI</td>
<td>Human-Computer Interaction</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>MBUID</td>
<td>Model-based User Interface Development</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RST</td>
<td>Rhetorical Structure Theory</td>
</tr>
<tr>
<td>SE</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TAM</td>
<td>Technology Acceptance Model</td>
</tr>
<tr>
<td>UCP</td>
<td>Unified Communication Platform</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>XAML</td>
<td>eXtensible Application Markup Language</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>eXtensible Stylesheet Language</td>
</tr>
<tr>
<td>YQL</td>
<td>Yahoo Query Language</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WSDL</td>
<td>Webs Service Description Language</td>
</tr>
</tbody>
</table>
# Contents

1 Introduction  
   1.1 Research objective ............................................ 4  
   1.2 Background .................................................. 5  
   1.3 Methodology .................................................. 7  
   1.4 Thesis contributions ......................................... 9  
   1.5 List of papers ............................................... 10  
   1.6 Publications not included in the thesis ................... 12

2 Background  
   2.1 Prototyping ................................................... 15  
      2.1.1 Designers prototyping interactive behavior .......... 17  
      2.1.2 Prototyping tools ..................................... 18  
   2.2 What it means to be declarative ............................. 18  
   2.3 End-User Development ....................................... 20  
      2.3.1 Theoretical EUD research frameworks .............. 21  
      2.3.2 EUD approaches ...................................... 22  
   2.4 Data-binding ................................................ 24  
   2.5 Model-based UI Development ................................ 26  
      2.5.1 Cameleon Reference Framework ..................... 27  
      2.5.2 MBUID approaches .................................. 30  
      2.5.3 Discourse-based UI modeling ...................... 31

3 Towards prototyping with data through Model-based UI Development  
   3.1 From Makumba to MBUID ................................... 35  
   3.2 Fully declarative Discourse Models ..................... 38  
   3.3 Real data with Query Annotations ....................... 39  
   3.4 Towards control over UI .................................. 40  
   3.5 Discussion .................................................. 41  
   3.6 Conclusion ................................................. 44

4 Declarative prototyping with data  
   4.1 Towards prototyping with data approach ................. 45
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2 Prototyping in the wild</td>
<td>48</td>
</tr>
<tr>
<td>4.2.1 The study of prototyping in hackathons</td>
<td>48</td>
</tr>
<tr>
<td>4.2.2 Reevaluating the key qualities</td>
<td>49</td>
</tr>
<tr>
<td>4.3 Designing a new prototyping tool through exemplars</td>
<td>51</td>
</tr>
<tr>
<td>4.4 Endev tool for prototyping with data</td>
<td>55</td>
</tr>
<tr>
<td>4.4.1 Evaluation</td>
<td>57</td>
</tr>
<tr>
<td>4.5 Interaction Designers Prototyping with Data</td>
<td>58</td>
</tr>
<tr>
<td>4.5.1 Reevaluating the key qualities</td>
<td>59</td>
</tr>
<tr>
<td>4.6 Conclusion</td>
<td>60</td>
</tr>
<tr>
<td>5 Discussion</td>
<td>63</td>
</tr>
<tr>
<td>5.1 Answering the research questions</td>
<td>63</td>
</tr>
<tr>
<td>5.1.1 What are the current challenges when prototyping data-driven</td>
<td>63</td>
</tr>
<tr>
<td>interactive applications?</td>
<td></td>
</tr>
<tr>
<td>5.1.2 Which important qualities need to be supported by UI prototyping</td>
<td>65</td>
</tr>
<tr>
<td>tools for data-driven interactive applications?</td>
<td></td>
</tr>
<tr>
<td>5.2 A brief reflection on MBUID, EUD and SE</td>
<td>69</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>71</td>
</tr>
<tr>
<td>6.1 Future work</td>
<td>71</td>
</tr>
<tr>
<td>7 Bibliography</td>
<td>73</td>
</tr>
</tbody>
</table>
1 Introduction

Designing and developing interactive applications with a Graphical User Interface (GUI) has been a matter of research for more than half a century (Myers et al., 2000; Meixner et al., 2011; Rocha Silva et al., 2015) and still there are new challenges emerging from the rapidly shifting technological and social practices. The Open Data and the Internet-of-Things trends are producing more and more data that users are expecting to have access to via GUIs of interactive applications. The hacker, maker, and startup cultures are producing new kinds of environments and new kinds of software engineering practices through which such interactive applications are developed (Briscoe and Mulligan, 2014; Lindtner et al., 2014; Paternoster et al., 2014). This thesis identifies and addresses some of the challenges and opportunities of interactive application design and development, that have emerged as a result of the new technology and software development practices.

The amount of data present today and predicted to be available in the near future is staggering. Open Data initiatives are pushing government and civil society organizations to make their data freely available and reusable (Veljković et al., 2014). The Internet-of-Things promises around 30 billion connected devices by 2020 – compared to “just” 5 billion in 2015 – all of them generating data (Nordrum, 2016). Adding to this, there are already 16,083 web Application Programming Interfaces (APIs) available as of October 2016 according to ProgrammableWeb1, a directory listing for web APIs and mashups. To tap into this abundance there are presently many interactive applications that use the available data (e.g. apps for weather forecasts, public transport planning, bike and car sharing services), but the potential to create new ones is even larger.

This potential has been recognized in the End-User Development research in the form of mashups (Rode et al., 2005; Hartmann et al., 2007; Cappiello et al., 2011), lightweight web applications, usually built by end-users, that

1 http://www.programmableweb.com
combine data from different APIs. However, most of the mashup tools that have seen wider adoption are either targeted at enterprise use (e.g. IBM Mashup Center² or JackBe Presto³) or are very limited in what can be expressed with them (e.g. IFTTT⁴, Microsoft Flow⁵), and are thus far away from supporting rapid creation of interactive applications by the broad range of end-users.

On the other hand, more and more knowledge workers and end-users in general are interacting with data in their daily activities. It was estimated (Scaffidi et al., 2005) that by 2012 there would be 55 million end-users in the U.S. working with spreadsheets and other form of databases (and thus potentially engaging in a form of programming), while only a fourth of them would be professional programmers, which further motivates the need for end-user development. Adding to this is the number of end-users who foster the culture of participation (Fischer, 2009) through blogs, wikis, social media, and other collaborative environments. However, end-user might struggle to move beyond basic capabilities that their environments (e.g. spreadsheets, blogs or wikis) offer, when trying to prototype more interactive interfaces for their needs. Professional interaction designers are faced with a similar challenge when prototyping interactive applications, as they are often skilled at HTML and CSS authoring to get the look of the user interface they want, but struggle to achieve advanced interaction and sometimes even resort to learning basic JavaScript programming to support their task (Myers et al., 2008).

Prototyping offers a cheaper alternative to building a finished product while at the same allowing designers to identify the users’ needs, to involve users in the initial stages of the design and development process, to do early hypothesis testing, etc. (Szekely, 1994; Lim et al., 2008). While challenges of prototyping the “look” of the UI and the “feel” of the interaction have been a central point of HCI research for decades, the abundance of available data potentially brings additional challenges and opportunities of prototyping interactive applications. With traditional interactive applications the data architecture was something that was either designed during development (e.g. for a new system it might be defined in the requirements) or was under the control of the

---

² https://www-10.lotus.com/ldd/mashupwiki.nsf
⁴ https://ifttt.com/
⁵ https://flow.microsoft.com
product owner (e.g. when building a new application or feature over an existing company database). Either way, the data characteristics (e.g. structure and content) were defined and controlled by the application developers allowing designers to easily prototype by using static data (manually entered) and mocked-up data (fake data that has characteristics of the real data). However, the new, abundant data is more complex to understand (e.g. it is controlled by web service providers, it comes in less strict formats like document-based JSON) and therefore should be used directly in the prototypes, as its structure, content, and “nature” (e.g. streaming or asynchronous data from an API) will affect the GUI and the user experience.

Today there are many tools supporting the creation of UI prototypes for interactive applications: from graphic design software and prototyping tools that allow exporting designed interfaces into a form usable by developers, to GUI builders that allow direct manipulation of interface elements. Many vendors of prototyping tools have recognized the need for interactivity when designing user interfaces, so they support various forms of basic navigation and interaction between UI elements that can be easily prototyped. However, when it comes to accessing data, designers have to resort to using dummy data or to rely on advance programming and engineering effort, which is likely to hinder their efforts in achieving interactive prototypes that work with data. In contrast, creating an interactive application is easy for skilled developers. Proliferation of cloud services, popularization of scripting development technologies, and the omnipresence of the web have reduced the prototype building time to days or even hours. This can be witnessed by the growing popularity of hacker, maker, and startup cultures that gather around events like hackathons and design jams, constantly producing working prototypes of various innovative interactive applications (Van Waart et al., 2015). Throughout this process many prototypes are produced and thrown away, often through opportunistic design and development (Hartmann et al., 2006) rather than organized software engineering.

Can some of the “success” of the technologies and development environments used in opportunistic programming be transferred to prototyping tools to lower the threshold for prototyping with data? The work presented in the following chapters attempts to answer that question in order to support designers and end-users in prototyping interactive applications that take advantage of current and future abundance of data.
1.1 Research objective

The objective of this thesis, to support designers in prototyping data-driven interactive applications, is broken down into the following research questions:

1. What are the current challenges when prototyping data-driven interactive applications?

2. Which important qualities need to be supported by UI prototyping tools for data-driven interactive applications?

We define **data-driven interactive applications** as software whose primary function is to allow users to interact with data (e.g. browse, explore, collect, edit) and to do so through interfaces and features that contemporary users expect (e.g. drag-and-drop interaction, real-time data synchronization). That is, we are looking at prototyping beyond just data-driven functionality (for which, for example, a spreadsheet software or mashups might often suffice). We are also looking at prototyping the interaction, the user interface *look* and *feel*, in relation to the data. Examples could be applications to keep a list of tasks, search and book flights, store and manage picture collections, and so on. Although such applications have been around for decades, and design and development of their user interfaces have been studied thoroughly (Myers et al., 2000; Meixner et al., 2011), the recent technological trends (web development technologies, Open-Data, etc.) and emerging opportunistic practices (present in hacker, maker, and startup cultures) have potentially created new opportunities and challenges for development of such applications.

When we talk about **prototyping**, we are focusing on creation of concrete artifacts, i.e. prototypes, that are “concrete representation of part or all of an interactive system” (Beaudouin-Lafon and Mackay, 2002). As such, prototypes have several characteristics:

They support *creativity*, helping the developer to capture and generate ideas, facilitate the exploration of a design space and uncover relevant information about users and their work practices. They encourage *communication*, helping designers, engineers, managers, software developers, customers and users to discuss options and interact with each other. They also permit *early evaluation* since they can be tested in various ways, including traditional usability studies and informal
user feedback, throughout the design process. (Beaudouin-Lafon and Mackay, 2002)

While this definition indirectly implies a structured software engineering process (e.g. through use-specific roles such as managers, developers and engineers, or formal method as usability study), we believe that creativity, communication and early evaluation are also present in more opportunistic processes.

Our research focuses on high-fidelity interactive prototypes (i.e. that run on a device) as opposed to offline prototypes (e.g. sketches, storyboards). Such prototypes can be rapid throw-away prototypes or they can be evolutionary prototypes that evolve into a final system.

The target group for the research questions, and this research in general, are designers prototyping interactive applications. We take a broad definition of “designer” to reflect their intent (i.e. we could have chosen to call them “prototypers”) rather than their design background or level of skills (Nardi, 1993). In other words, they can be end-users hacking an app for a personal use or interaction designers prototyping for a client. Furthermore, they can have various levels of interaction design or programming skills. Such wide definition allows us to escape the traditional software engineering silos (Burnett and Myers, 2014) and potentially have results that better fit the opportunistic style of prototyping (Brandt et al., 2009).

The application of results of this thesis aim to both support designers in prototyping data-driven interactive applications and to draw the attention of the Human-Computer Interaction and Software Engineering research community to the new technology and opportunistic processes for prototyping which, we believe, have only been given marginal attention so far.

1.2 Background

During my undergraduate studies, before I moved to Sweden and started this long journey towards a Ph.D., I encountered Makumba (Bogdan and Mayer, 2009), an innovative web development framework. In its core it was a predominantly-declarative framework that was used in an iterative design process in which end-users, designers, and developers worked together to prototype new features for the tools they were using. As such, it is the origin and inspiration of the core ideas that appear and reappear in this thesis.
At that time, Makumba was used in an amateur and volunteer setting (Bogdan, 2003) to build a data-driven intranet platform. The group of students using Makumba were part of an international student organization and, most of the time, had to work collaboratively over distance. Because of the fast turnover of members, and the nature of the volunteer and amateur setting, the team dedicated to maintaining the IT systems had scarce resources. Therefore, people who were going to use the platform, the end-users, were often involved in the process of designing and prototyping the features. Makumba was the key to such human-centered end-user development process, as it allowed individuals not skilled in programming to build interfaces. The learning process consisted of a basic 2–4 hour training and a few exercises. After that, the students could modify or create new domain models and interfaces that involved form-based data editing. The task of the more skilled developers, a resource that was very scarce, was then to work on more complex features and educate and mentor the learners.

