A Separation-Based UI Architecture with a DSL for Role Specialization

A dissertation submitted in partial fulfillment of the Requirements for the degree of Doctorate of Philosophy in Computer Science and Engineering

by

Ivan Gibbs

Dr. Sergiu Dascalu - Dissertation Advisor

December, 2013
We recommend that the dissertation prepared under our supervision by

**IVAN GIBBS**

entitled

**A Separation-Based UI Architecture With A DSL For Role Specialization**

be accepted in partial fulfillment of the requirements for the degree of

**DOCTOR OF PHILOSOPHY**

Sergiu Dascalu, Ph. D., Advisor

Frederick C. Harris, Jr., Ph. D., Committee Member

Dwight Egbert, Ph. D., Committee Member

Yantao Shen, Ph. D., Committee Member

Sherif Elfass, Ph. D., P.E., Graduate School Representative

Marsha H. Read, Ph. D., Dean, Graduate School

December, 2013
Abstract

This dissertation was aimed at creating a new software development methodology centered on the natural role specialization characteristic of people. Based on existing work in the field, we propose a unique separation-based UI architecture focused on specialized roles, namely user experience (UX) professionals and computer programmers. A distinguishing characteristic of our approach is the use of a domain specific language (DSL) to bridge the gap between the two roles’ knowledge domains, to accomplish higher-level abstraction, and to free the user-interface (UI) designer from having to learn and master a programming language. Furthermore, we rely on model-driven engineering (MDE) techniques of code generation to further enhance the benefits of using a DSL. In this dissertation we describe a complete methodology that implements our architecture, reaps the benefits of role specialization, takes advantage of DSL-based abstraction and automated code generation, and provides step-by-step practical guidance in a variety of software development applications that contain UI components. To illustrate our approach, an NetCDF file subset extractor application built using the proposed methodology is also presented. The proposed method is compared with related approaches and the results of applying it are evaluated using a usability study, an analysis of automated code generation efficiency, and a comparison with several tools that could have been used to create a similar software application to our NetCDF subset extractor program. Background details on domain modeling, model driven engineering, and user experience and user interfaces, together with pointers to several directions of future work are also included in the dissertation.
I would like to thank my advisor Dr. Sergiu Dascalu for the support and guidance in this endeavor. I would also like to thank my committee members (Sergiu Dascalu, Frederick C. Harris, Jr., Dwight Egbert, Yantao Shen, and Sherif Elfass) for serving on my committee. I also would like to thank my parents for their encouragement. In addition, I attribute my economic ability to go to graduate school being largely due to the generous jobs that I worked at and my frugality and long term planning taught to me by my parents.

This material is based in part upon work supported by the National Science Foundation under grant number EPS-0814372. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.
Contents

List of Tables x

List of Figures xii

Chapter 1  Introduction 1
  1.1  Motivation .................................................. 1
  1.2  Overall Approach ............................................. 2
  1.3  Dissertation Structure ....................................... 3

Chapter 2  Domain Modeling 5
  2.1  Introduction ..................................................... 5
  2.2  History ........................................................... 6
  2.3  Description ....................................................... 7
  2.4  Successes ......................................................... 9
  2.5  Challenges ....................................................... 10
  2.6  Research Today ............................................... 10
  2.7  Resources ....................................................... 13
  2.8  Conclusions ................................................... 13
Chapter 3  Model Driven Engineering

3.1 Introduction ........................................ 15
3.2 History ............................................. 16
3.3 Description ......................................... 18
   3.3.1 Graphical MDE ................................. 19
   3.3.2 Textual MDE .................................. 21
   3.3.3 Transformations ............................... 23
3.4 Luminaries ........................................... 25
3.5 Successes ............................................ 26
3.6 Challenges .......................................... 27
3.7 Research Today ...................................... 30
3.8 Resources ............................................ 31
3.9 Conclusions ......................................... 32

Chapter 4  User Experience and User Interfaces

4.1 Introduction ........................................ 35
4.2 History ............................................. 36
4.3 Description ......................................... 36
   4.3.1 Design Ethnography ........................... 42
4.4 Successes .......................................... 43
4.5 Challenges .......................................... 44
4.6 Research Today ...................................... 44
4.7 Resources ............................................ 45
4.8 Conclusions ......................................... 46
Chapter 5  Approach: User Experience Meets Model Driven Engineering

5.1 Introduction ................................................................. 47
5.2 Derivation ................................................................. 47
  5.2.1 Role Specialization .................................................. 48
  5.2.2 Communication Gap ................................................ 52
  5.2.3 Multiple Communication Gaps .................................... 54
  5.2.4 How to Bridge the Gap? ............................................. 56
5.3 Description of the Proposed Approach .............................. 59
  5.3.1 Code Creation ....................................................... 62
  5.3.2 UX Creation .......................................................... 63
  5.3.3 Steps of Implementation ......................................... 71
  5.3.4 Conclusions .......................................................... 71

Chapter 6 Application Specification and Design ...................... 73

6.1 Concept ................................................................. 73
6.2 NetCDF ................................................................. 74
6.3 Persona ................................................................. 76
6.4 Requirements ........................................................... 76
6.5 Use Cases ............................................................... 79
6.6 Technological Choices ................................................ 84
6.7 High Level Design ..................................................... 84
6.8 Medium-Level Design ................................................ 86
6.9 Data Design ............................................................ 87
6.10 Detailed Design ....................................................... 90
Chapter 7 Applying the Methodology

7.1 Introduction ................................................. 99
7.2 Step 1: Identify the gaps ........................................ 99
7.3 Step 2: Identify the knowledge domains ......................... 101
7.4 Step 3: Create a DSL for the domains to talk to one another .......... 102
7.5 Step 4a: Creating the UI ....................................... 103
  7.5.1 User Testing ............................................ 107
7.6 Step 4b: Creating the code ..................................... 107
7.7 Step 5: Integration ............................................ 109
7.8 Conclusion .................................................. 111

Chapter 8 Evaluation of the Methodology and Related Work

8.1 Introduction .................................................. 114
8.2 Other Comparable Approaches ................................ 114
  8.2.1 Plastic User Interfaces .................................... 115
  8.2.2 Usability-Supporting Architectural Patterns ................. 116
  8.2.3 Game Maker Language .................................... 118
  8.2.4 Linden Scripting Language ................................. 119
  8.2.5 User Interface Markup Languages ......................... 121
  8.2.6 Intentional Programming ................................ 124
  8.2.7 Separation-Based Architectural Patterns .................. 125
  8.2.8 GUI Builders ........................................... 126
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2.9</td>
<td>Other Research</td>
<td>126</td>
</tr>
<tr>
<td>8.2.10</td>
<td>Comparison</td>
<td>128</td>
</tr>
<tr>
<td>8.3</td>
<td>Discussion: Benefits, Drawbacks, Other Opportunities, and Impacts</td>
<td>131</td>
</tr>
<tr>
<td>8.4</td>
<td>Conclusions</td>
<td>133</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Evaluation of the Application</td>
<td>134</td>
</tr>
<tr>
<td>9.1</td>
<td>Introduction</td>
<td>134</td>
</tr>
<tr>
<td>9.2</td>
<td>Informative Usability Study</td>
<td>134</td>
</tr>
<tr>
<td>9.3</td>
<td>Usability Analysis</td>
<td>135</td>
</tr>
<tr>
<td>9.4</td>
<td>Code Generation</td>
<td>136</td>
</tr>
<tr>
<td>9.5</td>
<td>Comparative Analysis</td>
<td>136</td>
</tr>
<tr>
<td>9.5.1</td>
<td>ncdump and ncren</td>
<td>140</td>
</tr>
<tr>
<td>9.5.2</td>
<td>MATLAB</td>
<td>141</td>
</tr>
<tr>
<td>9.5.3</td>
<td>EverView Slice and Dice Tool</td>
<td>142</td>
</tr>
<tr>
<td>9.5.4</td>
<td>Ncview</td>
<td>142</td>
</tr>
<tr>
<td>9.5.5</td>
<td>Data Basin</td>
<td>145</td>
</tr>
<tr>
<td>9.5.6</td>
<td>Interactive Data Language</td>
<td>147</td>
</tr>
<tr>
<td>9.5.7</td>
<td>Comparison</td>
<td>149</td>
</tr>
<tr>
<td>9.6</td>
<td>Discussion: Benefits, Drawbacks, Other Opportunities, and Impacts</td>
<td>152</td>
</tr>
<tr>
<td>9.7</td>
<td>Conclusions</td>
<td>153</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>Conclusions and Future Work</td>
<td>154</td>
</tr>
<tr>
<td>10.1</td>
<td>Introduction</td>
<td>154</td>
</tr>
<tr>
<td>10.2</td>
<td>Conclusions</td>
<td>155</td>
</tr>
<tr>
<td>10.3</td>
<td>Future Work</td>
<td>157</td>
</tr>
</tbody>
</table>
Appendices

Appendix A  NCCP Usability Study

A.1 Introduction ................................................. 160
A.2 Details ....................................................... 161
A.3 Quantitative Analysis ................................. 165
  A.3.1 Overall SUS .......................................... 165
  A.3.2 Timing Test Results ................................. 165
  A.3.3 Completion Rates ................................. 167
  A.3.4 Correctness Rates ................................. 168
  A.3.5 Mouse Measurement ............................... 169
A.4 Qualitative Analysis ................................. 170
  A.4.1 Contrast ............................................. 170
  A.4.2 “Download Data” Button ......................... 170
  A.4.3 Query Errors ....................................... 171
  A.4.4 Map location ....................................... 171
  A.4.5 Aggregate Data Every box ....................... 174
  A.4.6 Choosing Temperature ............................. 174
  A.4.7 Small Screen Error ............................... 177
  A.4.8 Zero Records Error ............................... 178
  A.4.9 Gray Out of Start Time ......................... 180
  A.4.10 Numeric Keypad Failure ....................... 180
  A.4.11 Location Obfuscation ......................... 180
  A.4.12 Download to a File ............................. 180
  A.4.13 Download Link Failure ....................... 181
List of Tables

5.1 Scores of different professions [32] .................................. 51

6.1 Climate scientist persona ................................................. 77
6.2 Functional requirements .................................................. 78
6.3 Non-Functional requirements ........................................... 79
6.4 Use case descriptions ..................................................... 81
6.5 Use case 1–detailed .......................................................... 82
6.6 Use case 2–detailed .......................................................... 83
6.7 Functional requirements–use case traceability matrix ............. 84