My role in the beginning was to be part of the IT team that worked on improving the intranet platform. As such, I was using Makumba extensively and I gradually became more and more interested in how it worked and how it could be improved. My main interest was building support tools that would help developers to be more efficient by providing some features like suggestions, syntax-highlighting, etc. I developed such tools for my Master’s thesis (Kis, 2010). While seen as useful and valuable, these tools were not used for very long, as they were dependent on the environment which had high setup costs and because they quickly became outdated and more work would be needed to update them.

However, this experience with improving the development tools and technology has lead to the start of my PhD. The initial goal was to see if the qualities that made Makumba successful a decade earlier could be generalized and applied in a more modern environment. By engineering, evaluating and reflecting on other solutions I have developed a deeper understanding of building data-driven interactive applications and framed my work, presented in this thesis, in the context of prototyping with data.

Through the course of my PhD I have been part of several research projects and collaborations that have influenced my work, of which I would like to single out two most influential ones:
Collaboration with the Unified Communication Platform (UCP) research group from Institute of Computer Technology of Technical University of Vienna. The UCP group is behind the Discourse-based modeling approach present in my work (Papers II and III) and collaboration with them has deepened my understanding of Model-based UI Development domain in general and the inner workings of the Discourse-based modeling.

Working on EU funded CIVIS project has provided the opportunity to explore data-driven interactive applications development in practice (Hasselqvist et al., 2016). Throughout this project I have been involved in a series of hackathon events during which the participants built prototypes of applications and services that used energy and social network data. Furthermore, I had a chance to test some of my technical solutions in practice with users.

Finally, by engagement in the Doctoral Program of the European Institute of Innovation and Technology (EIT), my personal interest in entrepreneurship and technical startups has had an important impact on my work. Through the involvement in the startup and tech communities in Stockholm (and Europe at large) I have had a chance to observe and to be part of a context in which interactive applications and related technologies and associated business strategies play a key role. This has both motivated some of my critical views regarding (my own and related) research and inspired some of the solutions proposed in the thesis.

1.3 Methodology

The research presented in this thesis was carried out as an applied research through an iterative design process with the purpose of developing and reflecting on new tools and methods for prototyping interactive applications. Throughout the iterations, various methods, outlined below, were employed to gather input, define, implement, and evaluate new solutions, with the results of each iteration feeding into the design goals of the next one. Each of the iterations contributes to the overall research results of the thesis, which are discussed and evaluated against the research objective and questions (Chapter 5). The four iterations can be summarized as follows:
Iteration 1: The initial input for the design process was previous work carried out in the context of prototyping and end-user development in amateur settings, and my personal experience coming from that setting. From there the initial conceptual approach was designed and a proof-of-concept was implemented (Paper I and Chapter 4). The reflection on the results prompted a search for a formal framework which has lead to Model-based UI Development and the next iteration.

Iteration 2: The initial key qualities were explored within the context of Model-based UI Development. New solutions were engineered (Papers II and III) and self-evaluated through critical reflection (Chapter 3). The reflection resulted in understanding that the formal nature of Model-based UI Development provides certain benefits to engineering of UIs, but modeling activities and models themselves were too structured for the kind of prototyping we were after. Therefore, the next iteration turned more to practice of opportunistic and end-user development.

Iteration 3: The key qualities were further refined based on lessons from previous iterations and on additional input gathered through observations of prototyping practice in hackathon settings (Chapter 4). Various solutions were explored through rapid prototyping resulting in the latest solution, Endev, presented in Paper IV. Endev was evaluated with users experienced with prototyping interactive systems (Paper IV) by collecting and analyzing qualitative data from their experience of using Endev in tutorial sessions and with the Technology Acceptance Model (Davis, 1989) measuring perceived ease of use and perceived usefulness.

Iteration 4: Prototyping with data was evaluated with interaction designers through a study that involved semi-structured interviews and practical prototyping tasks with Endev (Paper V). The Cognitive Dimensions framework (Green and Petre, 1996) was used to analyze problems designers might face when using Endev (and other prototyping tools) in the context of prototyping with data. The results were used for a final refinement of the key qualities which then fed into overall thesis results and potential future work.

Throughout the work presented in this thesis we have complemented the traditional HCI methods like surveys and interviews, with several specific
analytical tools in order to theoretically ground the findings. In the study presented in Paper IV, Technology Acceptance Model (TAM) was used in the survey part of the study to measure perceived ease-of-use and perceived usefulness of the proposed solution. TAM is based on Theories of Reasoned Action (Fishbein and Ajzen, 1975) and is used to predict behavioral intention which has consistently been shown as a strong predictor of actual use (Davis, 1989). It is, therefore, a predictive model and cannot be used to identify specific system design flaws, but rather an acceptance of technology (Morris and Dillon, 1997). It is often used on systems that have not yet been implemented (Davis and Venkatesh, 2004) or are not yet in use by the users (Venkatesh et al., 2000; Subrahmaniyan et al., 2008) and thus presents a useful method for early evaluations.

In our follow-up study with interaction designers, presented in Paper V, we have used Cognitive Dimensions (CD) to identify usability issues the designers faced when using our proposed solution. CDs provide a broad-brush descriptive approach to identifying cognitive implications that underline a design of a programming language or a visual notation. While originally intended to be used before the language or notation were developed, researches have used CDs to evaluate various notations and at different stages of development (Blackwell and Green, 2000; Burnett et al., 2001; Clarke, 2005; Dagit et al., 2006; Green et al., 2006).

1.4 Thesis contributions

The work presented in this thesis has the following main contributions:

As an answer to research question 1, several challenges to prototyping data-driven interactive applications were identified (Chapter 5) and can be summarized as: high threshold of having real data while maintaining control over the UI; prototyping certain functional aspects related to data; discovering and understanding the data.

As an answer to research question 2, the following key qualities of tools for prototyping data-driven applications are established (Chapter 5): declarative prototyping, heterogeneous data support, familiar and setup-free environment, support for advanced look and feel, and developer support.
» Fully-declarative Discourse Modeling was designed and implemented in order to reduce the complexity of high-level Discourse Models (Paper II, Chapter 3).

» A Query Annotated Discourse Modeling approach supporting prototyping of data-driven interactive applications was designed and implemented (Paper III, Chapter 3).

» Critical reflection on Model-based UI Development as prototyping tools resulting from the exploration of various solutions (Chapter 3).

» Declarative Prototyping with Data approach was devised through iterative design (Paper I, Chapter 4).

» Observations were made of prototyping practice in hackathon settings (Chapter 4).

» The Endev tool as a proof-of-concept for Declarative Prototyping with Data was developed (Paper IV, Chapter 4).

» A study of prototyping practice was conducted with interaction designers (Paper V, Chapter 4).

» A Cognitive Dimension analysis of Endev and other tools supporting prototyping with data from the perspective of interaction designers was carried out (Paper VI, Chapter 4).

1.5 List of papers

This thesis is based on the following publications, which are referred to throughout the thesis:


This paper introduces a new approach to prototyping (referred to as UI modeling in the paper) that relies on annotating the user interface template with queries which are then used to achieve a working prototype. The paper
also presents the proof-of-concept implementation. I have conceptualized the proposed model for the prototyping and have led the writing of the paper. The second author has implemented the proof-of-concept and contributed to the writing.


The paper presents a fully declarative version of the Discourse-based modeling approach achieved by removing the procedural constructs and using model annotations to automatically generate a GUI. The generated prototype presents the layout and the navigation of the UI. The initial idea for this paper stems from previous work by the co-authors. I have implemented the developed idea and lead the writing process. Second and third co-authors contributed to writing.


This paper builds on the previous two papers and introduces query annotations to a declarative Discourse-based modeling approach. The resulting modeling approach is used to generate working prototypes that support interaction with data. I have designed and implemented the proposed approach which was conceptualized by both authors. I have also led the writing process. The second author contributed to writing.


This paper presents an approach for prototyping web applications that support use of real data. The approach presented is built upon the general concept presented in Paper I and expanded by providing a setup-free
environment and support for web service data. The solution was evaluated with programmers experienced with building interactive systems. I initiated the idea, implemented the solution, designed and conducted the study and wrote most of the paper. The second author provided feedback throughout the process and helped with data analysis.


This paper summarizes a study on prototyping with data conducted with interaction designers. The study included interviews with professional interaction designers and observations of how they use the prototyping approach presented in Paper IV. I have designed and conducted the study, analyzed the data and written most of the paper. The second author provided a designer perspective in the analysis and contributed to writing.

1.6 Publications not included in the thesis

Throughout my PhD studies I have contributed to the following publications that are not included in the thesis. Work on these publications has, however, had an impact on my thesis as I have either had the role of developing data-driven applications or observing others performing that task.


2 Background

The work presented in this thesis is informed by research from several domains including End-User Development and Model-based UI Development (MBUID). This chapter presents the relevant work in these domains and introduces the overarching concepts of prototyping and “declarativeness” that are recurring themes throughout the thesis.

2.1 Prototyping

Developing support for prototyping of interactive systems is the core theme of this thesis. Prototyping is used in fields such as human-computer interaction (HCI), software engineering and architecture to create objects (prototypes) that are used in the design process. Prototyping in interaction design aims to evaluate design ideas, solicit feedback from users and other stakeholder, and allow designers to explore the design space by generating design alternatives (Beaudouin-Lafon and Mackay, 2002). Prototyping is an iterative process where creation, testing and refinement of a prototype are repeated until a satisfactory solution is achieved (Nielsen, 1993). Produced prototypes are a representation of an interactive system in a concrete way, not an abstract description. Thus, from the economic perspective:

The best prototype is one that, in the simplest and the most efficient way, makes the possibilities and limitations of a design idea visible and measurable. (Lim et al., 2008)

In order to satisfy various constraints (e.g. time and budget) typically present in software development projects, prototypes should be produced in a fast and cheap manner. While benefits of prototyping are often not understood by stakeholders other than designers, research show that rapid iterative prototyping produces more valuable design than immediate realization of a final solution (Dow and Klemmer, 2010). Evolutionary prototyping is often
used to address the concerns regarding constraints by incrementally improving prototypes, rather than throwing them away, towards the final solutions. Our work targets the wide range of both rapid and evolutionary prototypes since research show that end-users often create solutions with intention to “throw them away”, but those solutions end up being used over long periods of time (Mackay, 1990).

In order to categorize prototypes, Lim et al. (2008) define the filtering and manifestation dimensions. Filtering dimensions, such as appearance, data, interactivity and functionality, “correspond to various aspects of the design idea that the designer tries to represent in a prototype”. On the other hand, manifestation dimensions represent the form of the prototype, i.e. the material (pen and paper, prototyping tool, programming environment), resolution (fidelity) and scope (range of features covered). The resolution of the prototype is commonly expressed in terms of filtering dimensions, for example, level of details of the appearance (e.g. low-fidelity sketch, high-fidelity implementation), the data (e.g. fake or real data), the interactivity (e.g. static, feedback, input and output behavior) etc.

Throughout this thesis, the focus is on prototypes with generally high resolution, in other words, fully interactive prototypes with high level of detail when it comes to appearance and data. Furthermore, the aim is to also support medium level of functionality such as creation, modification and persistence of data. This corresponds to the definition of prototyping coming from the domain of Information Systems (related to this thesis for its data-driven focus) where the prototype is an early version of a system that exhibits the essential features of the later operational system (Sprague Jr. and Carlson, 1982).

The term opportunistic prototyping is used throughout the thesis to define rapid prototyping that occurs with little or no upfront design and planning. The definition is inspired by Stanford HCI group’s work on opportunistic programming (Brandt et al., 2009) which they define as:

“activity where non-trivial software systems are constructed with little to no upfront planning about implementation details, and ease and speed of development are prioritized over code robustness and maintainability” (Brandt et al., 2009)

Opportunistic design is common in early stages of software development when working on ill-defined problems (Guindon, 1990), therefore resulting in a bottom-up approach.
An emerging trend in software development is live prototyping (Aycan and Lorenzoni, 2015) where the prototype is given to users for actual use and not just testing. Such prototypes are typically used to test the market appeal of an idea and therefore have just enough fidelity to satisfy the market conditions (e.g. mobile app store criteria, offering alternative to established solutions).

Another study (Israel et al., 2016) gives perspectives on the future of prototyping. Interviewed experts expect the future prototyping to be high-tech, participatory (allowing citizens to develop products according to their own needs) and evolutionary (prototypes are easy to deploy and use), which is also what this thesis is aiming to support.

### 2.1.1 Designers prototyping interactive behavior

When prototyping, according to Schön (1996), designers approach ill-defined problems through reflection-in-action. Through iterations of defining, acting (e.g. prototyping) and reflecting they engage in a series of “conversations with material”. The creation of concrete prototypes leads designers to realizations that they would have likely not had by just thinking about the possible solutions. However, when it comes to designing interactive systems, designers are struggling to engage in conversation with digital material (Ozenc et al., 2010). The reason for this is the continuous change of the material (new hardware, software and programming environments) as well as the designers’ lack of skills to work with digital material (Myers et al., 2008; Buxton, 2010).

Therefore, most designers need to resort to some form of programming when prototyping interaction (Myers et al., 2008) since prototyping tools support only the static look or very basic interaction. When asked what features future prototyping tools should support, the designers mentioned, among other things: “support for more dynamic behavior”, “data-driven interactions” and “ease of connection to live data sources”, which are all among the aims of the work presented in this thesis. This is further corroborated by the research of needs of interactive application designers (Grigoreanu et al., 2009), which showed that most designers sought the ability to prototype the flow of data, events, and other resources.