7.1 Classifying steps into domains ........................................... 102
7.2 Designing a DSL ............................................................. 102
7.3 GOMS Analysis .............................................................. 107
7.4 The formalized DSL ........................................................ 108

8.1 Approach descriptions ..................................................... 129
8.2 Approach comparison .................................................... 130

9.1 Parser code generation .................................................... 137
9.2 Code generated for org.xtext.demo.mydsl.ncDsl .................. 137
List of Figures

1.1 Separation Based UI diagram ........................................... 2
2.1 Modeling diagram .......................................................... 7
2.2 A sketched domain model ................................................. 11
2.3 An informal domain model .............................................. 11
2.4 A formal domain model .................................................. 12
3.1 Abstraction layers .......................................................... 20
3.2 A UML class diagram ..................................................... 20
3.3 An internal DSL [64] ....................................................... 21
3.4 A script in Graphviz ....................................................... 22
3.5 A large graph from Graphviz .......................................... 22
3.6 Seasons diagram ............................................................ 23
3.7 A complex mapping of PIM to PSM (IM = Intermediate Layer) . 24
3.8 A diagram in MetaCase [156] ........................................... 33
4.1 User interface interaction model ....................................... 36
4.2 Remote control duplication ............................................ 38
4.3 Usability professionals design poster .................................. 40
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6</td>
<td>A sketch of the text-oriented interface</td>
<td>106</td>
</tr>
<tr>
<td>7.7</td>
<td>Xtext grammar</td>
<td>110</td>
</tr>
<tr>
<td>7.8</td>
<td>Xtext template</td>
<td>110</td>
</tr>
<tr>
<td>7.9</td>
<td>DSL script</td>
<td>111</td>
</tr>
<tr>
<td>7.10</td>
<td>Generated Code</td>
<td>112</td>
</tr>
<tr>
<td>7.11</td>
<td>A script, original.ncdsl, for our DSL</td>
<td>112</td>
</tr>
<tr>
<td>7.12</td>
<td>Command to create generated java files from original.ncdsl</td>
<td>112</td>
</tr>
<tr>
<td>7.13</td>
<td>Generated java files</td>
<td>113</td>
</tr>
<tr>
<td>7.14</td>
<td>Compiling generated java files</td>
<td>113</td>
</tr>
<tr>
<td>7.15</td>
<td>Running the final program</td>
<td>113</td>
</tr>
<tr>
<td>8.1</td>
<td>CAMELEON reference framework for the development of Plastic User Interfaces [36]</td>
<td>117</td>
</tr>
<tr>
<td>8.2</td>
<td>GML example</td>
<td>119</td>
</tr>
<tr>
<td>8.3</td>
<td>LSL example</td>
<td>120</td>
</tr>
<tr>
<td>8.4</td>
<td>Snow example</td>
<td>123</td>
</tr>
<tr>
<td>8.5</td>
<td>Pension Workbench</td>
<td>125</td>
</tr>
<tr>
<td>8.6</td>
<td>A GUI builder for thin clients</td>
<td>127</td>
</tr>
<tr>
<td>9.1</td>
<td>sample.nc contents</td>
<td>141</td>
</tr>
<tr>
<td>9.2</td>
<td>Using ncdump</td>
<td>141</td>
</tr>
<tr>
<td>9.3</td>
<td>Using ncgen</td>
<td>142</td>
</tr>
<tr>
<td>9.4</td>
<td>Matlab code for opening and inspecting a NetCDF file</td>
<td>143</td>
</tr>
<tr>
<td>9.5</td>
<td>Matlab code to read a NetCDF file</td>
<td>144</td>
</tr>
<tr>
<td>9.6</td>
<td>EverVIEW file contents view</td>
<td>144</td>
</tr>
<tr>
<td>9.7</td>
<td>Ncview command screen</td>
<td>145</td>
</tr>
</tbody>
</table>
9.8 Ncview ‘precip’ plot ........................................ 146
9.9 Ncview ‘precip’ detailed graph ............................. 146
9.10 Data Basin import settings, page 1 ....................... 147
9.11 Data Basin import settings, page 2 ..................... 148
9.12 Data Basin results ........................................... 148
9.13 Interactive Data Language sample ....................... 150

A.1 The NCCP homepage ........................................ 162
A.2 The text-based search page for the NCCP ............... 163
A.3 The map-based search page for the NCCP ............... 164
A.4 SUS scores .................................................... 166
A.5 Start Time >End Time ...................................... 172
A.6 Download dialog ............................................. 173
A.7 0.5 Hours error .............................................. 175
A.8 Adding temperature adds many checkboxes ............. 176
A.9 From Mac Airbook 1440x900 resolution ................. 177
A.10 Beginning of download progress box .................... 178
A.11 Later picture of download progress box ................. 179
A.12 Occlusion of Snake Range Subalpine (West) .......... 181

B.1 The text-oriented search page for the NCCP .......... 184
B.2 The search page subsetting NetCDF files ............... 185
B.3 SUS scores .................................................... 187

C.1 Disciplines for Search User Interfaces [196] .......... 194
C.2 Stages of user search ...................................... 194
Chapter 1

Introduction

1.1 Motivation

In the world of software development there are many deadlines, dead ends, long hours, and other difficulties. We believe that a large amount of the accidental complexity [22] contained in the development of a software project lies at the boundaries between machine programmers and other high level designers. While the traditional view of engineering teaches us that the designers create a specification and the programmers implement that specification, the reality of real-world software development has to deal with changing specifications and therefore requires substantial and frequent interaction and exchange of information between designers and programmers. This exchange is hampered by a communication gap between those two groups and we have identified this accidental complexity and attempted to rectify it in our approach.

In this dissertation, we address the UI–Code interface of the traditional Separation Based UI architecture (Figure 1.1) in order to simplify that interface and thereby alleviate a number of difficulties pertaining to developing software. In contrast to
other approaches, we attempt to determine the design of the UI–Code interface by basing it on specialized roles rather than solely on the code and some principle such as don’t repeat yourself (DRY).

1.2 Overall Approach

For the world of software engineering, there are many methodologies that have been proposed and we have used several that currently exist. We combined already existing methods and technologies in order to fashion our generalized approach to application development. It is through this combination of tools and techniques that we have achieved a number of positive results. Some of the techniques used are Interaction Design (IxD), Domain Specific Language (DSL)s, code generation, Graphical User Interface (GUI) builders, and Integrated Development Environment (IDE)s. And though all of these tools have existed for long periods of time, we found that our DSL tool excluded the ability to work with grammars for binary files and that there is a dearth of GUI builders which do not require the user to know how to write software. Due to time constraints, we focused on the tools as they exist today and did not attempt to greatly modify them.

Based on the traditional architecture shown in Figure 1.1 we designed a new
architecture that emphasizes specialized roles involving User Experience (UX) professional designers and programmers (throughout this dissertation we refer to the group of UX professionals and programmers as developers). The architecture and the methodology based on it benefit from automated code generation and are generic and flexible enough to be applied in numerous software development projects. Our main premise is that specialization, combined with a mechanism for integration and bridging knowledge domains such as a DSL can be particularly effective in software applications with significant User Interfaces (UIs).

Herein lies the work that is the result of several years of study and creative ideation. This dissertation provides a structured presentation of that work written to communicate the prominent details to the reader. We hope that the reader is interested in browsing the work and picking up on the unique characteristics of our approach without necessitating that the entire dissertation be read from front to back. Liberal use of pictures in the dissertation was intended to enhance textual descriptions and to provide the reader with an abundance of rich information at a glance.

1.3 Dissertation Structure

The layout of this dissertation is separated into descriptions of background concepts, the presentation of our approach, a project created with our approach, the inclusion of an evaluation and a comparative analysis, and a conclusion. Chapters 2 - 4 describe background concepts and are laid out in a similar manner, intended to introduce the reader to these concepts in a systematic way. Chapter 2 explains the area of domain modeling used by many in the software field. The area of Model-Driven Engineering (MDE) is described in Chapter 3. The field of UX is explained in Chapter 4. As our
approach is largely focused on the differences between programming and UX, these background chapters provide information to delineate our view of these fields in order to contrast them later. Chapter 5 contains an outline of our approach and identifies our main contribution of merging the worlds of UX and Model Driven Engineering in a way that supports our focus on the UI-Code interface. Chapter 6 contains a detailed description and design of how we applied our approach to a real-world project in the area of climate science. The implementation of our approach on the climate science application is described in Chapter 7. An evaluation of our approach and a comparison with related work are presented in Chapter 8. The application artifact created with our approach was analyzed and the results are shown in Chapter 9. We conclude our presentation with Chapter 10—conclusions and future work. In order to provide some more in depth material that was not necessarily prominent in our approach, we have also included several appendices. Specifically, a usability study that informed our work is contained in Appendix A, a second usability study that looked at the results of applying our method is provided in Appendix B, and details regarding UIs for the searching of data are presented in Appendix C. In addition to these, we have included a Glossary to elucidate our intended meaning of the terms used in this work.
Chapter 2

Domain Modeling

2.1 Introduction

In the world we have a lot of complexity—so much so that it is overwhelming to humans. Much psychology supports the fact that our conceptual construction of our worlds are incomplete and limited [102]. Today, it seems that humans deal with this complexity with simplifications, metaphors, generalizations, assumptions, abstractions and the like. Although this complexity seems obvious once it is revealed, we generally do not acknowledge that the world is so complex and feel satisfied with our simplification mechanisms. Mathematics is an endeavor which deals with solving isolated particular problems and describing those problems and solutions in a formal and abstract manner. After gathering together solutions to different problems of a specific area, abstractions may be classified and categorized. Subsequent to this abstraction classification, we can combine abstractions to create models of the specific area of concern. These models can then become quite useful in our ability to tackle the complexities of a particular part of the world. In other words, the combinatorial
ability of abstractions allows us to expand our abstract problem solving abilities using a hierarchical approach to our world of complexity. A domain model is a model that describes the concepts in a problem domain and the relationships among different concepts. It captures the necessary information, relationships, and constraints presented by the problem domain and reduces them to a vocabulary of terms and key concepts. Once the domain model is created, it can be used as a common representation of the problem domain and thereby facilitates communication and understanding.