The need to resort to writing code while prototyping interactive applications places interaction designers in the category of end-user developers (Lieberman et al., 2006; Ko et al., 2011). Their primary goal in writing code is not to
program, but rather to support their design activity. During such opportunistic development efforts, speed and ease of development are often preferred over code robustness and maintainability (Brandt et al., 2008).

2.1.2 Prototyping tools

A wide variety of tools can be used for prototyping: graphic design software, animation software, presentation software, user interface builders, Model-based UI Development tools, dedicated UI prototyping tools as well as software code (Szekely, 1994; Beaudouin-Lafon and Mackay, 2002). Among those tools, Carter and Hundhausen (2010) identify graphic design software and presentation software to be preferred (excluding art supplies used for offline prototypes). Rocha Silva et al. (2015) identify an emerging trend of dedicated industry prototyping tools inspired by past academic research.

These findings are partially corroborated by a more recent large-scale industry survey of the tools designers are using, with more than 4,000 designers responding (Subtraction, 2015). In the earlier stages of design, offline and of lower fidelity, designers preferred dedicated tools (Sketch6 being the most favored) to created mock-ups. However, in the prototyping phase the preferred tool was HTML (+ CSS) over tools specifically designed for prototyping (e.g. InVision7) or even general tools that designers are more familiar with (e.g. Photoshop, Keynote). These results further indicate the benefits of having an “executable design” with greater control over details already during prototyping.

2.2 What it means to be declarative

Throughout this thesis the term “declarative” is used in its general meaning implying the focus on what is the desired outcome (e.g. UI populated with data coming from a data source) instead of how it is supposed to be achieved (e.g. connecting to data sources, retrieving the data, drawing a UI element for each data result, etc.). This is, for example, applied to UI models (Chapter 3, Paper II and III) and data-query languages (Chapter 3 and 4, Paper I, III

---

6 https://www.sketchapp.com/
7 https://www.invisionapp.com/
and IV). Since there is no clear-cut definition of the term “declarative”, it is important to define its origin and uses in related work.

The concept of declarative knowledge first appeared in the field of Artificial Intelligence in the 1970s but it originated from the longstanding philosophical distinction between “knowing that” and “knowing how” (Ryle, 1945). The declarative knowledge represents the world through facts about which one can then reason to come to a conclusion (result). The procedural knowledge, on the other hand, represents the world through different procedures that, when executed in order, provide the result (conclusion).

Winograd (1975) describes several benefits of declarative versus procedural knowledge. It is seen as more economical as it does not require the specification on how it is used, thus one declarative expression (e.g. logical formula) could express knowledge that would require several procedural statements to achieve the same result. Furthermore, the simplicity of declarative expressions makes it easier to understand and acquire new knowledge. Finally, since natural language is primarily declarative, it is easier to communicate declarative knowledge compared to procedural.

This debate had a direct impact on programming since programming is, in essence, transforming knowledge (about a task or a problem) into code that can be automated and executed by a computer. Early declarative programming languages were soon developed following the same principles of describing what rather than how. The primary example from the artificial intelligence domain was Prolog and many of its derivatives.

Even though general-purpose declarative programming languages have not reached mainstream adoption, many special-purpose languages today are considered to be declarative. For instance, database query languages (SQL, XQuery), spreadsheet “programming” expressions, and various markup languages to present and format information (XML, HTML, XSLT, CSS), to name a few. Furthermore, there is an extensive body of work that promotes declarative approaches for challenges related to this thesis such as UI modeling (Szekely et al., 1993; Da Silva, 2001; Tran et al., 2010), end-user development (Vuorimaa et al., 2016), and web engineering (Hevery and Abrons, 2009; Heinrich and Gaedke, 2011; Toffetti et al., 2011).

In general, when a higher level of abstraction is required, often the declarative expressions are used as they provide advantages identified earlier
by Winograd. These benefits are especially important when a special-purpose language is meant to be used by end-users which is the case with many from the above list. For example, SQL is often used by domain-experts in various business settings while HTML and CSS are extensively used by interaction designers.

Declarative concepts are an important part of the software design and development process in general. Various types of declarative models are used to provide system specifications, for instance use-cases, sequences, states, and domain models, with the most widely known models belonging to the UML (Unified Modeling Language) group. Such models have been crucial in human-centered development approaches, as they allow various stakeholders, including non-technical ones, to discuss concepts of interactive software.

2.3 End-User Development

This thesis aims to support designers of various skills in using computer software to produce data-driven interactive applications. This places the work in the area of End-User Development (EUD), which is defined as:

“a set of methods, techniques and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact” (Lieberman et al., 2006)

This definition does not necessarily exclude professional software developers but rather focuses on the end-users’ goals. As Nardi (1993) defined it, their goal is not to produce software, but rather software is a means to accomplish their task. Therefore, even users with significant programming skills, e.g. system administrators (Barrett et al., 2004) or research scientists (Carver et al., 2007), consider programming only as means to complete tasks in their work. However, while the end-user’s skills are not necessarily limited, in practice many end-user developers lack formal programming training.

The reason why end-users with lack of training engage in programming is that there are not enough professional developers to meet all the software development needs emerging from growing number of users using computational devices (computers, mobile phones, watches, etc.) for rapidly
changing tasks (Scaffidi et al., 2005). Such diversity among end-users and their tasks has resulted in a broad scope of formative empirical studies looking at EUD characteristics among children (Petre and Blackwell, 2007), knowledge workers (Scaffidi et al., 2006), web designers (Rode et al., 2006) and interaction designers (Brandt et al., 2008; Myers et al., 2008), etc.

According to Myers et al. (2000), desirable traits for end-user development solutions are: low threshold, high ceiling and gentle slope. The “threshold” defines how difficult it is to learn to use the system, while the “ceiling” defines how much can be achieved by using the system. The gentle slope is a relation between threshold and ceiling where for each incremental increase in the level of customizability (i.e. how much the user want to do), only incremental learning is required. Therefore, the desirable end-user development solution is one the that can be used to build complex systems, and which the user can learn to use quickly for simple tasks and easily progress to more advanced use through incremental learning.

2.3.1 Theoretical EUD research frameworks

Burnett and Myers (2014) argue that there is a need for greater inclusion of theory that can ground the work among such empirical EUD projects. For instance, Cognitive Dimensions (Green and Petre, 1996), used in Paper V, provides a framework for identifying usability problems of programming languages and notations (Blackwell and Green, 2000). The framework defines high-level criteria (dimensions) such as closeness of mapping, hidden-dependencies, abstractions, viscosity and so forth that are used to evaluate a (programming) system. Various design alternatives for one systems can then be evaluated by looking at tradeoffs between the criteria.

Another theoretical method is the Model of Attention Investment (Blackwell, 2002) that presents a generalized way to explain how and why an end-user might choose to use a specific EUD tool. It balances users perception of cost, benefits and risks of using a specific tool.

Ko and Myers (Ko and Myers, 2005) have developed a framework for studying the causes of software errors in programming systems. Using these theoretical frameworks and models can provide a “more holistic understanding” of how various EUD research results fit together (Burnett and Myers, 2014).
2.3.2 EUD approaches

When it comes to EUD approaches and tools, the spreadsheet paradigm, where users program by writing declarative expressions and mathematical formulas, is considered to be among the first and most widespread (Nardi, 1993; Burnett and Scaffidi, 2014). Since the early days of spreadsheets various approaches have been developed to support end-users in different context: LabVIEW (Johnson, 1997), a dataflow visual programming environments for lab applications; CoScripter (Leshed et al., 2008) a programming-by-demonstration approach for automating web interaction; DENIM (Lin et al., 2000), supporting early-stage website design through direct manipulation; and so forth. From the perspective of creating data-driven applications Click (Rode et al., 2005) presents a relevant contribution that shares opportunistic work-style goals (Brandt et al., 2008) with this thesis. Click is an integrated development environment for creating web applications that supports EUD by offering a gentle slope of complexity, allowing users to start by customizing template web applications and progressing through wizards and direct manipulation all the way to editing low-level code. However, similar to Makumba (Bogdan, 2003; Bogdan and Mayer, 2009), it is based on the Web 1.0 concepts and technology and, therefore, does not support modern client-side web interaction and web service data sources.

More recently, many efforts have been directed at supporting creation of data-driven web applications in the form of mashups (Hartmann et al., 2007; Lin et al., 2008; Cappiello et al., 2011; Aghae et al., 2013). Mashups, named after the concept of music and video mash-ups, are typically lightweight applications that combine data (and sometimes interfaces) from several web sources to provide new functionality. The typical example is combining mapping service (e.g. Google Maps) with location-based lists (e.g. rental listings on Craigslist).

However, most of these solutions are limited in terms of prototyping data-driven interactive applications in a couple of ways: most solutions focus on data-related functionality while limiting the control over interface design; they mostly offer the possibility of combining existing data sources and do not support user-owned data. Furthermore, while some industry tools (e.g. IBM Mashup Center\(^8\),

\(^8\) https://www-10.lotus.com/ldd/mashupswiki.nsf
JackBe Presto”) offer a wider range of functionality, they are targeted towards professional use and thus require higher skill.

Gneiss (Chang and Myers, 2014b) is a recent contribution that allows users to create data-driven interactive applications combining web service data with the spreadsheet paradigm in a live programming environment. It builds on the results of the similar earlier solution FAR (Burnett et al., 2001) by offering support for combinations of multiple web services, and it offers sorting and filtering on top of the querying mechanism seen in FAR. With Gneiss, the users can drag and drop widgets to the page and connect them with spreadsheet values – thus they have the possibility to define and use their own data. Furthermore, Gneiss supports “importing” of any REST web service which can be interactively combined in the spreadsheet before its data is used in the interface. The main drawback for Gneiss, from the perspective of prototyping, is that the users are limited to working with UI widgets that exist in the system, which significantly reduces the design possibilities. Furthermore, even though the Gneiss’ live programming environment allows users to directly see the result of any change they make in the data or the interface, it also makes them dependent on the environment: once they export the code from the environment, any modification to HTML code could result in breaking the connection of the interface to the data, known as “the round-trip problem”, thus making the exploration of design alternatives more complicated.

Certain groups of mashup tools are focused only on web service data composition that produces data output in raw format – in other words they are not meant for building interactive applications that have UI. For example, data-flow tools such as Yahoo! Pipes! support building mashups of web service data by direct-manipulation of input objects that can be linked together to produce output data. However, research has shown that data-flow programming is often difficult for end-users to understand (Cao et al., 2010). Another example is Yahoo Query Language (YQL) technology that addresses a similar goal of allowing the combination of various web service data, but instead of using data-flow language and direct-manipulation, it utilizes SQL-like syntax to support web service combination. The resulting data can be accessed via a REST API. Although both Yahoo! Pipes and YQL provide a

9  http://mdc.jackbe.com/products/mashboard.php
uniform way of combining various web service data, there is still a significant threshold for users if they wish to use the resulting data in their own interactive applications.

2.4 Data-binding

Data-binding, a form of declarative constraint programming, is used throughout the thesis to enable prototyping with data. Constraints are declarative relationships that are expressed once and then automatically maintained by the system. In the context of user interface development, constraints have a long history starting from Sketchpad (Sutherland, 1964), one of the first GUIs, that allowed users to specify constraints between geometric shapes as they were drawn. If one shape moved the constraints (e.g. two lines being perpendicular or connected via point), other shapes would automatically be adjusted. Constraints have been seen, in general, as helpful during the development of user interface behavior as they removed the need of writing complicated and error-prone spaghetti code of callbacks (Myers, (Hartmann et al., 2007). Despite the various benefits of such declarative constraints, procedural code and event-callbacks have prevailed when it comes to specifying interactive behavior (Myers et al., 2000). However, many user interface frameworks use data-binding, a subset of constraints, to connect GUI elements to underlying data (i.e. variables in the code).

Quilt (Benson et al., 2014) is a recent solution that supports prototyping of data-driven applications by end-users. It uses HTML annotations to connect the interface to a spreadsheet that serves as the datastore. Quilt allows both data read and write, and keeps the interface in synchronization (three-way binding) with the spreadsheet data. Any data sorting, formatting, and computation needs to be defined in the spreadsheets. The HTML annotations introduced by Quilt serve only for data-binding. Quilt takes advantage of the interface tree hierarchy to deduct the relationships between parent and nested annotations. It uses only four different annotations: connect, delete, show-if, hide-if. The annotations have different effects depending on what type of data they connect to. For instance, in some cases connect annotation can result in an array of values, while in others it connects to only one value. The authors claim that such relational annotations provide the possibility to automate
tasks depending on the context and while this might be true, limiting most data-bindings to a single kind of annotation can lead to less readable interface code.

XFormsDB (Vuorimaa et al., 2016) is a declarative data-binding solution that binds to server-side data. It is based on XForms\textsuperscript{10}, a W3C Recommendation, that was designed to be the next generation of HTML forms. XFormsDB depends on having a complex server setup and supports only XML based databases, thus it is not ideal for quick prototyping. Furthermore, even though XForms are relatively old-standard (first version published in 2007), none of the major browsers currently natively support it.

WebSoDa (Heinrich and Gaedke, 2011) is another solution that utilizes declarative annotations and a dedicated server to establish data binding to server-side data. The annotations are specified as HTML Microdata\textsuperscript{11}. Microdata was designed as machine-readable metadata to be used by search engines and web crawlers and as such the browsers do not directly interpret it in any way. Therefore, WebSoDa has a client-side JavaScript library that establishes the binding and connects to the WebSocket provided by its server component. The actual data-binding to the data source is achieved through implementing server-side binding components (i.e. JavaBeans). As with XForms, dependence on server-side technology is a limitation for prototyping. Furthermore, annotations need to be defined twice, once in the HTML and the second time in the server-side technology.

Industry client-side web-development frameworks, like Angular\textsuperscript{12}, Backbone\textsuperscript{13}, Ember\textsuperscript{14} and React\textsuperscript{15}, provide data-binding constraints that keep HTML elements automatically in synchronization with the application data values. In research, ConstraintJS (Oney et al., 2012) applies similar data-binding features as the industry frameworks, but extends their functionality to support finite-state machines, allowing developers to also bind to the state of the data (e.g. in progress of being retrieved, successfully retrieved or error while retrieving). In general, the popularity of data-binding techniques among

\textsuperscript{10} https://www.w3.org/MarkUp/Forms/  
\textsuperscript{11} https://www.w3.org/TR/microdata/  
\textsuperscript{12} https://angularjs.org/  
\textsuperscript{13} http://backbonejs.org/  
\textsuperscript{14} http://emberjs.com/  
\textsuperscript{15} https://facebook.github.io/react/
framework designers indicates its appeal and preference among developers over the traditional approach of manually writing event-based code for keeping the UI in-sync with data.

The drawback in terms of prototyping is that all of the reviewed industry solutions still require a certain amount of procedural JavaScript code to provide the actual connection to the data source, and they often require understanding of Model-View-Controller (Krasner and Pope, 1988) or similar software engineering design patterns, which results in a higher threshold for prototyping. Furthermore, in these approaches data-binding serves only for two-way binding, i.e. binding the view with the value of the client-side instance of the data (e.g. data pre-retrieved from the web service). In case three-way binding is desired, which also updates the data when the data source information changes (e.g. another user changes the data through another user interface), complex code would be needed (e.g. AJAX pull request or implementing WebSockets).

2.5 Model-based UI Development

Parts of the work presented in this thesis (Chapter 3, Paper II and III) belongs to the domain of Model-based UI Development (MBUID) (Szekely, 1994; Paternò, 2000). MBUID strives to allow developers and designers to describe the interactive system through declarative definitions without being concerned with the underlying implementation aspects. MBUID approaches employ declarative models at various levels of abstraction to make them independent of modality (touch, voice, gesture), platform (desktop, mobile), and other aspects of context of use. Such models can then be transformed into more concrete models and, in many cases, a final UI code can be generated.

MBUID originated as an evolution of User Interface Modeling Systems (UIMS) with the aim to provide high-level models to address the emerging complexity of UIs as a result of new technologies, such as object-oriented programming languages, mobile devices, etc. (Puerta and Szekely, 1994). Several works throughout the years have given a snapshot of the current status of MBUID and provided challenges for the future (Da Silva, 2001; Vanderdonckt, 2008; Meixner et al., 2011; Mijailović and Milićev, 2013). Initial MBUID efforts focused around defining UI abstractions and related
models (Sukaviriya et al., 1993; Szekely et al., 1993; Paternò et al., 1997; Griffiths et al., 2001) with more recent efforts focusing on complexity arising from a variety of platforms and modalities (Mori et al., 2004; Meskens et al., 2008; Paternò et al., 2009).

2.5.1 Cameleon Reference Framework

The Cameleon Reference Framework (CRF) is a well established framework for classification of MBUID approaches (Calvary et al., 2003). Figure 1 shows the four levels of abstraction at which UIs can be defined according to the CRF and the transformation paths between these levels. The levels and abstractions are described in detail in the following paragraphs.

At the highest level of abstraction, models of **Tasks and Concepts** (T&C) define the tasks that need to be carried out and the domain-specific concepts that are required by these tasks. Many approaches at this level use **Task Models** derived from task analysis (Shepherd, 1989). The most widely adopted notation for Task Models are ConcurTaskTrees (CTT) (Paternò et al., 1997). The CTT model (an example shown in the top part of Figure 2) represents tasks that can be accomplished by actors (i.e. users or the system itself) during interaction and that can be hierarchically organized and decomposed to the
level of elementary actions. The tasks have temporal relations (i.e. Temporal Operators) that are defined between them to specify the order of execution.

*Discourse Models*, an alternative to Task Models, are based on the principles of human communication and instead of focusing on tasks they focus on communication between the actors (Falb et al., 2006). Discourse Models (bottom of Figure 2) also employ a hierarchical structure and they define relationships among the communication elements. Therefore, there is a *duality* between the two approaches (Popp et al., 2014) in that they can be used to model the same interaction, but they are focusing on different high-level aspects of the interaction. These differences are present among the relationships that can be defined between tasks/communication units. Other differences mentioned by Popp et al. (2014) are a result of research focus and they are mostly constrained to features of the supporting tools. Discourse Models are currently supported through the Unified Communication Platform (UCP)
toolset, while CTT and Task Models in general have a longer history with over a dozen tools implemented. Details of Discourse Models and the related modeling approach, used in the work presented in this thesis (Chapter 3, Paper II and Paper III), will be outlined in the following section.

The next level of abstraction according to CRF (Figure 1) is Abstract UI (AUI). It is the first level on which UI elements are expressed. However, the AUI models are still modality and platform independent. The UI is expressed by defining interaction units that group together several tasks (or communication elements) from the Tasks & Concepts level according to various criteria (logical and conditional grouping, semantic relationship, etc.). For instance, in a flight booking system, one unit could represent searching for flight and the other making the purchase.

The Concrete UI (CUI) defines the UI in a modality dependent, but platform independent manner. For instance, at this level it is known whether the CUI model is for a mobile interface or a standard desktop. Thus the units at the AUI level may be decomposed into smaller units (e.g. to fit a smaller screen), for instance, the searching for flight could be split into input the search criteria and displaying the results. The specific elements of CUI also define the modality, e.g. if the input is keyboard or voice. However, the notion of development platform or GUI toolkit (e.g. Android, HTML, Java) is still abstracted out.

The Final UI (FUI) is the interface defined through platform specific code. The code is either compiled or interpreted to get an interface that can be executed. MBUID approaches typically generate the UI code, but additional application logic code needs to be added at this level to obtain an interactive application. For instance, the generated UI code could be executed to render the UI, but the interface navigation might not work and it would still not be populated with data.

Model-based UI Development typically starts from defining one model (or more) for a specific context of use and at a certain level of abstraction. Model transformations are then employed to generate models for different contexts or at different levels of abstraction. Marked with different arrows, Figure 1 shows the 3 possible directions of the transformation:

- **Abstraction** transforms the lower level (more concrete) models to higher level (more abstract) models, e.g. CUI to AUI
Reification is the opposite of abstraction; it transforms higher level to lower level models, e.g. CUI to FUI.

Translation transforms models from one context of use to another, while keeping the same level of abstraction, e.g. transforming a CUI model targeted for a desktop interface to a CUI model for mobile devices.

2.5.2 MBUID approaches

Many MBUID approaches have been developed by proposing different models at different CRF levels and tools focusing on various transformations. Several MBUID approaches that offer the possibility to generate and run applications from the models are presented in what follows.

UsiXML (Limbourg et al., 2005), developed through the same project as the CRF, is a language specifically designed to support model transformation in all 3 directions and enabling various paths of UI development (e.g. reverse engineering from FUI, generating from AUI, etc.). UsiComp (Frey et al., 2012) is a tool based on UsiXML that supports composition of models on various levels of the CRF and guides the designer through multiple development paths. The modeled application can automatically be generated and executed provided that a functional core exists (required for accessing the data and other functionality of the application).

MARIA (Paternò et al., 2009), based on earlier TERESA approach (Mori et al., 2004), is another language, with a related tool, specifically designed for supporting generation of UI for various device from WSDL16 based Web Services. It supports reification starting from any CRF level and translations between CUI models. If the modeled applications is linked to an existing WSDL a executable application can be generated for various platforms (desktop, web, mobile, etc.).

Tørtteberg (2007) presents a UI modeling approach aimed at modeling information systems and thus supporting domain modeling as a central activity. The solution is designed to support top-down or bottom-up approach and supports user testing of the prototyped solution with sample data and “live” widgets that are available during design-time. A later solution (Tørtteberg, 2016)...

---

16 Webs Service Description Language
integrates with a Eclipse Modeling Framework to offer a rich set of domain modeling capabilities.

The Metawidget (Kennard and Leaney, 2010) is an approach similar to Makumba, the work preceding this thesis, in its support of domain data. It employs CUI level models to generate parts of the UI based on the direct analysis of domain objects (thus avoiding Task Models). An important principle, identified by the Metawidget authors after an extensive analysis of UI development practice, is the possibility to integrate Metawidget techniques with standard industry technologies and practices. Therefore, rather than modeling a whole application, Metawidgets represent reusable components that can be easily integrated in existing applications (e.g. Android, HTML, various industry Java frameworks etc.). The components are bound to underlying application logic and FUI elements can be created either at design-time or during runtime.

While MBUID have not seen wide adoption by the industry, several industry tools like Mendix17, OutSystems18 and WebRatio19 support building of enterprise data-driven application through MBUID. For instance, WebRatio is a tool based on WebML (Ceri et al., 2009) that has recently been accepted as a standard (and renamed to IFML20 to support wider range of UIs rather than just web) by the Object Management Group21. WebRatio supports UI modeling at CUI level which can be linked to data modeled through Entity-Relationship, and generating code for mobile or web platforms. Therefore, it is possible to generate not only UI code, but fully functional data-driven applications. However, these tools target enterprise application development and are therefore not suitable for opportunistic prototyping.

2.5.3 Discourse-based UI modeling

The results from Paper II and III, outlined in Chapter 3, build upon previous Discourse-based modeling work where it was used to generate user interfaces for various devices including desktop (Raneburger et al., 2012), mobile

---

17 https://www.mendix.com/
18 https://www.outsystems.com
19 https://www.webratio.com
20 http://www.ifml.org/
21 http://www.omg.org
(Falb et al., 2007) and robot UIs (Bogdan et al., 2009). As Discourse-based modeling presents a less-known alternative to Task Modeling approaches this section introduces the details of the high-level Discourse Model.

The starting point of Discourse-based modeling is the high-level Discourse Mode situated at the Tasks and Concepts level of the CRF. It is used to define the communication flow between the user and the interactive software system. Figure 3 shows an example of a Discourse Model used to define a communication in an online shop scenario. In this example, the user is able to select product categories, add a product from a category to a shopping cart, view the products in the cart, and enter the payment details.

The basic elements of communication in the Discourse-based modeling are Communicative Acts (CA), represented as leaves in Figure 3, derived by generalization of speech acts from Speech Act Theory (Searle, 1969). They represent single utterances of either the user or the system. The intention of the utterance is represented by the type of CA, which could be: asking a question, giving an answer, stating a request, informing about a new fact, etc. For example, the system can ask a question like Select a product category, Enter payment details, etc., or provide information Selected category, All products in the cart.

Adjacency Pairs define a sequence of two Communicative Acts within a communication dialog. Adjacency pairs are inspired by Conversation Analysis (Luff et al., 1990) and depict a turn-taking interaction such as question-answer, request-accept, etc. In Figure 3 (represented by diamonds), questions asked by the system are paired with the answers provided by the user. Adjacency Pairs
can contain just one element, e.g. an *Informing* CA, since it only presents information and has no response.

Adjacency Pairs are organized in relations according to Rhetorical Structure Theory (RST) (Mann and Thompson, 1988). For example, in Figure 3 the *Elaboration* relation provides the basic communication on the left branch (selecting a product category), and additional communication (selecting a product from the category) on the right.

While a Discourse Model is primarily a declarative model, Popp et al. (2009) introduced Procedural Constructs to extend RST Relations in order to be able to define a sequence of relations. In our example, the topmost *IfUntil* node is such a Procedural Construct. It defines that the left branch (tree) will be the active communication until the condition on the right side (then) is met. In other words, the user will be asked to select categories and products (the left side of the communication) until the user proceeds to checkout. After that, the user will be presented with the products in the cart and an option to enter payment details (the right side of the communication). The need for introducing such concepts arises from the fact that RST is based on the analysis of text, which is static, while Discourse Model defines communication, which is dynamic and dependent on decisions of communication parties. An alternative way to address this need will be presented in Chapter 3.

Having defined the Discourse Model, present at the Tasks and Concept level, the UI can be generated from it by a series of transformations (Kavaldjian et al., 2008). At the Abstract UI level the elements of the model are grouped in various states, together creating a state machine. In other words, each state is a subset of the original model, representing communication elements possible at the given state. At the Concrete UI level the elements of the Discourse Model are transformed into UI widgets through the series of transformation rules that can be extended or modified by modelers. Finally, the Final UI can be generated producing an UI code that can be executed. It will, however, only support transitions between various states (UI navigation), while data and other business logic still needs to be programmed and connected to it.