2.2 History

It seems that the history of modeling is longer than recorded human history. Storytelling is a crude example of a model where the story relates a number of events, but focuses on some specific area of life such as the Grimm Brother’s fairy tales which have been used to teach moral principles, cultural ideals, and obedience to children. Another example is the use of painted pictures to describe certain times, emotional states, or ideals. Some models of history even seem humorous and ignorant to us today, such as the model of a flat earth in comparison to our current concept of a spherical globe. This globe model also points out to the simplification in that our globe is not a perfect sphere as it is shown with the globes of today. One other area of perennial interest is the various models of the human diet throughout history in relation to health and sickness. Herbs and natural remedies for health are ancient, but in the light of current science many of them have been shown to be harmful thereby motivating a reevaluation of these ancient beliefs. Therefore, there are many simplified models of reality in human history and we have carried many of them into our times today. In regards to the field of engineering, models are mainly used to guide us on
how to construct something before constructing it (Figure 2.1). Civil engineers make models of bridges and then test them for strength, they build other bridge models to show to the public in order to judge the aesthetic appeal, and they use bridge models to evaluate the effects of earthquakes and other natural disasters. Physicists often build models in order to test their theories about the world. Software engineers use prototypes to test user appeal, elicit requirements, measure performance, and explore failure scenarios.

2.3 Description

Models are a simplification of reality and therefore have a number of limitations regarding their applications. The benefits of creating them are to use an easy to construct model to identify and resolve questions before spending more resources on creating a real artifact. The flexibility in the construction of the model allows this method to address a number of different aspects of the actual artifact. In engineering, the models generally have some mathematical basis, but they also contain a certain amount of assumptions as well. Models can be used in areas in which certainties are nonexistent (i.e. the emotional impact of video games) or elusive in order to increase the confidence in achieving a specific result.

In order to create a model via domain analysis, one could follow a number of steps. A first step is to identify the concepts used in one’s domain by collecting the
terms used by people when discussing the domain. These words will represent the
domain vocabulary for the next step which is to identify the relationships between
these words. The vocabulary and relationships can then be depicted with a diagram
in order to provide a conceptual map of the domain. One idea is that these concepts
can give guidance to the software design modularity concerns and therefore will re-
sult in better modularity decisions to enhance the understandability, flexibility, and
maintainability of the final software design. Another part of the process which the
domain model is purported to assist with is in creating a way to communicate with
stakeholders and thereby improving the requirements for the project which enhances
all further stages in the development cycle.

In software engineering, modeling has shown itself to be an effective technique
for analysis, planning, or development. GUI prototyping models the GUI for the pur-
pose of evaluating stakeholder requirements or technical constraints. Formal methods
use mathematical models to determine the characteristics of a design, such as robust-
ness or fault tolerance. User stories provide a model to build use cases and to then
derive requirements from those use cases. Emulation allows developers to work on a
modeled platform such as using a PC to develop and test software for an Android
phone or Programmable Interrupt Controller (PIC). Domain modeling is also impor-
tant in software methodologies such as Feature Driven Development (FDD) [30, 150]
and Domain Driven Design (DDD) [45].

Like many things, modeling may not be very useful in specific cases. If model-
ing is described as a limited but clear representation of a more complex artifact, then
in areas that already have clarity, modeling is of limited use. Attempting to make
one’s model too comprehensive can also defeat the benefits of modeling and bog one
down with the complexities that have contaminated one’s model. Modeling requires
the ability to understand the important parts of the artifact to model and if one lacks
the skill to do this for a particular area, one’s modeling will not be very successful.

Lest we believe that modeling is simple, we can take note of how long it could
take to discover a relevant model. In Chemistry, Hennig Brand first discovered an
element in 1649, and it took until 1862 before the first periodic table of elements
was created by A. E. Beguyer de Chancourtois (who was a geologist). And, it was
not until 1869 that Dmitri Ivanovich Mendeleev published a paper containing our
contemporary periodic table. On a large temporal scale, the software engineering
field has just gotten started and we are likely to have more and more sophisticated
models as a result of research and experimentation.

2.4 Successes

There are some significant successes in the area of modeling, as it pertains to software
engineering. Virtual machine technologies have been used for a long time in the
area of software and system development and can be considered to be a success in
the modeling arena. Formal methods have been successful in specific areas [199]
and object oriented methods have found a fair portion of success. Until we have
completely describable software through mathematically based foundations, the area
of modeling will continue to be important. Models that programmers interact with
often are Structured Query Language (SQL), HyperText Markup Language (HTML),
and the programming constructs from the realm of structured programming.
2.5 Challenges

The concept of modeling contains a number of challenges today. One problem is the need to keep the model in sync with the artifact that it models. This is especially true if the artifact is subject to updates or trends. Since a model is a simplification, there are many decisions to be made regarding what is to be extracted from the artifact and incorporated into a model. While this problem is made worse the more complex the subject matter is, a general strategy is to make models address a small part of the artifact in order to simplify those decisions. Due to the abstract nature of software, we are often faced with choosing the types of building blocks to create our model from, and this has led to domain-specific modeling concepts, which are intended to allow us to use domain building blocks to create our models. Additionally, there does not appear to be a consensus on which way it is best to build a model, top-down or bottom-up. The drawing of models generally follows a progression of sketched models (Figure 2.2), informal models (Figure 2.3), and then formal models (Figure 2.4), but which is appropriate and when is it appropriate? We also have to deal with politics, since modeling generally involves a group of people in a complex area. Can models be reused, can the components of models be reused, and what is the best form of models for the purpose of reuse? Though our listing here is not comprehensive, it does illustrate many of the current difficulties involved in modeling.

2.6 Research Today

Part of research today on modeling (as it relates to software) involves the implementation of the concept in different domains [8, 110, 153, 162] in order to assess its usefulness. Others are working on creating modeling languages to enhance the ability
Figure 2.2: A sketched domain model

Figure 2.3: An informal domain model (Picture accessed from http://www.archive.arm.gov/Carbon/dataneeds/dataneeds.html on 10/2013)
Figure 2.4: A formal domain model (Picture accessed from http://www.unidata.ucar.edu/software/netcdf/workshops/2008/netcdf4/Nc4DataModel.html on 10/2013)
for applications to inter operate [40, 61, 62]. Some researchers are focused on creating tools to assist with modeling [121]. The ability to evolve models as new requirements become evident is also being studied [115, 116, 132]. Approaches to domain modeling also receive a fair amount of research [7, 38, 85, 112]. Graphical modeling languages and their usability are also being studied [168].

2.7 Resources

The concept of domain analysis and design is not new and therefore many resources exist. Eric Evans has created a popular book on the subject in the area of object-oriented design and he explains what needs to be done and how to do it [45]. Vaughn Vernon wrote a book which instructs developers on how to apply domain modeling to enterprise applications [189]. Another book by Kelley and Tolvanen provides actual real-world examples of domain modeling projects [108]. There is a website [70] that posts case studies, videos, discussions, books, and examples of domain modeling. As patterns are popular, there is the Domain Model Pattern for domain modeling and its anti-pattern the Anemic Domain Model Anti-Pattern [145]. A number of software tools allow developers to create Unified Modeling Language (UML) diagrams to represent their models.

2.8 Conclusions

Domain modeling allows designers to better understand and formulate the requirements of their project. Additionally, the domain model may be used for verification and validation at the design stages of development. One of the major benefits of this
technique is increased communication with stakeholders and another benefit is the clarity reached through testing a domain model in the design stages and thereby improving early removal of problems. Though research is ongoing, the area may benefit by gaining wider and more comprehensive adoption of its different techniques among software practitioners.
Chapter 3

Model Driven Engineering

3.1 Introduction

In regards to this dissertation, MDE\(^1\) is a software development technique which attempts to allow developers to work at a higher level of abstraction than Third-Generation Languages (3GLs). The buzzwords of Model Driven Architecture (MDA)\(^2\), Model-Driven Development (MDD)\(^3\), Model-Driven Software Development (MDSD)\(^4\), and others all refer to particular implementations of MDE. However, despite the many different names in this sphere, there is still much ambiguity in these monikers. Questions that exist in these different methodologies are:

- Are the models only used for better conceptualization (i.e. to find correct abstractions) of the design and then discarded?

- Are the models used to auto-generate code?

---

\(^1\)http://en.wikipedia.org/wiki/Model-driven_engineering
\(^2\)http://en.wikipedia.org/wiki/Model-driven_architecture
\(^4\)http://en.wikipedia.org/wiki/Model_Driven_Software_Development
• Is verification and validation performed on the models to improve the analysis and design stages of software development?

• Are the models used to improve communication between the engineering team and the stakeholders?

• Are models created using top-down or bottom-up approaches?

As we can see, the details here are as voluminous as those of the different ways of programming with 3GLs. Overall though, all of the different types of MDE are fundamentally different than current development via 3GLs which can be characterized by the common ‘Algorithms + Data Structures = Programs’ [198]. With MDE we have ‘Models + Transformations = Software’ [21], in which it is meant that we are working at a higher level of abstraction than 3GLs. It is hoped by the users that the higher abstractions will result in better designs and subsequently better software programs.

3.2 History

The uninterrupted appearance of more powerful and lower cost hardware has continually affected the way that software is created. From switchboards to punch cards to terminals to workstations, the need for software has grown. As this need grows, so do the size and complexity of our programs, which prompts new ideas regarding how to contain the problems associated with the situation. Assembly language raised us from coding in machine code, machine-independent programming languages and compilers hid some hardware complexities, and structured programming helped to enhance programmer capabilities. As the hardware kept advancing, programs continued
to get larger, eventually necessitating formally including a design stage in software
development to ensure that the architecture of a program was constructed as well
as the code. Bottom-up and top-down designs arose and structured design [201]
followed along with Parnas decomposition and Jackson System Development (JSD).
Eventually, structured designs began to have problems with larger business applica-
tions and the need for changes after initial construction. One source of the rigidity
of the software was the dependency tree created using functional decomposition [114]
or algorithmic decomposition [20] in structured design. The dependency tree arises
from a top-down perspective of the problem, thereby linking each dependency on the
top-level solution in which any single change can ripple throughout the tree and be
the source of multiple changes. This led to new methods, such as Object Oriented
Analysis (OOA) and Object Oriented Design (OOD). OOD arose from years of expe-
rience in the object-oriented approach, languages that better support Object Oriented
Programming (OOP), systems of increased complexity and size and volatility, and an
increasing domain-orientation of programs [31]. With object oriented methods, we
now focus on the nouns of the domain and how these nouns interact, which eliminates
the dependency hierarchy resulting from top-down functional decomposition.