This limitation is addressed in the thesis by creating a fully-declarative Discourse Model (Paper II) and combining the qualities from previous Makumba work with Discourse-based modeling approach (Paper III). The following chapter outlines these results and places them in the context of this thesis, that is prototyping with data.
3 Towards prototyping with data through Model-based UI Development

This chapter introduces the approach that was developed with the aim of addressing the research objective, which is to support prototyping of data-driven interactive applications. The work summarized here, and detailed in Papers II and III, is presented as follows. First, the key qualities from the work preceding this thesis are outlined and motivation for using Model-based UI Development is given. Subsequent sections present how Discourse-based modeling fits in the outlined key qualities. Finally, the discussion on MBUID as solution to prototyping is given, followed by a conclusion of the chapter.

3.1 From Makumba to MBUID

Makumba, the work that preceded this thesis, enabled a group of end-users to develop and continuously expand an intranet portal to fit their needs (Bogdan, 2003; Bogdan and Mayer, 2009). We have initially identified the following key qualities of Makumba that empowered the said group of end-users to engage in evolutionary prototyping:

» **Declarative approach** – allowed end-users with little or no programming skills to modify or create new interactive features, i.e. to engage in end-user development. Makumba employed queries and annotations, shown in Figure 4, which were easy to understand and use by end-users. They typically went through a two-hour training which gave them the basic concepts needed to build the classical form-based CRUD (Create-Read-Update-Delete) pages. This satisfied most of their needs without having to write procedural code.

» **Real data** – was available during prototyping which provided progressive evaluation of their work. The end-users could make small changes to their interface (e.g. change filtering conditions or add a new field to the
results) and see the effect of the modification in the context of real data, thus being able to understand if they made a mistake (e.g. wrong data is shown). Furthermore, such incremental development has gentle slope of complexity (Myers et al., 2000) that allows end-user to improve their skills gradually (Repenning and Ioannidou, 2006), taking on more challenging tasks as proceeded with work. Finally, real data supports reflection-in-action (Schön and Bennett, 1996) which allows users to evaluate their design decisions immediately.

» Control over UI – was possible through HTML-like template of the interface allowing for an easy exploration of design alternatives without the need for complex programming. While template interfaces are common in current UI development practice (e.g. HTML for web, XAML for Microsoft .NET framework), at the time Makumba was developed most web development frameworks constructed user interface (at least the dynamic elements) through procedural code, thus providing a challenge for quick prototyping by end-users.

These qualities lowered the threshold for end-users, however, Makumba was built on and closely integrated with “Web 1.0” technology that had low ceiling in regard to the emerging types of user interfaces and interaction (e.g. single-page web application, client side libraries, mobile apps). Therefore, initial proof-of-concept approach was developed and presented in Paper I (and discussed in Chapter 4) where these qualities were used to prototype desktop
interactive applications. We considered the activities of creating a template UI and providing annotations to be low-level modeling activities, thus, we turned to MBUID in order to support prototyping applications of a wider range (e.g. desktop, mobile, web). MBUID employs high-level declarative models that are independent from the underlying technology on which the application will operate (Paternò, 2000). From such models, UIs can be automatically generated for various contexts of use (defined as a combination of device, platform, and environment in which the application is used).

To achieve this we have employed the Discourse-based modeling approach. The initial inspiration for working with the Discourse-based modeling came from our close collaboration with its authors. Our choice was further motivated by the fact that the Discourse-based modeling approach supports iterative design exploration (Raneburger et al., 2014) which is crucial for prototyping. Finally, Discourse-based modeling supported incremental development allowing for progressive evaluation and reflection. Most of other MBUID approaches that work with high-level abstract models require orchestration of several different tools in order to define the high-level model and then generate the final UI from it. However, the Discourse-based modeling tool, the Unified Communication Platform, offers an integrated solution where users can start working from scratch by defining the high-level Discourse Model and generate the UI at any time to evaluate and reflect on their work.

On the other hand, our motivation was to create support for real data and ensure higher control over UI to MBUID approaches (see Figure 5) in order to support prototyping with data in a similar manner that we have seen to be successful earlier. MBUID approaches have not reached wide adoption and techniques developed have mostly been applied in narrow domains and by professional developers (Meixner et al., 2011). One of the reasons

Figure 5: Combining Makumba qualities and MBUID with the goal of achieving a prototyping with data approach that can be applied to prototyping of wider range of UIs
MBUID has been criticized in the past is that they suffer from the problem of unpredictability (Myers et al., 2000). There are certain heuristics involved during the generation process that are often difficult to understand and control, especially for end-users who struggle when working with UI models (Paternò, 2013). This results in low control over the generated UI. Additionally, MBUID is concerned with generating user interfaces and interaction which includes layout, input and output elements, and sometimes states and transitions. However, the underlying application logic and – especially important for this thesis – data access typically still need to be implemented separately, by skilled developers. Both of these (low control of UI and lack of support for real data) were qualities addressed by Makumba and, therefore, the aim was to bring them to MBUID.

The following three sections outline how each of the qualities derived from Makumba (i.e. declarative approach, real data and control over UI) are achieved through the work presented in Paper II and Paper III.

3.2 Fully declarative Discourse Models

In order to achieve a declarative approach, we first aimed to remove Procedural Constructs from the original Discourse Models (Paper II). According to Popp et al. (2009), such constructs had been introduced deliberately, even though they violated the declarative principles of the models, to answer the need for a certain type of interaction. For instance, they were needed in order to define an interaction where the systems should repeatedly request the user credentials until the correct ones have been provided or ten tries have been exhausted. For the purpose of consistency (not mixing procedural and declarative concepts) and with the aim to also support prototyping by users not skilled in procedural programming we strived to achieve a fully declarative high-level model. Paper II presents the result that were achieved by removing the need for such procedural constructs and instead taking advantage of model annotations, defined as logical expressions, to compute potential sequence or repeatability of operations.

The original discourse modeling, as introduced in Chapter 2, uses the high-level Discourse Model to compute an Abstract UI, in other words, a behavioral model representing a state-machine of possible states that the UI can be in
depending on the results of interaction. The state-machine is synthesized solely based on the type of nodes present in the Discourse Model. The other domain-specific information (e.g., annotations present on either nodes or branches) is not used during the computation of the state-machine, but rather employed at later stages, either during the Concrete UI transformation when selecting types of widgets to be shown for each UI element, or during runtime to trigger state transitions. However, as demonstrated in Paper II, these annotations can be used during the state-machine generation as pre-conditions for enabling certain parts of communication and thus still achieve repeatability and sequencing. While the use of pre-conditions has been seen in other MBUID approaches (Mori et al., 2002; Trætteberg, 2009) to our knowledge our approach was the first time that they were analyzed at design time for generating an Abstract UI model (and, subsequently, the Concrete UI model), rather than just at runtime.

The state-machine generated in Paper II from the purely declarative Discourse Model was identical to the one obtained from the Discourse Model with Procedural Constructs of the same interaction. This has given us empirical proof that such an approach is feasible. With this, we have simplified the Discourse Model by removing several constructs while keeping the same expressiveness and, more specifically, we have achieved a fully declarative model. This has the potential to reduce the complexity and lower the threshold when working with high-level Discourse Model in order to prototype interactive applications.

3.3 Real data with Query Annotations

In order to achieve support for real data in the UI generated through Discourse Modeling we applied Query Annotations on the high-level Discourse Model (Paper III).

Query Annotations in Makumba (and Paper I) allowed designers and domain experts to annotate user interfaces with easy-to-learn expressions (SQL-like queries) and thus get a fully functioning interface for displaying or editing data. Furthermore, the domain model was expressed in a declarative domain-specific language that allowed for easy modification of the domain model. On the other hand, the original Discourse-based modeling, as well
as other MBUID approaches (Paternò et al., 2009), supported connection to the traditional web services (e.g. defined through WSDL), which meant that working with real data was possible only if there was a backend functionality implemented. While this might be fitting for the classical Software Engineering practice, with strict and separated cycles of functionality and UI development, the goal of this thesis was to support opportunistic prototyping.

This was addressed (Paper III) by replacing the original annotations with Makumba-inspired Query Annotations. This allowed Discourse-based modeling to reuse the same domain models that were shown to be successful with Makumba (Bogdan and Mayer, 2009), which in turn were directly linked to the datastore. The rest of the generation process was thus modified by having the generated UI components remain annotated with Query Annotations instead of just being connected to the placeholders for the glue-code. These modifications enabled the execution of the queries during runtime, which resulted in generating a prototype that was able to display and edit real data coming from the domain model datastore.

3.4 Towards control over UI

The introduction of Query Annotations to Discourse-based modeling also partially addressed the key quality of control over UI, as detailed in Paper III.

The lack of designers’ understanding of the generation process was seen in the past as one of the causes of the absence of adoption of MBUID approaches (Myers et al., 2000). Indeed, the original Discourse-based modeling approach generates several intermediate models between the high-level Discourse Model and the final UI. Designers have the option to modify these models for fine-grade control over the look of the final UI. While intermediate models might retain some indication of their origin (e.g. having a name that contains the indicator of the high-level model node) they are only tree-based structures of the UI and do not represent visual layout. Therefore, the designer needs to generate the final UI in order to visually inspect and evaluate the prototype. However, at that level there are no more connections to the high-level model available for the designer’s inspection.

By showing the same Query Annotations at the high level and in the final UI (see Figure 6) the designers are able to evaluate how their high-level modeling
affects the resulting UI. The designer can switch between exploration mode (showing the annotations) and data mode (running the query and populating the UI).

Finally, having Query Annotations present on all levels of modeling has the potential to partially address the “round-trip” problem where the modifications to the interface that happen on the lower level (e.g. by modifying the abstract, concrete or even final UI interface) are normally lost when the interface is regenerated from higher-level models. This would allow users to, for instance, modify the generated user interface or any of the intermediate UI models and still be able to go back and work on higher level models, thus supporting the iterative process. As long as the Query Annotations remain the same, the changes would be recognized and a new abstract generation rule could be created, ensuring that any modification on the higher level does not override the changes the user made on the generated UI. The conceptual working of such a process is briefly outlined in Paper III.

3.5 Discussion

This chapter presented the modifications of Discourse-based modeling approach which introduced the key qualities (declarative approach, real data and control over UI) allowing users to prototype interactive applications. However, during these explorations we have come to realize that MBUID
approaches are not suitable for prototyping data-driven interactive applications in an opportunistic manner. As Rode et al. (2005) state, MBUID are designed to address “problems related to productivity, consistency, security and platform independence”, which are more present in professional development process where there are necessary skills and working culture to deal with high-level abstractions. Instead, we seek to support more opportunistic prototyping where functional software is produced rapidly with the purpose of exploring the design space (Brandt et al., 2008).

In what follows we will outline the main shortcomings that we have identified through MBUID exploration as well as new qualities that have emerged as important for supporting the prototyping with data.

**High threshold in regards to models and tools.** The models employed in MBUID are too complex and too high-level for prototyping. Although these models are designed to focus on important aspects of interaction (e.g. tasks or discourse), they are abstract concepts, implying hard mental operations (Green and Petre, 1996), which present a barrier to end-users (although it is probably a lower barrier than programming). Therefore MBUID have a higher threshold to use, especially compared to other prototyping tools in which more concrete concepts (e.g. interface elements) are used. Furthermore, the majority of the recent MBUID approaches have their own dedicated tools supporting the modeling process and dedicated models due to a lack of consensus in the modeling community, even though this has been identified as the main problem with the first generations of tools (Da Silva, 2001; Meixner et al., 2011). The few tools and methods (e.g. WebRatio, Mendix, OutSystems) that have reached a wider adoption are exclusively targeted for enterprise software development which falls outside our desired prototyping scope.

**Low ceiling when it comes to the UI look and feel.** While the quality of control over UI was partially addressed in the previous section, the generation process of the MBUID approaches still limits the control over the UI in the sense of exploration of design alternatives.

In the early days of MBUID research, the UI look was mostly standardized through available widgets for the main computer platforms (Myers et al., 2000). Therefore, MBUID offered a quick way to create usable applications by automatically generating the standard UI. Since then, with the spread of the web and later mobile platforms, the UI design has drastically changed and
become more complex (Dix and Cowen, 2007). Therefore, designers want the possibility to be creative and original. Most MBUID approaches do offer the possibility to apply one’s own graphical styles and modify the generated UIs, however, these options come too late in the prototyping process, i.e. one has to have the interface modeled and generated before one can start adjusting the look. Much like the UI look (appearance), there has been an increase in novelty of the UI feel (behavior) design over the past years. In other words, thanks to touch interfaces, faster devices and various libraries (e.g. D3.js) designers are exploring a wider design space of UI interactions. However, because defining advanced behavior is highly complex they often need to resort to programming to achieve it (Myers et al., 2008). Implementing such novel behavior in the context of MBUID would require end-users not only to engage in EUD, but to do it on the higher level of abstraction so that it could then be used in the automatic UI generation. Such effort would require design and implementation planning effort which end-users are typically lacking (Rode and Rosson, 2003), instead of more opportunistic and iterative development efforts more common for end-users and prototyping in general.

Focus on multiple contexts of use might be less relevant than initially perceived. While the initial motivations for using MBUID was its declarative approach that allowed abstraction over implementation details, several factors came into focus that make such ambitions less relevant. First, we have narrowed our focus on prototyping in a more opportunistic manner where the goal is to quickly produce a functional prototype, rather than be concerned with the scalability of the solution. Second, with mobile and web becoming the main platforms for interactive applications, and web being ubiquitous, there are now dedicated frameworks (e.g. Ionic22, Cordova23) that offer possibilities of creating the app (including the interface) once and then deploying it anywhere (i.e. web or any mobile platform). These frameworks therefore offer the abstraction needed to cover several contexts of use (at least in terms of platform) without requiring higher level abstractions which we recognized as a limitation in the prototyping through MBUID.