The importance of moving from functional decomposition to a noun verb rep-
resentation of the problem becomes very obvious when we use real world analogies to
explain it. The focus of functional decomposition is on the steps to get from one state
to another and the sub steps of those steps. In my experience, this sometimes happens
when a novice computer user wants to know how I did something in my computer and
when I begin to explain this, the questioner begins writing down what I am saying
in a series of steps. That user wants to have a full listing of steps to accomplish the
task, but what happens if they have a different version of the operating system than
I do? Well, they need a new listing of steps to accomplish the task. What happens if they want to accomplish the task on their tablet computer as well as their PC? As you can see, this step by step instruction method is quite limited in flexibility as we have changes in our environment. In addition to needing multiple versions of steps for different environments, the user is always dependent upon their steps and cannot escape because they are unaware of more abstract conceptual ways to deal with how to accomplish their task. This is often evident when we begin to talk to said user and we become aware they are ignorant of the difference between memory and hard drive space, software and hardware, applications and data files, etc. While they have the intelligence needed to write down and follow steps, they lack the understanding behind those steps. In order for the user to become more accomplished they need more understanding of the computing domain, which will lead to an understanding of the steps. This understanding leads to flexibility because they will no longer need the specific steps which were provided before and can deal with their task at a higher level of abstraction. By the user’s new found understanding of the domain, they have raised their level of abstraction and greatly increased their flexibility in dealing within that domain. Small changes such as a new operating system will no longer pose the challenges they did previously.

3.3 Description

MDE, which has been used as far back as the 1950s [19], is proposed to be the next step in the evolving approaches to creating software. Though others may define it differently, I define MDE as a software process which focuses on a domain understanding which is captured in a model. The domain model may simply be used in the
analysis stage, but a more universal approach is to use domain models for analysis, design, and implementation and transforming the models amongst each other. There are multiple attempts to achieve MDE in efforts such as Executable UML (xtUML), MDD, MDA, Agile MDA, MDSD, and others. Many of the concepts of MDE are part of these methodologies, but the extent and focus of each methodology is different. The core concept of MDE is to create a domain model of the product and then to generate code automatically; the degree of generation to hand written code differs in methodologies and practice. The languages to express the product in the domain rely on a mathematical basis, which eventually enables the automatic generation of source code. Many of the concepts of MDE are from the OOA and OOD methodologies.

3.3.1 Graphical MDE

The Object Management Group (OMG) has created the MDA standard which provides a vendor-neutral way to define software models for forward engineering. The MDA specifies that a Platform Independent Model (PIM) is created using a DSL and after the PIM has been defined, it can be translated to a Platform Specific Model (PSM). In general, the PIM is created to express the problem in a domain specific way. A transformation is performed in order to get an implementation via the PSM, thereby making the PIM a representation of our design that is then implemented (Figure 3.1). Although the PIM to PSM transformation is generally represented as a single transformation, it can be multiple transformations depending on the goals of the project. Currently there is a lot of discussion regarding how to properly document a software program and the approach of MDA is to make the PIM to be the design

\[5\] http://en.wikipedia.org/wiki/Executable_UML
\[6\] Object Management Group http://www.omg.org
and the documentation of that design. The UML (Figure 3.2) diagram used to record the PIM can be translated into the XML Metadata Interchange (XMI) standard and thereby allowing the transfer of a PIM from one tool to another.

Though the ideal for MDA is to automatically generate all of the necessary source code from the model, there is often a need to hand write some of the code. The more one can elaborate one’s design in the PIM though, the more potential one has to reach complete code generation. The generation of the PSM is done by defining mapping functions from the PIM to the PSM by use of the Query/View/Transform-
Order o =
    new Order.Bulider()
    .buy(100, "IBM")
    .atLimitPrice(300)
    .allOrNone()
    .valueAs(new OrderValuerImpl())
    .build();

Figure 3.3: An internal DSL [64]

Textual MDE representations have existed as long as the graphical representations and today they are typically named DSLs. A routine DSL design begins with identifying the nouns of the specific domain and then identifying the actions that can act on those nouns. A DSL can be categorized into two types, called internal and external DSLs. An internal DSL is defined within some general purpose programming language and the goal is to make it simpler to programmers to understand and write the code for a specific area of a larger project. An internal DSL can be simple, such as the Java example shown in Figure 3.3. The syntax of an internal DSL is generally constrained to whichever general purpose programming language that is used by the developer.

External DSLs are freed from their general purpose languages syntax and have the ability to be more precisely suited to one’s needs. These are used every day by programmers everywhere as they are represented by examples such as SQL, Graphviz,
Cascading Style Sheets (CSS), XML User Interface Language (XUL), eXtensible Application Markup Language (XAML), and Make. The usefulness of a DSL is illustrated by the many real world instances they currently have. An example of the Graphviz DSL is shown in Figure 3.4. And, after passing this through the Graphviz 'dot' generator, we get Figure 3.6. Lest one think that the Graphviz language is only capable of simple diagrams, one can refer to Figure 3.5.

A conception of a DSL can be described with semantic models and projections. The semantic model is the actual artifact which is populated by a parser reading from a DSL. In our Graphviz example, the text listing is the DSL (Figure 3.4), 'dot' is the

---

A typical argument in favor of using a textual language is to enable collaboration with business users, give better expression to domain rules, and to achieve scalable development \[64\]. In addition to these, I believe that a DSL can be used to increase the productivity of programmers by containing some task in a convenient component and associated Application Programming Interface (API). Other advantages are to move the execution context of code from compile time to run-time \[53\]. Some of the disadvantages of DSLs are language cacophony, cost of building them, mimicking a general purpose language, and loss of flexibility \[53\]. Though these disadvantages do apply, some of them can mitigated.

### 3.3.3 Transformations

Obviously, the process of idealized MDE is heavily dependent on the ability to transform one model into another one. And, in contrast to MDA, one can have any number of models involved and there may not be clear cut ideal separations of platform independent and platform specific code. In fact, manual mapping may still be done even though we aspire to have mapping work automatically. Some mappings are designed to refine an abstract model into a more concrete one, and conversely other mappings raise the level of abstraction for a particular model; both of these mappings are named...
vertical mappings because they change the level of abstraction. If a mapping does not change the abstraction level, then it may be termed as a horizontal mapping, such as mapping a model from one type or version of UML to another. Another type of mapping that exists is for the purpose of merging two models together and these mappings allow us to separate aspects of our models and thereby achieve more clarity in our models. A more complex mapping scenario is represented in Figure 3.7 where we create client and server PSMs for the final application. Though a complete and comprehensive concept is presented with MDA, we find that the OMG has yet to properly define what a Platform actually is [147] and that it only presents one perspective of MDE.
3.4 Luminaries

As with any new movement, there are those few individuals who become widely known due to their early experience with the movement along with their eloquent and persuasive writings regarding the movement. Though every person has specific biases, a group of luminaries are likely to have their biases balanced out by others in that group. This gives us one way to probe into an unknown and fairly new development in the software engineering field such as MDE.

In order to get a viewpoint of what these luminaries support, we have compared their viewpoints of MDA. Scott Ambler [4] is well known for his exposition of the benefits of Agile Modeling [6]. His overall opinion of MDA indicates that it requires sophisticated skills, is similar to the previously failed attempt to promote Integrated-CASE (I-CASE) tools, is contrary to suspected developer’s textual preferences, and does not have good inter-operation between vendors of tools. He provides a document [3] with guidelines on the prerequisites for pursuing an MDA strategy. Another luminary, Steve Cook [33], believes that we need to make domain-specific models instead of focusing on one model to encompass everything, that while many models support development only some of those models should be used to generate code, and that the benefits of MDE are from using the technique for multiple projects—not just one. Martin Fowler [54] agrees that MDA may just be a revisiting of the Computer-Aided Software Engineering (CASE) tools from the 1980s, thinks that using graphical notations in only sometimes better than text, and does not think that UML will be accepted by the development community [52]. Stephen Mellor believes that we already perform MDA today [133], that the model does not need to be graphical, and that MDA has been overly hyped. H. S. Lahman states that in
redoing a 1 MLOC program in MDA resulted in a 10x decrease in maintenance and a 50% reduction in defects; in addition, he believes that MDA is bound to be the next major shift in how we create software [113]. Markus Voelter [192] suggests that the focus of the MDA on for interoperability and not on creating software applications thereby making some of those standards non-optimal for some purposes, UML is too general for many cases, it is simpler to create domain specific models, and these domain models can then be transformed into UML for MDA tools later if MDA is important for your project. From these luminaries, we can observe that the field of MDE is quite large, has varying perspectives, and may be used optimally by tailoring it to the particular area of application in which we are interested.

### 3.5 Successes

As the field of MDE is quite diverse, there is not a single MDE technique that works well in all areas–or at least none that we have yet found. Areas of successful application of MDE can point the direction for our current explorations [19]. Currently, UML has found a lot of success in the real-time embedded system world [158]. However, due to the abundance of techniques available to create software, people want more evidence of the benefits before switching from their current techniques to new ones [134]. MDE has been used for decreasing the cost of testing [156] and the development of automation systems has benefited [119]. MDA has apparently also been used for business software [78]. With the many successes recorded for MDE, we might see an increase in the use of this methodology in the future.
3.6 Challenges

Currently, MDE is not a mature or widely used methodology of software engineering. This leads to some challenges in using MDE for one’s project. Some of the challenges experienced by actual industry software architects are [155]:

1. A lack of understanding of MDE combined with a lack of materials in which to learn it.

2. A lack of tools and infrastructure. My team will need to understand and communicate in UML. There are not many comprehensive tools. All of our code is currently in text, how do we migrate it to MDE?

3. Platform independence is challenging.

4. Coders will resent being replaced.

5. A lack of sample models.

Another author proposes a different set of problems for MDE [79]:

1. Redundancy – multiple models are used for a software solution which creates a problem when a change is needed for a particular class method then human intervention is needed to assess the impact on the use cases and model;

2. Moving complexity rather than reducing it – Complexity is the result of the problem and the solution method. MDE may just move complexity elsewhere in the development process where in its new location it becomes more cumbersome to the project;
3. More expertise required? – Having a domain model can allow different experts to discuss the design, but with the transformations to get the final product any expert will have difficulty to identify all of the possible implications of a change they desire to make. If we consider the increasing trend of distributed teams, this problem may become more prominent;

4. MDE languages –

- UML has been criticized by some [13].
- A lack of UML semantics limits the ability of 3rd parties to create interoperable tools.

Though there are others who might list different challenges [41, 55] involved with MDE, we will stick with these for this discussion. One of the problems I’ve experienced directly is the lack of tools and infrastructure and this problem is related to the marketplace for MDE in industry [88]. One difficulty here is that most people seem to think of UML as the only way to write MDE code while UML Profiles or DSLs may be a much more productive approach. Common domain-specific abstractions can be contained in UML Profiles and thereby assist developers just as IDE’s contain certain base project settings for different domains. In researching the challenges to MDE, it was sometimes clear that the author had a perspective of MDE which might have been creating some of the challenges listed while others overly idealized or futuristic dogmatism created other challenges. It seems that many of the challenges that the many conflicting opinions regarding MDE implies that the term MDE is somewhat ambiguous and might be a real problem in trying to assess the challenges of MDE. Without a common vocabulary among researchers or practitioners, the communication of experiences is fraught with futility. This also brings up a
view from psychology and biology regarding the apparently different capabilities of individuals in society as a whole. Are some individuals better able to understand abstractions which subsequently empowers them to leverage MDE while others struggle in the area of abstractions? Do some individuals possess a better ability to understand information of a graphical nature and others of a textual nature? Some evidence indicates that the field of Engineering requires a higher ability to understand graphical notations than does the field of Computer Science [103]. If so, what could we expect regarding the adoption of a graphical technique among Computer Scientists at large? At this time, the real challenges of MDE may be hidden behind other problems such as vocabulary, mass opinion, subjective experiences, and intellectual differences between individuals.

Another challenge are the use of standards in the area of MDE. Though the OMG has proposed many standards, they are not followed and there may also be suboptimal standards in some cases. Fortunately, Software Engineering Radio had a round table discussion [185] regarding the state of MDE topics that gives us some insight into the standards regarding MDE. XMI was discussed in that it may have been created at a time before there was much knowledge of even how to do transformations and that it doesn’t really work between current systems. Also, the Meta Object Facility (MOF) may have suffered from special interests on the standards committee which resulted in the creation of Essential Meta Object Facility (EMOF) and Complete Meta Object Facility (CMOF), and UML may not be a great modeling language. One topic that is brought up is that the OMG does not offer validation tools in order to help people to follow their standards as the World Wide Web Consortium (W3C) actually does. It seems that the general opinion of this panel is that the OMG has done a good job regarding conceptualizing many standards to help the modeling
community, but that that effort has never reached the promise of those standards for multiple reasons including conflicts of interest among OMG members and a lack of follow through regarding their own standards. A real world example regarding the problem is that the Eclipse modeling tools are based on a meta-modeling architecture called Ecore and IBM does not follow EMOF or CMOF despite their involvement in the creation of these standards [185].

### 3.7 Research Today

Some of the areas of MDE that can be researched are the models, meta-model languages, transformations, and actual applications of MDE. In comparison to traditional source code development can we develop analogous tools available for MDE such as code comparison, source control, refactoring tools, text completion, search and replace tools, et cetera which can be applied to graphical models? How can a model best be validated? How may transformation languages be created in an effective manner? How may the methodology of MDA be taught to others [107, 194] and how could adoption of it be increased [171]? When is textual modeling better than graphical modeling and vice versa [137]?

Actual research is currently happening in the area of MDE today. Though some solutions may arise out of industry first, those solutions can be refined and codified by research. Currently there is research going on in the area of Aspect Oriented Programming (AOP) and MDE regarding horizontal mappings to weave aspects [128, 140, 166, 197]. Another area is on understanding how to create model transformations [118, 179] and how to categorize them [130]. And though Traceability of requirements was touted as being a given with MDE, is it really [167]? With the
interest in Empirical methods, some people are investigating MDE to evaluate the use of empirical methods [27], reflect on real world experiences [29], and learn more about industry’s use of MDE [88]. And, with the emerging importance of parallel computing, there is more research to be done [97]. User Interfaces have gotten a fair amount of attention as well [28, 56, 136, 184]. Still others are exploring the application of MDE in different areas [46, 48, 56]. Others are trying to assess the current state of MDE [79].

In addition to individual research papers, there are some labs dedicated to the exploration of MDE. Formal system analysis for modeling the domain and building applications more efficiently are aspects of MDE that are topics at Technische Universiteit Eindhoven [180]. The University of Alabama also has a very active research lab [75]. At Vrije Universiteit Brussel, the Software Languages Lab [193] performs research regarding models, transformations, and verification.

3.8 Resources

There are many resources for the topic of MDE and the concept has been around for a number of years. Some of the standards offered by the OMG [148] are XMI, MOF, Enterprise Distributed Object Computing (EDOC), UML, Business Process Model and Notation (BPMN), Workflow Management Facility (WfMF), Object Constraint Language (OCL), UML, Semantics of a Foundational Subset for Executable UML Models (FUML), UML Profile for Enterprise Application Integration (EAI), Abstract Syntax Tree Metamodel (ASTM), Knowledge Discovery Metamodel (KDM), and Common Warehouse Metamodel (CWM).

There are a number of groups for MDE [6, 37] and multiple books exist as
well [5, 21, 21, 58, 114, 117, 129, 151, 177]. The Model Driven Software Networking [37] website helps those interested in MDE to network amongst each other. The Agile Modeling Website [6] website offers a lot of information regarding Agile MDA by an author of a leading Agile MDA book [5]. The Software Engineering Radio contains a few podcasts relating to the thoughts of pioneers in the MDE field [55]. Markus Voelter offers a tutorial to MDE [190] and there are also educational programs offered for learning MDE [24].

Many tools currently exist for the area of MDE. The Eclipse Platform has a Graphical Modeling Framework [182] which enables developers to develop graphical editors based on Eclipse Modeling Framework (EMF) and Graphical Editing Framework (GEF). The MetaCase organization provides a MetaEdit and Modeler tool [131] to define models and generate appropriate source code (see Figure 3.8). Enterprise Architect [176] from Sparx Systems enables you to define your UML models and generate code. IBM provides a Rational Software Tool [89] and a video series [90–95] gives a demonstration for the Rational System Architect. ArgoUML\footnote{Tigris.org. ArgoUML \url{http://argouml.tigris.org}, accessed on March 2013} provides a way to design in UML and generate a variety of code outputs. JetBrains Meta Programming System (MPS) [98] gives you an editor creation along with your DSL creation to add to your productivity. There are some comprehensive modeling tools and others just support part of the modeling process.

### 3.9 Conclusions

MDE is illustrated here as a diverse and complicated field. The origins of MDE have great influence in its practice today, so much so that some people suggest that reading
Figure 3.8: A diagram in MetaCase [156]
books in OOA and OOD will provide a firm basis for understanding the MDE concepts more completely. The MDA provides a comprehensive overview of the concepts. Though the MDA is conceptually cogent, luminaries greatly disagree regarding the importance and usefulness of this topic, which really solidifies our viewpoint of this field as being in a developmental stage and diverse. The challenges for MDE are unclear due to the different perspectives involved in people who report the challenges and the level of expertise needed to use MDE tools appropriately. With the subject MDE, much research is currently going on which will help to clarify and refine MDE. So though the concepts of MDE hold much promise for the future, they need some real research and development in order to succeed for a larger audience.
Chapter 4

User Experience and User Interfaces

4.1 Introduction

Humans have used machines for a very long time and since these machines are created for human beings, they need to conform to the needs of those users. Though this concept is basically simple (see Figure 4.1), by exploring the details the model can become quite complex. Each machine has a particular environment in which it needs to operate and this compounds the problems in designing user interfaces. In generally, a UI applies to any kind of interface, but here we are only concerned with software interfaces of which most are currently GUIs. However, the future UIs for software are likely to be much more diverse than the typical Graphical User Interfaces (GUIs) today.
4.2 History

Originally, computers were quite limited in what they could accomplish due to speed and memory limitations. User interfaces for older computers were switches and lights—the user programmed the computer with switches and the lights provided the output. As this method of I/O was quite taxing for the user, as computers gained speed and memory, the user interface was modified to do more for the user. At one time, punch cards were used to store information thereby eliminating the tedious process of programming by switches. A program could be stored on cards and then fed to the computer in order to program the computer. Later, computer keyboards and terminals allowed users to program without cards. At first, the terminals were quite limited, but they eventually improved. A big step forward came with the GUI, which allowed the user to be free of having to know a command language to communicate with the computer.

4.3 Description

A user interface defines the way that a user may communicate with a complex machine. Though a machine may be usable, we can discern that there are different levels of usability in products today. Much of the field of User Interface Design is concerned
with understanding the different levels of usability and the ways in which to achieve those levels. And, the intricacies of the field are such that the process of UI design is rarely automated, because all UIs still need to be tested and validated by humans in order to be finished.

One typical usability problem today is that of remote controls for different entertainment devices such as the DVD, VCR, Roku, TV, stereo, etc. (Figure 4.2). Universal remotes have been created to deal with this abundance of different types of remotes, but they have their own usability problems such as modes and complex buttonery themselves. While it may not be difficult to understand usability problems from a subjective viewpoint, the complexities of designing a UI from an objective viewpoint is difficult and time consuming. To deal with objective design and evaluation of a UI, people have proposed guidelines, principles, and theories to use in the creation of UIs. In fact, the theories regarding UIs are so numerous that they can be classified as descriptive, explanatory, prescriptive, and predictive [173]. One potential answer to the sprawl of guidelines is to address the source of the guidelines through items such as the psychological rules for them [102].

The field of UI design has a number of different approaches such as those shown in the list below [74]. Each method is unique and has different strengths and weaknesses and some people try to provide an overall view of user experience methods [169, 188]. The Usability Professionals’ Association (UPA) has published a poster (see Figure 4.3) illustrating a methodology of user experience design. Of all the methods of user experience, we have chosen a small subset of these for our dissertation:

- research planning—the why, what, when and how
• competitive research—determine competitor products and analyze them with respect to how a user uses that product and how they feel about it

• interviews—recruit users from one’s target audience and interview them

• focus groups—select a group of target users, gather them together, and have a discussion about topics related to the product

• object-based techniques—ways to extend a verbal interview by using photos, collages, and other objects

• field visits—learn by observing how actual users use a product in situ

• diary studies—users record their activities in a diary

• usability tests—have users complete a task with one’s completed product

• surveys—to determine who one’s actual users are and what they might want in the product
• global and cross-cultural research—determine what different cultures think of one’s product

• published information and consultants—read studies, articles, and reports; one can also hire consultants for expertise one doesn’t have

• analyze qualitative data—analyze the data and generate ideas

• usage data—collect data automatically from the product while it is being used and reflect on how to improve the product accordingly

• customer feedback—use customer complaints, problems, and comments to generate ideas for improving the application

The field of designing user interfaces is large and this has resulted in some divisions of roles that are commonly seen in the field. Some of these roles include [160]: usability engineer, web designer, UI designer, information architect, and user experience (UX) designer. Our inference from the number of roles and techniques that are available is that user interface design is a complex field which has resulted in specialist roles.