22 http://ionic.io/
23 https://cordova.apache.org/
3.6 Conclusion

In an attempt to create support for prototyping of data-driven applications, we have explored the potential of applying qualities from earlier work on Makumba to MBUID approaches (detailed in Papers II and III). While this work has resulted in a proof-of-concept implementation that has technically brought MBUID closer to supporting prototyping with data, the main results were (1) the understanding that MBUID is not suitable for such a task and (2) a refinement of the key qualities of prototyping with data.

The next chapter will present an approach to prototyping-with-data developed from the reflections on the MBUID experience and from additional observations of prototyping practice.
4 Declarative prototyping with data

This chapter presents the proposed approach to declarative prototyping of data-driven interactive applications. First, the qualities of prototyping tools are refined based on the results from Chapter 3, followed by the outline of the conceptual approach to declarative prototyping with data presented in Paper I. Next, the qualities are revised through observations of prototyping in hackathon settings and new solutions in the form of prototyping tools are explored. This is followed by a presentation of the latest implementation, detailed in Paper IV. Finally, the key qualities are compared to the results of study with interaction designers on prototyping with data and on use of the proposed solution (Paper V).

4.1 Towards prototyping with data approach

Bearing in mind the results of the MBUID exploration, detailed in Chapter 3, and by comparison to our initial approach, presented in Paper I, we can revise the key qualities for prototyping with data as shown in Figure 7. First, the quality of supporting multiple contexts of use is dropped due to the change of focus towards quick production of functional prototypes instead of supporting prototyping for various contexts. Second, the control over UI,
that has not been fully achieved (indicated with red X) through MBUID, is further refined into support for prototyping advanced look and feel that affords interaction styles and design present in current practice as well as exploration of new solutions. Finally, a new quality is added to reflect the support for prototyping in the environments familiar to the users.

Let us examine how each of the key qualities is supported by a generalized approach to prototyping with data (detailed in Paper I).

**Declarative approach** is achieved by utilizing declarative concepts both for layout (look) and interaction (behavior). For instance, Query Annotations, seen in Chapter 3 and Paper I and III, are modeled on SQL, a query language designed to be easy to learn and use by non-programmers (Reisner et al., 1975). Apart from Query Annotations, which are used for linking the interface to the data, other annotations are employed to describe interactive actions (e.g. click, drag) and dynamic constraints of the visual layout (Myers, 1991). Such constraints automatically keep the conditions specified and thus reduce the work for the designers when specifying interactive behavior of the system.

Supporting prototyping in **an environment familiar to users** is one of the major differences in the approach proposed here compared to the MBUID approach from the previous chapter. The key concept here is utilization of the UI template for layout prototyping which should be created through the tools and methods familiar to the users. In other words, designers should be able to use tools such as GUI builders and WYSIWYG web editors or to write the plain layout code (e.g. HTML + CSS in the case of web) manually, which is the most popular prototyping method, based on a survey undertaken among more than 4,000 professional designers (Subtraction, 2015).

Furthermore, interaction prototyping (i.e. defining the declarative annotations and expressions) should be supported as well through either the same tools and methods as for layout prototyping. In other words, to support an opportunistic process it is important that the users can seamlessly switch between prototyping the layout and the interaction.

The extent of the support that the environment gives to the user can vary significantly depending on the tool and the method. For example, a basic text editor might allow the designer to define HTML layout and declarative annotations needed for prototyping. However, it would not recognize syntax- or other types of errors the designer might make. On the other hand, a GUI builder might have a plugin that allows users to define the needed annotations
and at the same time evaluates their correctness. While “smart” environments that are able to offer contextual support and guide the users throughout the prototyping have significant benefits, the minimal support expected is the possibility to define annotations on the UI elements. Otherwise, the number of tools and methods available for users to choose from would be greatly limited and thus the goal of a familiar environment would be harder to achieve. However, we do not have to give up fully on smart, contextual help, but can rather move it to the runtime, i.e. to when the prototype is running, as will be shown later.

Support for prototyping with real data has already been achieved in the context of earlier research that included testing the MBUID approach and was presented in the previous chapter. This can be further generalized to fit a wider range of UI prototyping approaches (as detailed in Paper I). The minimum requirements are: the existence of the UI components that can be “populated” with data and the Query Annotations that can be associated with such components. The annotations can then be interpreted (either during design-time or runtime, which is explored in detail in Paper V) and the interface elements can be populated with data coming from the datastore. While many prototyping environments work with the concept of UI components (e.g. GUI builders, graphical mockup tools), some early stage prototyping tools work only with static images (e.g. scanned sketches or screenshots) and thus a more complex solution would be required.

The results presented so far (Papers I and III) gave little attention to the source of the data (i.e. in both cases the proof-of-concepts applications used relational databases). However, in the following sections (and Papers IV and V) the heterogeneity of the data is addressed in greater detail.

Advanced look and feel that affords appearance and behavior present in the current practice aims to provide a prototyping method that allows users to create modern interfaces and interaction. This is partly achieved by giving the users full control over the look (appearance) of the UI. Instead of automatically generating it from declarative models, the users create the UI template using the technology of their choice. This gives them full freedom of using the latest frameworks and components to achieve a modern look.

The second part is achieving advanced feel (behavior) by supporting extensible annotations that can present abstractions for certain interaction and that facilitate a “gentle slope of complexity” (Rode et al., 2005). The
lightweight design of such annotations allows them to bind to any property or event of the UI components, meaning there is no need to extend the annotations or the engine when new components are introduced (e.g. when a framework is updated or third-party components are found). For instance, Paper I presents annotations that work for click or drag-and-drop interaction, allowing users to engage in iterative design and quickly prototype that interaction without needing to resort to complex programming. Gradually, as their skills develop, the users can also extend the annotations. For instance, they can introduce new or more complex behaviors.

The four detailed key qualities were present in the proof-of-concept implementation of prototyping approach presented in Paper I. The key qualities and the proposed approach were inspired by the community where a similar declarative prototyping technology had been developed and used for more than a decade (Bogdan and Mayer, 2009). The next section presents the observations of an emerging prototyping practices and evaluates the key qualities as the result of the observations.

4.2 Prototyping in the wild

While the overall iterative design principles have not significantly changed in the past decade, the technology advancements and new social aspects have introduced new challenges and opportunities that have to be considered when evaluating the proposed approach (Brandt et al., 2008). To understand these challenges and opportunities I have co-organized and observed several hackathon events during which the goal was to prototype data-driven interactive applications.

4.2.1 The study of prototyping in hackathons

Hackathons are events where developers, designers, end-users, and other stakeholders come together to work intensively, typically over a short period of time, to prototype digital innovation (Leckart, 2012). They seem to take place everywhere (Trainer et al., 2016), and are made popular by companies like Facebook, Google, and Spotify to promote their services or test new features. They are increasingly used by governments, researchers, and other stakeholders to evaluate and advance their technical solutions, address social
issues (Zapico Lamela et al., 2013) or foster technical education. The specific aim of hackathons is to produce a working prototype, often utilizing data or technology provided by one of the stakeholders (Van Waart et al., 2015).

The three hackathon events in which I took part were organized in the scope of an EU-funded project focusing on social innovation in the context of energy systems. The participants of the hackathons were given tasks to prototype interactive applications that enabled energy awareness and reduced household energy use. In all of the events they were given access to various social and energy data that they were encouraged to use in their prototypes. My role during those events was that of a mentor and jury member. As a mentor I had direct contact with participants, which gave me a thorough understanding of the technical and practical issues they were struggling with during their prototyping efforts.

4.2.2 Reevaluating the key qualities

The results of study of prototyping in hackathon settings and their influence on the key qualities is summarized in two main observations.

**Everything is a web service.** Web applications, mobile applications, and “smart” devices – somewhere in the background, or rather in the cloud – all connect to web services where the user data is stored. The user interfaces (and their implementations) are decoupled from the data. While such separation of concerns is not new to professional software engineering, it used to be exclusive to them. Building an interactive application that accessed some data required software engineering skills for the orchestration of several components (e.g. databases, SOAP protocols, server-side code, web-servers, etc.). The recent years, however, have seen the rise in popularity of the more user-oriented “Web 2.0” and related REST protocol (Dix and Cowen, 2007; Hagemann et al., 2007). Suddenly a few lines of code were enough to programmatically access the data. However, a new challenge of understanding the data emerged. In the past, people creating the application were also in charge of designing the data. The hackathon participants, on the other hand, spent a good amount of time trying to understand what are the possible APIs, what exactly does the data look like, are there any limitations, etc.

As a result, the key quality of support for real data was redefined as support for **web service data**. This distinction was made, on one hand, to indicate the
importance of web services as the predominant source of data. Even proprietary domain-data in today’s world of mobile and client-side web applications is “hidden” behind web services. On the other hand, the distinction was made to emphasize that there are inherent challenges coming from web service data (e.g. quality, reliability and latency) that need to be addressed in prototyped applications (e.g. by failing gracefully or by showing progress and providing error indicators) and thus supported in the prototyping approach.

The second observation is that setup can present a high threshold, especially when working with data. Some of the hackathon teams failed to produce a working prototype because they spent too much time setting up the development environment. This was especially the case when people with different degrees of technical knowledge needed to work together (Trainer et al., 2016). Prototypes with real data often present an additional challenge. On the one hand, if the prototype needs to store some data, a database is required. In the case of web applications, this meant that the participants now needed to have an “invisible” backend with a database next to the “visible” interactive interface and the data. When prototyping interactive applications, the users should not have to be concerned about the “invisible” but should rather focus on the interface, the interaction, and the data. Furthermore, as it often turned out, while getting the first glimpse of the data was easy (e.g. through a browser or an example link provided in the documentation), there were some challenges that came up during the actual development (the most common among them was Cross-Origin Resource Sharing24), which again required a setup of additional “invisible” backend components. Therefore, the setup-free quality was introduced as a requirement for supporting

Figure 8: Revised qualities for Prototyping with Data after field observations

24 https://www.w3.org/TR/cors/
opportunistic prototyping with data. Figure 8 shows an updated overview of the key qualities.

4.3 Designing a new prototyping tool through exemplars

The implementation of the declarative prototyping approach, presented in Paper I, was an extension to the original Makumba approach with the support for building GUI applications for desktops. However, the observations of the prototyping practice in hackathon settings revealed new qualities which triggered a series of incremental improvements or re-implementations using new technologies. This led to Endev, standing for end-user development, an approach to prototyping with data.

To explore the vocabulary of declarative concepts necessary for prototyping with data, several exemplar prototypes were developed. With each iteration the corpus of exemplar prototypes grew, as each prototype was an example of a real-world application and was used to evaluate different key qualities. A selection of exemplars are explained in the following paragraphs.

Figure 9 show one of the first prototypes, a real-time queue system application, and was used to evaluate and explore alternative syntaxes for

![Queue System](image)

Figure 9: Prototype of a Queue System that can be used by students seeking assistance during lab sessions
declarative annotations. The first variant used the SQL-like query annotation (as seen in Chapter 3, Papers I and III), while the second one used syntax inspired by NoSQL-like APIs (e.g. MongoDB API\textsuperscript{25}) to accommodate diverse type of data (e.g. JSON documents) as well as to resemble JavaScript notation which is typical for web development. Informal evaluation with users resulted in realization that SQL-like syntax presented better \textit{closeness of mapping} to the desired data operation.

Additionally, the example application was meant to be used as a single-page with real-time data synchronization which typically requires complex code to implement and therefore presents an \textbf{advanced behavior (feel)}. Since real-time data is becoming a common feature in modern applications (e.g. live feed or collaborative features like in Google Docs) a new annotation was introduced to allow designers to prototype such features. To achieve the actual data synchronization functionally, the support for Firebase\textsuperscript{26} cloud service – that automatically offers such functionality – was implemented, therefore providing \textbf{setup-free} support for prototyping such data-driven interactive applications. For the designers, the difference between an application sup-

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figures/figure10.png}
\caption{Photo Collector allows the user to build a personal collection of photos by searching through Flickr photos}
\end{figure}

\textsuperscript{25} [https://docs.mongodb.com/manual/crud/]
\textsuperscript{26} [https://www.firebase.com/]
porting and not supporting data synchronization is possible by setting one true or false annotation.

The prototype shown in Figure 10, a photo collection app, was used to evaluate several aspects related to web service data. First, the intention was to support accessing a third-party API (e.g. Flickr), which was achieved by using the YQL27 web service as a proxy for various APIs. This allowed end-users to access different APIs (i.e. not only Flickr) through the same declarative Query Annotations as discussed earlier, This removed the need for writing complex and error-prone “spaghetti code” (Mikkonen and Taivalsaari, 2008). Second, the aim was to be able to combine several data sources, in other words, to store the results from Flickr in the application’s own storage. Since Query Annotations were used for both types of data, combining the data was achieved seamlessly. The details of this prototype are presented in Paper IV.

A prototype of a chess game, shown in Figure 11 and code given in Listing 1, was implemented to explore various aspects related to advanced look and feel. First, the declarative support for utilization of 3rd party library for chess logic was explored by allowing users to include (import) any JavaScript library and to use its methods in the annotations. In other words, the ChessJS28

![Chess Demo](image)

Figure 11: A prototype of the Chess game represents a fully playable application including chess logic that prevents false moves.

---

27 https://developer.yahoo.com/yql/console/
28 https://github.com/jhlywa/chess.js
library could be used to enforce the chess games rules and retrieve information about the status of the game (whose turn it is, the current position of each figure, etc.).

Furthermore, new annotations were added to Endev to support drag-and-drop interaction. These annotations could then be linked to library functions to validate if the move is possible (chess.valid() in Listing 1) and to perform the move when the chess figure is dropped on a valid square (chess.move() in Listing 1).

Finally, by using the data-synchronization feature from the earlier example the prototype became a playable multi-player game (i.e. two users opening the code on different computers would be able to play against each other). Listing 1 shows all of the code (excluding CSS for formatting) an end-user would need to write to create such a prototype.

The final iteration of Endev, outlined in the following section, was influenced by reflections on web development and tech community practice in general. Where possible, the technical aspects were implemented by using currently popular libraries or by employing available online services, while the conceptual aspects, such as syntax and the naming of declarative annotations, were designed as a tradeoff between simplicity and similarity to other aspects.

Listing 1: Prototype code for the chess game shown in Figure 11
the web practitioners might be familiar with (e.g. basic HTML concepts or popular UI design frameworks). These implementation and design choices were made with the aim to produce a solution that supports the quality of a familiar environment (i.e. takes advantage of the user’s existing skills) and facilitates incremental development through “gentle scope of complexity”, thus, creating potential for evolutionary prototyping with data.

4.4 Endev tool for prototyping with data

The details of Endev are presented in Paper IV, while the main concepts are outlined here to show how they relate to the key qualities.

» **Declarative approach:** Endev employs HTML attributes as UI annotations that define data-access queries and data-binding, as shown in Listing 2. The code shown there is the only code needed for making the prototype

```
<hi>ToDoApp</hi>
<h2>Your current tasks</h2>
<div data-from="firebase:TodoList item" data-where="item.done = false"
 data-auto-update="true">
  <input data-value="item.done" type='checkbox'/>
  <input data-value="item.name"/>
</div>
<h2>Create a new task</h2>
<input data-value="newTask"/>
<button data-insert-into="firebase:TodoList"
data-click="insert({name:newTask,done:false})">Add task</button>
```

Figure 12: A prototype of a simple collaborative task planning application implemented with Endev

Listing 2: Prototype code for the task-planning application shown in Figure 12
functional and working with real data, excluding additional basic HTML code for nicer formatting.

» **Web service data:** Endev supports access to any REST web service as well as a variety of other datastores accessed through cloud services. The data access is designed in a modular way so that additional data providers can be added (by skilled developers) and be available through the same queries as all the other services.

» **Familiar environment:** By using HTML annotations to define interactivity and data access, Endev can be used in any HTML authoring tool including various What You See Is What You Get editors that support direct manipulation of the UI elements.

» **Advanced look and feel:** Firstly, through simple annotations Endev supports some of the more advanced interactions (such as drag-and-drop) and functionality such as real-time data synchronization (keeping all the instances of an application in sync when the same data changes in one of them). Furthermore, Endev is built on top of Angular, a popular JavaScript framework, meaning that various components that exist for Angular can be easily used with Endev. Finally, this enabled the evolution of prototype as the developers could gradually replace the Endev annotations with more advanced code.

» **Setup-free:** As mentioned earlier, the code shown in Listing 2 (plus layout and library inclusion code, omitted here for brevity) and a browser is all that is required for running a functioning prototype shown in Figure 12. By using cloud services as datastore, the HTML file representing the code can be run on any device with a browser and it will access the same data and store any changes the user makes. Such a prototype could easily be shared over email, file-sharing services (such as Dropbox) or even put online in one of the code playgrounds (e.g. JSFiddle, CodePen), where it can be viewed directly and modified seamlessly by anybody.

---

29 https://jsfiddle.net
30 https://codepen.io
4.4.1 Evaluation

Endev was initially evaluated with 15 potential users in order to get a better understanding of how it could support rapid prototyping. The study consisted of participants going through an online tutorial in which they learned about the main features of Endev and applied this knowledge to solving several tasks. This was followed by qualitative questions in order to gain a deeper understanding of the opportunities and challenges that the participants perceived from Endev and a questionnaire that was analyzed using the Technology Acceptance Model (TAM) (Davis, 1989) questions to assess the perceived Ease of Use and the perceived Usefulness of Endev in terms of prototyping.

The analysis of answers to qualitative questions and the comments provided during the tutorial (Paper IV) indicate that participants saw Endev as a valuable prototyping tool, especially in regard to qualities of prototyping with data, declarative approach, familiar, and setup-free environment. There were some concerns regarding the support for advanced functionality, but we believe those were mostly due to the fact that the tutorial did not offer any hints on how more advanced features could be achieved (with or without Endev).

Furthermore, the comments also indicated several issues that participants faced when making mistakes, that caused delays and failures in achieving the tasks, which could have been mitigated through better support during prototyping. While such support for other programming efforts is typically available in Integrated Development Environments (IDEs), implementing such an environment would go against the key quality of supporting prototyping in familiar environments. However, because of the forgiving nature of HTML (typically a web page will always render, even if there are syntax mistakes in the code) (Wilde, 2007), a better support could be provided for the users when they run a prototype: mistakes that could be deduced from interface code analysis or during execution (e.g. if a specified web service is not found) could be displayed either in the context (UI element where they occurred) or outside the prototype (e.g. new window, browser console).

Finally, the results of the TAM evaluation, presented in detail in Paper IV, can briefly be summarized as indicating high scores for both Ease of Use and Usefulness. This places Endev in the category of Super tools, as shown in
Figure 13, according to Keil et al. (1995), which are considered to be both powerful and usable, and they have the highest chance of gaining acceptance among users.

4.5 Interaction Designers Prototyping with Data

The results presented so far, inspired by opportunistic settings like hackathons, evaluated Endev as prototyping tool for rapid evolutionary prototyping. On the other hand, prototyping plays an important role in the interaction design, where designers build prototypes (sometimes arguably without programming) to explore design solutions. Therefore, a study was conducted, and detailed in Paper V, to explore how interaction designers approach prototyping with data and to evaluate Endev with designers not skilled in programming.

The results of the study showed that interaction designers consider it important to have real data in their prototypes, but doing so is not trivial for them. The real data is important to inform the design decisions, understand the limitations (“edge-cases that can break the design” as reported by one participant), to communicate with other stakeholders, and in order to have more realistic scenarios when testing with users. However, we have also identified several challenges that interaction designers are faced with when prototyping with real data.

On the one hand, there is the challenge of accessing the data, which can be a result of technological limitations (e.g. lack of appropriate APIs) or policies (e.g. security and privacy concerns). On the other hand, interaction designers
report that there is a lack of appropriate tools or skills for using real data in their prototypes. However, programming is not an appropriate solution, not only because of the lacking skills, but rather because interaction designers want to iterate fast when prototyping. Therefore, they prefer tools where they can quickly explore various design solutions (typically through direct manipulation) in a less formal way.

When it comes to prototyping with data by programming, we have explored the challenges that interaction designers face in their use of Endev. These challenges were analyzed through Cognitive Dimensions (Green and Petre, 1996) which give a framework for identifying usability issues of notational systems. The result of the analysis identified abstraction of data as the main challenge. That is, going from static data – which is manually placed into the design – to an abstraction of data, which is then mapped to the interface. While certain concepts, like data-binding, were easy to understand because of their closeness-of-mapping, concepts that required understanding the data structure or operations for modifying the data were harder to understand as they had no direct representation in the resulting interface.

4.5.1 Reevaluating the key qualities

We can now evaluate key qualities against the presented results (Figure 14). First, the previous refinement of the key qualities went from the generic quality of real data to a more specific version aiming to emphasize web service data. However, the study with interaction designers reveals that, while they do want to work with real data, it is not always available in the form of web services. Therefore, alternative methods like web scraping or manual entry of

![Figure 14: Final qualities for Prototyping with Data after Endev evaluation](image-url)
data might be required. To emphasize that we have refined this key quality to support **heterogeneous data**.

Second, the biggest challenge for interaction designers (and likely other end-users with no programming skills) comes from the lack of understanding of the abstraction of data. This is something that can be addressed through appropriate “abstraction managers” (e.g. data explorer and editor or wizards). Similar to the debugging support presented earlier, these kind of features are normally seen as part of IDEs. However, we can also envision how they could be available as runtime support and thus not conflicting with the flexibility of choosing a familiar environment. Because of this similarity we group the abstraction managers and debugging support together in a new key quality entitled **support tools**.

When we consider **declarative approach**, we are focusing on the activity of programming the prototype that designers engage in when moving from manually entered (direct-manipulation) static content to some automatic way (abstraction) of populating the interface (Blackwell, 2002). Therefore, we speak of about declarative vs. procedural approach to creating the needed abstraction for populating the interface. In that regard, a declarative approach is still considered a key quality as it can be more easily encapsulated in a prototyping tool. For example, dragging-and-droping the collection of data to appropriate interface elements to define data-binding (e.g. as in Craft) is preferred over defining a control structure which will iterate over each item in collection and populate appropriate interface elements (e.g. as in Form).

In regard to remaining qualities, they have been further reinforced by the study with interaction designers. They often use graphic tools for prototyping which at the same time provide a **familiar environment** and **high ceiling** (at least in terms of the look of the UI). Furthermore, due to the highly opportunistic nature of their work, the focus on rapid iteration and their lower technical skills, a **setup-free** environment is crucial.

### 4.6 Conclusion

This chapter presented the key qualities for prototyping with data which were discovered throughout the thesis and motivated the design of Endev. The key qualities were initially built upon the MBUID exploration presented in
Chapter 3 and were refined through several iterations. First, the qualities were refined through observation of prototyping with data in a series of hackathons. Next, a series of exemplars were given that were used to explore the support for the key qualities when building a new prototyping tool. Finally, the key qualities and Endev were evaluated with professional interaction designers.

The identified key qualities, while not necessarily forming a complete list, present major aspects that can enable opportunistic prototyping of data-driven interactive applications.

In the next chapter we shall discuss these results in the general context of the objective and of the research questions stated for this thesis.
5 Discussion

Before the results are discussed in the general context of the thesis, some of the key definitions from the Chapter 1 and 2 are restated. In regard to prototyping of data-driven interactive applications, the focus was on applications that primarily serve the purpose of browsing, exploring, creating, and editing data through modern interaction and functionality that contemporary users expect. This type of prototyping can occur in a professional software development environment, but is also increasingly present in end-user settings, as well as hacker and startup cultures, where less formal processes and methods are in place. This brings about opportunistic design and development efforts that are characterized by little or no upfront planning and focus on speed and ease of development rather than maintainability and robustness (Guindon, 1990; Brandt et al., 2008). Coupled with this opportunistic process are new technologies resulting in more data, services and innovative interaction; together they present new challenges and opportunities for prototyping data-driven interactive applications. The aim of the thesis was to understand these challenges and opportunities in order to support designers in prototyping data-driven interactive applications.

5.1 Answering the research questions

The research objective was formulated as two questions to which the answers are provided in what follows.

5.1.1 What are the current challenges when prototyping data-driven interactive applications?

First, designers lack the skills and tools needed to opportunistically prototype with data, which is considered to be an important part of prototyping the look and feel of interactive applications (Grigoreanu et al., 2009). Paper V shows
that most of the tools that give designers control over the look and feel either produce static prototypes or support only dummy data and basic interaction. This limits the designers possibility to engage in a reflective “conversation with material” (Schön, 1983; Ozenc et al., 2010). On the other hand, tools that might be able to build more interactive prototypes and even connect to real data, like the MBUID approach demonstrated in Chapter 3, do this through heuristic methods that (semi-)automatically generate the UI and thus take the control away from designers. This challenge is, in practice, most often leveraged by designers learning new programming skills and thus engaging in end-user development. While such efforts are becoming increasingly important, as there is a lack of professional developers who could handle the users’ needs stemming from the constantly emerging and changing technology, end-user prototyping tools should be designed with the focus on the intent of the user (Burnett and Myers, 2014) which is to engage in opportunistic prototyping.

Second, working with data often requires complex setups that raise the thresholds for prototyping. Chapter 4 identifies various challenges that designers might face when working with data. For example, interactive applications often need to not only read data, but also to support creating and storing new data. This normally requires a setup of backend technologies that are beyond designers’ skills and, as shown in Chapter 4, present an unnecessary overhead in rapid prototyping (e.g. hackathons) even if they do possess the needed skills. Furthermore, certain technical aspects (e.g. problems of Cross-Origin-Resource-Sharing or web service authentication) might use of backend technologies as well. While mashup research has produced various methods that make the access to data easier for both professional developers and end-users (Hartmann et al., 2007; Lin et al., 2008; Chang and Myers, 2014a), these solutions are mostly focused on achieving the goal of combining the data, while other aspects of the interaction (i.e. the look and feel) are ignored or non existent (i.e. when mashups are just compositions of events and triggers, of various services, without an actual GUI, as in the case of the popular IFTTT\(^{31}\) service).