Another area of UI design concerns the understanding of users and their limitations. Many models have been created to describe the ways the humans can interact with something, such as a Human-Computer Interaction (HCI) model of the mind as an information processor from cognitive psychology (Figure 4.4).

For our needs, we have focused on using Interaction Design (IxD) methods to create the UI. Though IxD has many techniques, we chose a small subset that satisfied our requirements for this project. Those methods were design ethnography, personas, UI sketching, and usability testing. More details about these methods are provided
Figure 4.3: Usability professionals design poster
Figure 4.4: Human processor model [160]
in the following sections. A general design process that was generally followed is Goal Directed Design (Figure 4.5).

### 4.3.1 Design Ethnography

Design ethnography is concerned with studying how real users use software tools. The results of this study are used to inform the entire design process and to guide that design thereby. The methods here range from surveys, observing users in a usability lab, observing users in their place of work, and more. The importance of this task is emphasized by many user experience professionals, but it is accomplished in different ways. One way of performing this task is through studying the users through participant observation, interviews, and surveys; all of which may be done in a laboratory setting or in situ. In addition to performing a small ethnography study, they can be done on a larger scale to attempt to get a picture of a large user base.

The key here is to learn what the users’ goals are when they are working. This involves getting to understand the context and environment experienced by users and how this shapes their work. In addition to this, we (as designers) want to keep the designing to ourselves and refrain from allowing the user to dictate a design to us. It is not uncommon for some fields with intelligent and subjective users to begin to believe that they understand what they need, but this is antithetical to a design ethnography. One problem here is that the user generally lacks the design knowledge to come up with...
with good solutions to their problems and discussing their proposed design clouds the view of what the user’s goals actually are. We are only attempting to identify user goals and further understanding of the users, the design ideas will be generated from this data and not before. Therefore any designs proposed by users are ignored at this time. The interviews progress by asking open ended non-leading questions to learn about the user’s goals and environment. The first interviewees provide a base of understanding and later interviewees are used to refine that understanding by tailoring questions appropriately.

4.4 Successes

Success in this area can be seen today. The GUI is a result of research work at Xerox PARC, and was then incorporated into the Apple Macintosh and Microsoft Windows. The UI for listening to music has advanced from CDs to electronic storage of music in MP3 players, iTunes, and Windows Media Player. The typical radio has advanced to streaming music services on the internet which provide suggestions of songs one might like based on one’s listening history. Even the television has progressed from users consulting a TV station schedule to learn when their desired shows would be playing to an on demand TV show and movie service such as Amazon Instant Video\(^1\), NetFlix\(^2\), YouTube\(^3\), and Hulu\(^4\).

One of the areas of UX that has gotten a fair bit of attention are search based UIs [196]. These have been in development for many years, and their importance grows as the amount of accessible information grows. While Google does provide an

\(^1\)http://www.amazon.com/Instant-Video/b?ie=UTF8&node=2858778011
\(^2\)https://signup.netflix.com/
\(^3\)http://www.youtube.com/
\(^4\)http://www.hulu.com/
example of success in the realm of search, other search engines exist for other purposes such as scientific data searches.

4.5 Challenges

Some challenges for the field of UX are currently being talked about. One of these challenges concerns spreading the understanding in industry of the importance of UX and hiring trained practitioners. Another challenge is the constantly changing technical landscape and the new interfaces and methodologies [43,172] that are needed. As the technical landscape changes, the field also needs to address human concerns such as privacy and safety. Though a UX professional may know what needs to be created for a UI, those UI needs may conflict with the underlying software architecture for an application and this is a persistent problem. For the immediate future, it appears there will be more than enough work for UX professionals to do.

4.6 Research Today

Theories are being created in the HCI field today. The Technology Acceptance Model (TAM) describes how users develop acceptance of a particular technology by using a model employing information systems theory (Figure 4.6). The TAM is being applied to the medical industry [84].

Researchers are also attempting to judge the usefulness of current techniques of UX [149]. Some are investigating the reproducibility of a usability study [135]. A long running topic is to measure the ROI of usability [124,144,161]. Another persistent item of research are conflicts between desired usability features for a UI versus the
features that a software's architecture will easily support [10, 50, 51, 122, 157, 170]. The topic of generating personalized user interfaces using Artificial Intelligence (AI) is also being investigated [59, 60]. An interest of how human psychological needs are related to usability is being looked at [81], and in a similar way age-related issues are being studied [125]. There are many methods, and this has spurned some to collect and discuss these methods [106]. Newer multi-person displays are being created and studied [181], and the specialized needs of the web are being elucidated [49]. With server based applications, studying real user activity is becoming more interesting [42].

4.7 Resources

Schools offer education in Human Factors, and HCI at numerous universities–often from the Psychology department [86]. There are also a number of groups supporting HCI and UX practitioners such as Human Factors International [87], the Interaction Design Association [96], the User Experience Professionals Association [187], and the Nielsen Norman Group [77]. Numerous books exist describing IxD [35], general overviews [80, 173], cursory introductions to the field [111], up and coming online studies [2], and even specialty books for focused areas such as search user interfaces [82, 165, 196]. Outsourcing of usability testing also exists\(^5\).

\(^5\)UserZoom www.userzoom.com
4.8 Conclusions

This chapter has covered a wide range of topics in UX and HCI. By presenting this range, we desired that the reader is provided with a high level view of the field. We have attempted to show the impressive breadth and depth of this field in order to emphasize that it is a discipline of its own. Even practitioners in the field specialize and focus on a small area. The reasons for presenting the material in this way will become more apparent as the reader goes next over our description of the communication gap between the different people involved in the production of software.
Chapter 5

Approach: User Experience Meets Model Driven Engineering

5.1 Introduction

This chapter describes the essential aspects of our proposed approach. The approach is a combination of a separation-based UI architecture, UX considerations, and MDE. We identify the premises we used to base our architectural decisions on and illustrate our deductive process to create the architecture. The approach is then described in detail with regards to how it should be implemented. The role of the developers is explained and the tools and techniques we used are described.

5.2 Derivation

Now we will delve into observable and measurable differences between people and then draw conclusions about the effects of these differences as they relate to the develop-
ment of software products. Here, we are at odds with the common American saying "You can be whatever you want to be." Instead, we propose that each individual is quite unique and each has intrinsic adaptations to physical and mental environments. While an individual can learn, adapt, and change, these are limited ways to adjust to a specific environment and are still constrained by inherent qualities of the individual. These background ideas present a perspective that we have taken which essentially gave form to our design approach.

5.2.1 Role Specialization

What is wrong with the picture shown in Figure 5.1? Though Michael Jordan was an outstanding athlete, he was a top athlete in professional basketball. However, after years as a top athlete in basketball, he attempted to play baseball, and he only participated in the minor leagues. Why would Michael Jordan meet with less success in baseball than basketball? The problem here is specialization—people adapt to their environment and the more adapted they become to one environment the less adapted they will be to another environment. In the physical world of sports, performance measurement is very obvious which makes it easy to discern Jordan’s suitability to these two sports. Although some athletes do successfully play two sports at a professional level, they comprise a short list of people1. Another example of this principle would be the short person who desires to play professional basketball—it is obvious that this person will have a much more difficult time to achieve this dream than another with similar talent and drive who is taller.

The physical sports analogy is presented here to frame our perspective on the mixing of UI design and Software Engineering (SE). Though mental abilities are

Figure 5.1: Michael Jordan and role specialization
not visually observable they are still there and without getting into a Darwinian discussion of heredity versus environment, we propose that mental abilities can limit the effectiveness of a person to a particular environment. In support of this viewpoint, we have found some evidence which indicates that the abilities required for Software Engineering are different than those for UI design. The Johnson O’Connor Research Center measures aptitudes of individuals and correlates them to specific jobs, and they have published a report [23] showing the general aptitude patterns of software engineers working in industry. To assess the aptitudes of UX professionals, we refer to the aptitude pattern of psychologists in lieu of a better comparison [32]. The comparison with regards to that paper is illustrated in Table 5.1 and Figure 5.2. For a detailed discussion of one of these aptitudes listed here and how it relates to software engineering, see [103]. Here, we have evidence that the skills needed by different professions are different. In a loose generalization, our guess is that the UX professional skill set would be close to the Psychology skill set as they both need to understand people. We also assume that the Engineering and Systems Programming skill sets are very close to those needed for Software Engineering. The conclusion, therefore, is that the skill set of UX professionals and software engineers is quite different and this will result in poor performance of those individuals working in the wrong area. In addition to skill set specialization, we would like to assert our belief in a statement by Santiago Ramón y Cajal [200] “Although popular opinion may not agree, ‘Knowledge occupies space.’ ” To sum up our perspective, we believe that the limitations of humans confine them to either being good in one area or less proficient in a multiple of areas and that areas of ability seem to require specific sets of skills which are difficult and take time to hone.

**Premise 1: Role Specialization increases productivity and success**
Table 5.1: Scores of different professions [32]

<table>
<thead>
<tr>
<th>Test</th>
<th>Engineering</th>
<th>Systems analysis and programming</th>
<th>Psychology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphoria</td>
<td>-0.01</td>
<td>0.34</td>
<td>-0.26</td>
</tr>
<tr>
<td>Analytical Reasoning</td>
<td>0.34</td>
<td>0.39</td>
<td>-0.01</td>
</tr>
<tr>
<td>Number Series</td>
<td>0.83</td>
<td>0.56</td>
<td>-0.03</td>
</tr>
<tr>
<td>Number Facility</td>
<td>0.28</td>
<td>0.34</td>
<td>-0.14</td>
</tr>
<tr>
<td>Wiggly Block</td>
<td>0.69</td>
<td>0.55</td>
<td>-0.13</td>
</tr>
<tr>
<td>Paper Folding</td>
<td>0.97</td>
<td>0.73</td>
<td>-0.13</td>
</tr>
<tr>
<td>Structural Visualization</td>
<td>0.88</td>
<td>0.70</td>
<td>-0.19</td>
</tr>
<tr>
<td>Memory for Design</td>
<td>0.66</td>
<td>0.65</td>
<td>-0.22</td>
</tr>
<tr>
<td>Number Memory</td>
<td>0.17</td>
<td>0.53</td>
<td>-0.07</td>
</tr>
</tbody>
</table>
5.2.2 Communication Gap

A communication gap exists when the sender and receiver have different conceptual meanings for words. A common phrase in advertising is “Nothing is too good for our customers.” Upon a close inspection, we can derive two different meanings for this statement: (1) “Our customers deserve so much that nothing in the world can actually meet this requirement”, or (2) “Our customers are so undeserving that giving them nothing would be giving them too much.” Another expression “Heather had a little lamb” can be interpreted as (1) Heather owned a lamb of small size, or (2) Heather (a female lamb) had a baby, or (3) While eating, Heather ate a small amount of lamb. An example related to a communication gap between users and programmers has to do with Boolean logic. If a user asks the computer to “Find restaurants in Virginia and Georgia,” the user wants restaurants in Virginia and
restaurants in Georgia. However, a computer programmer setting up a search UI may have the software interpret the phrase so as to finding restaurants in both Virginia and Georgia. For programmers, the user would need to specify "Find restaurants in Virginia OR Georgia." [35] So, the looseness of language contributes to possible confusions during communication.

The complexity of the world is incompatible with the abilities of the human brain, which leads to simplifications and generalizations when we communicate. These simplifications become problematic for communication when two communicators have different simplifications and that difference is not detectable with the words used to communicate. In addition to mismatched simplifications, professions often create very definite meanings for some words which, at a vulgar level, are more generalized. When two professionals of the same group communicate, that communication is made efficient through these definite meanings (often called jargon). However, if professionals from two different professions communicate, they will have a communication gap caused by their different jargons. An obvious difficulty here is regarding the common selling point of computers being the frequency of the CPU to describe the speed of the computer. The laymen think the CPU speed determines the speed of the computer, while hardware engineers understand that CPU speed, number of instructions executed per cycle, cache, bus size, bus speed, memory size, software, et cetera all contribute to the speed of the computer.

These problems with communication gaps occur all of the time due to the looseness of language and the different conceptual understanding of the communicators. The problem is complex and multiple models of communication have been proposed, such as the one shown in Figure 5.3. A prime candidate for the illustration of communication gaps is provided by the creation of user interfaces by programmers.
A popular UX professional writes

“Our first four textual bloopers are about poor writing in the text displayed by software. They are the result of giving the job of writing text to the wrong people: Programmers.”

Numerous examples of programmers failing to write software to communicate well with customers illustrates the fact that programmer skill sets are unique and do not generally enable programmers to communicate clearly with the general population [34]. However, many programmers are not aware of the apparent fact that sender and receiver of a message could have entirely different encoding / decoding mechanisms (Figure 5.3). It is here that we arrive at our second premise:

Premise 2: Communication gaps cause confusion and inefficiency

5.2.3 Multiple Communication Gaps

We found that in any piece of software there are many communication gaps that can be identified. In this paper, we are going to attempt to classify the different communication gaps that we are aware of and to address only the largest of those gaps
in order to reduce complexity in that way. In our project, the user-machine interface can be viewed as a diagram (Figure 5.4). The software constitutes whatever we insert in between the user and the machine. Therefore, we can consider this communication gap to be represented by CG. If we illustrate the software using a Separation Based UI architecture with another diagram (Figure 5.5), then we notice that we actually have three separate communication gaps; $cg_1$, $cg_2$, and $cg_3$ (Figure 5.6). Here, we can find the largest communication gap by specifying the actual knowledge domains that each component belongs to (see Figure 5.7). It is this communication gap, $cg_2$, that we will address in order to alleviate some of the problems posed by communication gaps.
Premise 3: The largest communication gap in a Separation Based UI architecture is the UI-Code gap, between the user and machine knowledge domains

5.2.4 How to Bridge the Gap?

There are ways to eliminate a communication gap, such as having one person learn the jargon of the other, having both people learn the others jargon, or providing an interpreter. Any one of these solutions requires work and energy to accomplish. From premise 1, we believe that attempting to educate the programmer or the user is the wrong direction. Forcing the programmer to understand the user will take time away from their core expertise and thereby cause that expertise to decline. Or, if you take our perspective of innate abilities of different people, programmers will necessarily never be very good at communicating to the actual users. Suggesting that the programmer learn about the users and thereby eliminate the communication gap
is analogous to asking all of the users to learn programming jargon in order to use software more easily. Numerous articles have been written calling for educational institutions to teach programming to all students, others call for individuals to learn programming on their own\textsuperscript{23}, while still others suggest that learning programming is a bad idea\textsuperscript{4}. In reality, the subject of end-user programming is very much alive and well [109], but the problems encountered in this area are similar to the problems in SE in addition to the lack of experience and understanding of the end-user programmers. In this controversy, we subscribe to an interpreter option, and that interpreter is a UX professional who understands the realm of HCI theory [159]. Here, our UX professional serves as the interpreter between the user and the Code.

Another gap that we must deal with is the Code-Machine gap. We will not delve into the detailed opinions of the field about who is most appropriate to address this gap. Here, we will just state our opinion that a programmer who understands the software code and the limitations of the machine on which it must run is the most appropriate choice for this gap. And now we can create the entire depiction of the layout of responsibilities as shown in Figure 5.8.

To summarize our progress thus far, we have used knowledge domains to identify regions amenable to Premise 1. The isolation of knowledge domains allows us to identify where the most confusion and inefficiency will occur (see Premise 2). By isolating our professionals in the respective knowledge domains, we also isolate their


communication to that of a discussion of one communication gap between their knowledge domains. This isolates much of the confusion and allows for a focused effort to be put on bridging the largest communication gap (see Premise 3). Additionally, this method allows our professionals to be largely ignorant of each others work and thereby allows them to focus on their specialty rather than be dragged into learning about another area of knowledge. To conclude, we have Figure 5.8 to represent our distribution of responsibilities in order to achieve development efficiency.

We now propose that formalization can help development professionals to bridge their communication gap. There are many ways to formalize such a gap, but we chose to constrain the gap to a language–more specifically, a Domain Specific Language (DSL). We chose a DSL in order to use a technology that already exists and has available literature describing it. This is the final step in our architectural design (see Figure 5.9).
5.3 Description of the Proposed Approach

Our approach brings together three different ideas, that of UX, DSLs, and code generation. We believe that much of current industrial practice for software creation follows Figure 5.10. The designer is responsible for defining the User’s needs and specifying what needs to be done with the UI. The programmer then translates this specification into a program consisting of two parts–the UI and Code. Some are moving to a more advanced version of this method by using a UX designer to replace a general designer. At the same time, there are some who propose that we write software at a higher level of abstraction than Third-Generation Languages (3GLs), such as using UML. In previous chapters, we discussed communication gaps between the users, UX professionals, and programmers. Here, these communication gaps are used to set the boundaries for our proposed method of creating software (Figure 5.11). The DSL shown here is meant to be understandable to a UX designer and to thereby release the UX designer from the necessity of learning to program; which we hope will result in better role specialization and productivity. This DSL will provide the necessary interface between the UI and the code. In order to further increase productivity,
we believe that using MDE code generation techniques from the DSL will quicken software development, reduce errors, and increase flexibility (Figure 5.12). Though we have shown the PIM code and PSM code in typical MDA fashion, we make no assertion as to the number of abstraction levels to provide here—that is left for the individual’s preference.

First, we will begin discussing the UX designer role for this approach. The UX designer will be an expert in user interfaces and is not required to understand how to program. The UI will be created by the designer through the use of a simple GUI builder or by directing a programmer. Though many GUI builders exist, we have not found many that do not require some understanding of programming languages and these also have limitations in expressivity which might detract from the creation
of some User Interfaces (UIs). Therefore we leave it open as to how the UI creation actually gets done, whether from a simple UI builder or by delegating tasks to a programmer. The UX designer will interact with the DSL in order to communicate with the application.

The programmer will need to understand programming and DSL creation. The DSL will be defined by the stakeholders during the requirements and design stages of development. The programmer will create the DSL, or an improved option would be to have this created by a language designer, but many development teams may not have access to a language designer. All technical issues arising during the UX design can also be solved by the programmer. The ultimate responsibility of the programmer is to create the software application code through code generation and manual edits.

The overall process of a project is shown in Figure 5.13. We begin with gathering requirements for the application and then create the Software Requirements Specification (SRS) \[175\] with a specific vocabulary of terms, precisely specified, in order to prepare for our eventual DSL creation. From there, the design will be created to describe how a technical solution will be reached to meet the SRS and here we also focus on specifying terms exactly for the next stage. The critical stage of creating the DSL then gives the design a formal description from the perspective of a UI interacting with an application, because the intention of creating the DSL is to make a well defined interface between the UI and the application. For example, in an ATM application, requirements such as ”USER deposits MONEY,” ”USER withdraws MONEY,” and ”USER closes ACCOUNT” clearly indicate an interface between the USER and the ATM machine, which can be expressed in the language of a DSL. Now the benefits begin to show up because the code creation and UI creation may now progress independently. The UI may be prototyped, tested with users,
finalized. Consequently, the DSL can be used to generate partial code in lieu of full generation, such that code can be added by the programmer after the generation phase. After the UI and code have been created, they will be integrated to create the final product. At this point, we would like to note that the integration phase consists of merging a UI which interfaces with a human and produces a DSL script to communicate to the application, while the application is controlled via the DSL commands. The flexibility here is that two or more separate UIs can be integrated with the same application in order to accommodate different environments such as an instance running on an individual workstation or one running in a web browser. The connection between the UI and application can be managed via a connector component which handles the details of routing messages to and fro, thereby allowing both UI and the application to be ignorant of their distance from each other. The deployment stage will be similar for conventional applications as our process is not unique at this stage.

5.3.1 Code Creation

The code creation process shown in Figure 5.13 is generalized to show a typical DSL code generation process using several development tools that we have selected—see more details in Section 6.12. Figure 5.14 shows a much more detailed code generation diagram for the particular way that we implemented this process. The first step is to take the DSL from the DSL specification and distill it into a grammar for Xtext. Once we have an understanding of the form of the grammar, we start a new Eclipse Xtext project (Figure 5.15) and then enter the grammar (Figure 5.16). The creation of this grammar enables us to generate the parser, the metamodel, and an Eclipse editor tool via the 'Generate Xtext Artifacts' command (Figure 5.17).
In the next stage, we create templates for the code generation; these templates are coded in Xtend (Figure 5.18). The Modeling Workflow Engine 2 (MWE2) tool then allows us to generate either a generator or an interpreter for our DSL (Figure 5.19). When we take the interpreter path, we start a new Eclipse instance that incorporates the editor tool created earlier and that tool provides syntax corrections while writing our DSL script (Figure 5.20). After the DSL script has been created, we can then build that script which results in the creation of the generated code (Figure 5.21). Subsequently, the generated code is built to give us the final application. Here, we have only explained the Eclipse plugin branch of the code generation process, but if the reader desires to learn the details of the Stand Alone branch, please refer to [17].

5.3.2 UX Creation

The creation of the UI will be largely performed by experienced UX designers. In our project, we closely followed the methods contained in a popular book on IxD [35]. Design Ethnography is used to understand the user environment and goals, which provides fuel for determining the real requirements and needs of users. Other methods such as sketching the UI and getting feedback from users and stakeholders are also used. When the work has progressed to a stage where the UI can be created, the UX professionals will avail themselves of tools such as GUI builders that are simple so that they won’t need to understand programming; one such tool being MetaCase’s MetaEdit tool [131] or Meta-Gui-Builders [123]. The end result of this work will be that the UI provided will be capable of generating scripts in the defined DSL and thereby communicate with the application code.
Figure 5.13: Process diagram
Figure 5.14: Code creation details
Figure 5.15: Creating an Xtext project in Eclipse

Figure 5.16: A written grammar
Figure 5.17: Creating the artifacts from a grammar
Figure 5.18: A written template
Figure 5.19: The command for generating the parser with MWE2
Figure 5.20: A DSL script for our DSL

Figure 5.21: Code generated from our DSL
5.3.3 Steps of Implementation

There are a number of well-defined steps which, if followed correctly will allow the implementer to design software that conforms to our approach. The first step is to analyze the requirements and to determine what the actual problems are that the software is being asked to address. This can then be compared with the tasks that the users actually perform. Then the implementer separates the steps of user’s tasks into a two columned table identifying if the particular part of a task is in the User domain or the Machine domain. With this table, the UX professional and the programmers can work out what information needs to be communicated between the machine and the user with the DSL. Once the DSL has been specified, the UI and Code development can carry on independently until they must at last be integrated.

5.3.4 Conclusions

In this chapter, we have introduced our approach to creating software using a DSL as a bridge between UX professionals and programmers. Concepts were introduced sequentially and combined in order to arrive at our final conception of how software could be developed more efficiently. First, we introduced our premise of role specialization using analogy and actual research results. Next, we explained the communication gap issue and illustrated it with several examples of communication gaps and the resulting confusion and inefficiencies introduced by them. In the analysis of software development we identified many communication gaps and pointed out the largest of those gaps in a typical Separation-Based UI architecture. To bridge the major gap in a software project, we identified the knowledge domains related to software development and discussed who would be responsible for each knowledge
domain. For bridging the gap between the knowledge domains, we proposed the use of a DSL. Specific tools and techniques were then identified and used to detail how our view of software development would proceed using our suggested methodology.
Chapter 6

Application Specification and Design

6.1 Concept

In the world of scientific computing, the availability of information is growing due to an increased ability to store data and offer it through the internet. Researchers need to be able to deal with increased amounts of data which necessitates new tools for handling this data [76]. The increased amounts of data along with the sharing of data among researchers results in a need for a common vocabulary, taxonomy or ontology for referring to specific data to enable queries and interoperability [15, 120]. One part of this effort is that the National Science Foundation (NSF) has decided to support the interoperability of software among earth-system researchers by offering data services and tools through the Unidata program. Some of the services they provide for data is a central location to find data sources and tools to access data archives. They also offer display software, analysis software, and data access and
management software. The success of this program has lead to some of their formats for data storage becoming very prevalent, such as the NETwork Common Data Form (NetCDF) format [141].

The concept of this project is to create, using our proposed approach, software that is able to subset, filter, and combine different NetCDF data files. NetCDF is a file format defined by Unidata which is in its fourth version. This file format specification is provided by Unidata and they also provide tools for working with the files. This project’s outcome differs from current NetCDF software by using MDE in order to perform operations on NetCDF files. It is expected that MDE will bring productive advantages in dealing with the format.

An innovative aspect of this project lies in the application of MDE to the creating of NetCDF tools. We hope to considerably speed up the development process with MDE thereby providing more time for development of new features or in ensuring that the tools are robust enough for use by researchers. Another innovative part of the project is to focus on usability early in the design in order to improve resulting usability in the final product—we hope that this will increase the appeal of our tool to users.

6.2 NetCDF

The NetCDF binary file format was originally created to allow for interoperability between the many Unidata software programs. The data in the file is also described inside of the file and the format is compatible across different computing platforms. Interfaces in Fortran, C, C++, and Java exist for accessing data in NetCDF files. There are currently three types of files; Network Common Data Form Classic (NetCDF Clas-
NetCDF Classic applies to all file versions before 4.0 and NetCDF applies to versions 4.0 and later. All subsequent uses of NetCDF imply NetCDF-4 or above.

The NetCDF formats have various limitations: NetCDF Classic is generally limited to file sizes of 2 GB, NetCDF 64 is limited to 4 GB, and NetCDF is limited only by the support of the file system on the computer. The result of these limitations is that researchers must perform data searches over multiple files and they must produce data in multiple files. Other limitations exist, but they are not as important to this project.

There currently exist tools to subset, combine, and create the NetCDF format such as [186]:

- File Array Notation (FAN)
- File Interpolation, Manipulation, and EXtraction (Fimex)
- IRI/LDEO Climate Data Library
- Ivan Shmakov’s NetCDF Tools
- NetCDF add-in for Microsoft Excel
- WebWinds

Though we are proposing to create another tool to work with this format, this tool has been created using our proposed approach, centered on the techniques of MDE. We will attempt to extract subsets of data from one or multiple files and to also convert from the NetCDF format to American Standard Code for Information Interchange (ASCII) format (Figure 6.1).
6.3 Persona

In the process of IxD, user research is conducted in order to create a model of certain target segments of one’s user population. A persona is a structured way to record these models. For our project, we only have one target audience of climate scientists and after ethnographic interviews, we derived a persona (Table 6.1). Through the interviews there was a difference between beginner and expert climate scientists use of a NetCDF file, but those differences were not significant enough to create a second persona.

6.4 Requirements

In coming up with requirements, we focused heavily on usability of the tool in order to increase the appeal to researchers who lack the technical background of software specialists. We consider such a non-functional requirement at this stage relates to its importance in this project and the need to account for it in these early stages. The functional requirements were determined by researchers in climatology.

The functional requirements for this project are shown in the Table 6.2. The naming convention is F#.# where the first number indicates priority levels of 1, 2, or

Table 6.1: Climate scientist persona

- Name: Bob
- Job: Climate Scientist
- IQ: High
- Time Spent: Gather data, analyze data, write reports
- NetCDF Data Needs:
  - explore data sets
  - grab data sets, transform data to Matlab format, and analyze data
Table 6.2: Functional requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The system will convert a NetCDF file to an ASCII file</td>
</tr>
<tr>
<td>1.2</td>
<td>The system will extract a subset of data from a NetCDF file to a new NetCDF file</td>
</tr>
<tr>
<td>1.3</td>
<td>The system will extract a subset of data from multiple NetCDF files and export to a new NetCDF file</td>
</tr>
<tr>
<td>1.4</td>
<td>The system will combine the data from multiple NetCDF files and export into a new NetCDF file</td>
</tr>
<tr>
<td>1.5</td>
<td>The system will not corrupt the source files</td>
</tr>
<tr>
<td>2.1</td>
<td>The system will provide a graphical interface for defining the workflow of NetCDF operations</td>
</tr>
<tr>
<td>2.2</td>
<td>The system will provide a textual interface for defining a workflow of NetCDF operations</td>
</tr>
<tr>
<td>2.3</td>
<td>The system will save a workflow</td>
</tr>
<tr>
<td>2.4</td>
<td>The system will open a saved workflow</td>
</tr>
<tr>
<td>3.1</td>
<td>The system will validate a textual workflow description</td>
</tr>
<tr>
<td>3.2</td>
<td>The system will report errors in an invalid textual workflow description</td>
</tr>
<tr>
<td>3.3</td>
<td>The system will report errors in the execution of a workflow</td>
</tr>
</tbody>
</table>

3 and the second # is a sequential tracking number. Providing both a graphical and textual interface at the outset will allow us to appeal to a wide audience where we suspect that many new or infrequent users will prefer a graphical interface while more experienced users will prefer a textual interface along with the scripting possibilities that exist in that interface. The saving of workflows allow users to save time with regards to a complex conversion / extraction process.

A similar numbering convention is used for the nonfunctional requirements (Table 6.3). Many of these involve the usability concerns of the project. Other nonfunctional requirements such as speed were not considered to be of central importance to our goals and are therefore not provided here.
Table 6.3: Non-Functional requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Error messages will be understandable to user</td>
</tr>
<tr>
<td>1.2</td>
<td>No technical terms will be used</td>
</tr>
<tr>
<td>1.3</td>
<td>Validation of the workflow will be done automatically</td>
</tr>
<tr>
<td>2.1</td>
<td>Possible mis-spellings will be identified and reported</td>
</tr>
<tr>
<td>2.2</td>
<td>Documentation will be provided to describe how to use the system</td>
</tr>
<tr>
<td>2.3</td>
<td>Users will be informed of the progress of long-running conversions / extractions</td>
</tr>
<tr>
<td>3.1</td>
<td>Only work with NetCDF 4.0 and later versions</td>
</tr>
<tr>
<td>3.2</td>
<td>The system will work on Mac, Windows, and Linux computers</td>
</tr>
<tr>
<td>3.3</td>
<td>The user will not need to go through extensive installation procedures</td>
</tr>
</tbody>
</table>

### 6.5 Use Cases

These use cases (Figure 6.2 and Table 6.4) represent how the system may be used by users [18, 175]. Two types of actors have been identified: researchers and developers. The researchers will primarily be interested in creating some subset of data from an existing set of NetCDF files. Developer’s interest is in using the system to provide further capabilities such as visualization, search, or processing of a data set.

While the use cases presented here present a subset of the capabilities of existing tools, the main goal has been to illustrate our approach and we hope that the MDE process will be fruitful enough to expand this tool in the future. Our focus on usability is reflected in the detailed use cases (Tables 6.5 and 6.6).

In order to trace our requirements, we create a traceability matrix (Table 6.7).
Figure 6.2: Use case diagram
Table 6.4: Use case descriptions

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get data subset</td>
<td>Get a particular subset of data from a collection of NetCDF files</td>
</tr>
<tr>
<td>2</td>
<td>Email data</td>
<td>Email some of the values from a NetCDF file to someone</td>
</tr>
<tr>
<td>3</td>
<td>Discover data</td>
<td>Discover what variables are contained in a NetCDF file</td>
</tr>
<tr>
<td>4</td>
<td>Reorganize data</td>
<td>Use a textual interface to get a subset of data from a NetCDF file, reorganize that data, and create a new NetCDF file with the organized data</td>
</tr>
<tr>
<td>5</td>
<td>