Third, prototyping modern functional aspects of data-driven interactive applications is hard. The asynchronous nature of modern web and mobile applications, where data is retrieved separately (and typically at a later stage)

\(^{31}\) https://ifttt.com/
from displaying the UI, is something that affects the user experience and therefore prototyping should include the possibility to explore alternative designs that would deal with it. However, implementing such features requires a certain amount of error-prone boilerplate code (Myers, 1991; Mikkonen and Taivalsaari, 2008) that again raises the threshold for prototyping. Another such functional aspect, with even higher threshold, are real-time collaborative features (e.g. Google Docs) which allow multiple users to work on the same data at the same time and see each other’s changes simultaneously. The Endev tool, presented in Paper IV and outlined in Chapter 4, proposes a solution to this as well as previous two challenges.

Finally, **discovering and understanding the data is challenging**. While this challenge was not addressed as part of this thesis, the study outlined in Paper V gives us some insight on it to inspire future work. In the abundance of APIs one of the challenges is finding the right one that might fit the purpose. Although early research on web services did address this problem of discoverability, it was based on the automatic solutions that depended on structured web service definitions. However, the less structured “Web 2.0” (e.g. REST protocol), dominating today’s web services, presents the challenge for discoverability (Hagemann et al., 2007). Assuming that the potential API can be found, understanding exactly what data it gives access to is the next challenge. The documentation might be lacking or it might be very technical, the example data provided might be representing a “perfect case”, while average data might not give as complete results as designers desire, and so on. While there might be other challenges, for example technical ones such as optimization and reliability (Braga et al., 2008), the challenges mentioned here are those that appear first in the opportunistic process of prototyping data-driven applications; they have a higher impact on the interaction design and should, therefore, be prioritized by future prototyping solutions.

### 5.1.2 Which important qualities need to be supported by UI prototyping tools for data-driven interactive applications?

The initial qualities were devised starting from the previous work outlined at the beginning of Chapter 3. Through the iterative work presented in Chapter 3 and Chapter 4 the qualities were refined with the aim of addressing some of the challenges identified in the answer to the first research question.
Together these qualities present a set of features, not necessarily a complete one, for a tool that has low threshold and high ceiling (Myers et al., 2000) for prototyping data-driven interactive applications. They can be summarized as follows:

**Declarative prototyping.** UI prototyping is about representing what the interface will look and feel like without being concerned about how it is going to be implemented. Therefore, declarative approaches, that share the same goal of representing what instead of how, are important for prototyping. Declarative languages are often employed for interface development (e.g. HTML, CSS) and data management (e.g. SQL), and are used by end users with no programming skills.

The benefit of declarative approaches extends beyond just ease of use. For example, Papers I, III and IV have used similar declarative expressions for data-binding features to achieve prototyping with data even though they were used in different tools and targeting different platforms (Java Swing and HTML). Furthermore, Hartmann (2009) argues that the hierarchical structure of UI markup languages (e.g. HTML and HAML) reflects the hierarchical structure of the UI components and is therefore easier to reason about. He also argues that declarative UI code is easier than imperative code to interpret for visual editors, which are important in order to support rapid prototyping.

Finally, in Paper IV we were able to extend declarative HTML, already familiar to many end users, to include additional declarative constructs for the purpose of prototyping with data. This loose coupling feature of HTML is considered important for its success as it allows older browsers to show web pages designed for newer browsers in the best possible way, typically ignoring unknown constructs (Wilde, 2007). Such a feature supports the iterative nature of prototyping as it provides progressive evaluation allowing users to see their results even if their prototype is not yet complete or there might be errors in the prototype code.

**Heterogeneous data access.** Data access is central to prototyping data-driven interactive applications. Early solutions (Makumba, Paper I and III) worked directly with data coming from databases, while Endev (Paper IV) had support for a wide range of data (client-side local storage, web services, and cloud datastore). Regardless of where the data was stored, the principle of accessing and using it during prototyping was the same – query annotations.
While queries may not be the best way to access the data, the goal is to remove the need to learn how different technologies might require different access mechanisms and instead provide seamless access to any data. This includes both lowering the prototyping threshold (through declarative constructs that abstract the *how* and focus on *what*) and supporting the exploration of design alternatives by placing all data on the same level, thus fostering data composition and raising the ceiling. The results of the study presented in Paper V further reinforced the heterogeneous nature of data present during prototyping by indicating how there are additional challenges (e.g. privacy or technology related) that might limit designer’s structured (e.g. through web services) access to data.

**Advanced look and feel.** Today’s UI development practice is changing at a rapid pace. The major mobile platforms are being upgraded on a yearly basis, often including updates to the design guidelines (the *look and feel*) of their UIs. The popularity of JavaScript, the simplicity of HTML and CSS, and the omnipresence of web are regularly producing new libraries and frameworks for UI development (Dix and Cowen, 2007). Designers and developers are constantly adopting these new technologies and learning new skills to keep up with the trends. Technology that is producing outdated design or limiting the control over the UI (as we saw in Chapter 3) is not adequate for prototyping in such a rapidly changing environment. Instead, the vendors of prototyping tools should embrace the new possibilities brought by the new technology; they should abandon the one-tool-fits-all ideal of generality (Daniel et al., 2012), and support prototyping by utilizing tools and methods that the users are already familiar with.

This was addressed with Endev prototyping tool (Chapter 4 and Paper IV) by providing a thin abstraction layer, needed to support opportunistic prototyping, over the existing development technologies (i.e. AngularJS). The layer supported rapid development of various features (heterogeneous data access, form-based and drag-and-drop interaction, real-time data synchronization, etc.) while the user is able to include third party components and libraries to include additional features when needed. Furthermore, it utilized existing web services for data-access features instead of providing only

32 In fact, programming-by-example might be an alternative way.
33 [https://angularjs.org/](https://angularjs.org/)
a “prototypical” functionality which would need to be discarded and replaced if and when the prototype needs to be scaled up.

By aligning the tool’s underlying technology with the UI development practice, the tool can have high ceiling which supports the evolution of both the prototype and of the users’ skills. In other words, it can provide a “gentle slope” between prototyping technology and the one used in practice.

**Familiar environment.** The economic principle of prototyping (Lim et al., 2008) states that the best prototype is achieved in a simple and most efficient way. Therefore, designers should not be required to learn a completely new tool or environment in order to be able to prototype with data. This is further supported by the study of gender in software use which shows that females are less likely to use new software or features to solve their tasks (Beckwith et al., 2006).

The solutions developed in Paper I and Paper IV addressed the quality of familiar environment by utilizing annotations of the standard UI components (i.e. Java Swing and HTML). Therefore, any editor that supports these components and their custom annotation would work (e.g. What You See Is What You Get editors such as Adobe Dreamweaver in the case of HTML). However, several studies with interaction designers (Carter and Hundhausen, 2010; Subtraction, 2015) show they prefer to use graphic design tools over the dedicated UI prototyping tools. Graphic design tools typically do not work with UI components, but rather are based on vector drawings. In such case a dedicated solution (e.g. in form of a plugin) for the specific tool would be required. Such solution might have some trade-offs with respect to other qualities. The review of prototyping tools presented in Paper V shows several plugins, built for a popular graphic design software, that support prototyping with data, however they have low ceiling when it comes to interaction supported interaction with data.

**Setup-free environment.** Apart from the benefit of enabling rapid prototyping, this quality is also aimed at allowing designers to share the prototype with other stakeholders in order to solicit feedback. When the prototype is interactive, includes data, and requires a complex setup, it can be hard to share beyond screenshots or video that capture only a fixed-path interaction. However, with many computing services moving to the cloud and client-side technology – offering rich set of functionality – rapidly emerging,
setup-free prototyping environments can be easily supported. The Endev solution (Paper IV) represents one such environment, where a single HTML file is a fully-interactive prototype with data. Designers can share the file via email, put it in a Dropbox or share it in a code playground (e.g. JSFiddle\textsuperscript{34}, CodePen\textsuperscript{35}). The stakeholders who are asked to feedback it need only the internet and a browser to “run” such a prototype.

**Support tools.** Rapid prototyping of data-driven interactive applications requires programming, which still remains a difficult task for end users (Ko et al., 2004; Cao et al., 2010). One of the reasons identified in the related work is the lack of support in the programming environments to help end users develop problem-solving skills. In the studies conducted in the work presented in thesis (Chapter 4, Paper IV and Paper V) the participants showed similar problems when working in a proposed prototyping environment. While the work presented here does not address these challenges, the importance of addressing them is emphasized by introducing this key quality.

5.2 A brief reflection on MBUID, EUD and SE

The exploration of MBUID presented in Chapter 3 resulted in a shift of focus towards a more opportunistic environment. The solution presented in Chapter 4 therefore fits into the category of End-User Development (EUD) rather than the classical Software Engineering (SE) where prototypes serve to evaluate specifications (Floyd, 1984). This distinction is depicted in Figure 15.

Missing from Figure 15 is an emerging field of End-User Software Engineering (EUSE) (Ko et al., 2011), that would be placed in the area where the orange and blue circle overlap. EUSE aims to address the quality of software produced through end-user development efforts. In other words, EUSE research is concerned with aspects such as testing (Rothermel et al., 2001), debugging (Kuttal et al., 2013) and refactoring (Hermans et al., 2012). While the quality of throw-away prototypes might not be important, evolutionary prototypes are likely to eventually touch upon the concerns addressed by EUSE. Furthermore, we have already seen that lack of focus on debugging presents barriers for prototyping with data (Chapter 4, Paper IV and V).

\textsuperscript{34} https://jsfiddle.net
\textsuperscript{35} https://codepen.io
Therefore, the switch of focus from MBUID should not, in the future work, imply abandonment of all of the SE principles. As Burnett and Myers (2014) argue, EUSE is currently ahead of the classical SE in its focus on supporting the developers’ intents and work style. Some of these practices are likely to benefit professional developers as well as designers engaging in opportunistic prototyping.

An additional argument for transferring EUSE results to SE is the emergence of agile methodologies (e.g. Scrum) that value individuals and interactions over processes and tools (Cockburn, 2002). However, this trend has not been transferred to MBUID, which is still relatively process- and tool-dependent. Many MBUID approaches are top-down approaches that have specialized tools and employ formal “requirements” specification (i.e. abstract high-level models). Therefore, the question of the future role of MBUID still remains open.
6 Conclusion

The research reported in this thesis started off as an engineering effort to create tools for building data-driven interactive applications. Through the iterative process of designing, building, and reflecting on the results the focus gradually shifted from engineering to opportunistic prototyping. The shift was inspired by the observations of the emerging settings, such as hackathons and startups, in which speed, ease of development and focus on interactivity of the prototyped solutions are more important than performance and robustness. At the same time, while the focus on data-drive applications stayed, the opportunistic setting brought the broader about a definition of data and opened up the design space.

As the main result of the thesis several challenges for prototyping with data were identified and key qualities of the tools aiming to address them were defined. While neither the challenges nor the qualities present a complete list, they provide design implications for future prototyping tools and methods.

6.1 Future work

One of the aims of future work will be to address the programming barriers that designers might encounter when prototyping with data. Possible solutions in that direction, without making a trade-off with other qualities, could be better error detection and reporting (e.g. in case of wrong syntax or errors from the web services), designing visual editors for concepts like data-binding and specification of interactive aspects, and focusing on the improvement of problem-solving skills of the designers.

We could build upon ideas from problem-solving theories and solutions, such as Idea Garden (Cao et al., 2015), that offer suggestions in the form of problem-solving strategies that are designed to improve users’ problem-solving skills of the users. The learning could further be fostered by crowd-sourced repositories of examples that have been explored in other EUD
work (Rode et al., 2005; Hartmann et al., 2007). Instead of designing new prototyping environments, we could explore the possibility of using the existing code playgrounds for such sharing. Popularity of code playgrounds has had a significant impact on software development support because they allow developers that come across a specific problem to get help from the community by providing an example in the playground which is then solved by the community (Nasehi et al., 2012). Such approach would be inline with the results of the thesis as it would incrementally build on the current design and development practice.

From the perspective of prototyping, there are other aspects beyond displaying and modifying data that could be explored. One such aspect is data processing (e.g. filtering and aggregation) which is part of the emerging topic of data literacy (Carlson et al., 2011). The future tool for prototyping with data could be designed to allow designers to not only create a prototype, but also to develop their data processing skills.

Functional aspects like authentication, or email and push notifications are of also interest because they have impact on the interaction design and on the overall user experience of the prototype. Similar to data persistence, there are cloud services that could be used to “outsource” such functionality and we could focus on designing declarative abstractions that represent the functionality in a way meaningful to designers.

Throughout this thesis we have only looked at the data that is interesting to the user of the prototyped application. However, usage and analytics data can play an important role in prototyping, especially in the case of live prototyping (Aycan and Lorenzoni, 2015). Designers could acquire new insights and understanding by “seeing”, through such data, how their prototype is used. Also, the challenge here is finding an adequate abstraction to seamlessly integrate this into the prototyping activity.

Finally, in relation to MBUID research, a possible future work could also be to explore the path from opportunistic prototyping to MBUID. In other words, to find out if declarative annotations of a prototype created with Endev could be used to generate an abstract model of the interaction. This could create a path to a more formal SE, or rather, to EUSE, in the later stage of the development process.
7 Bibliography


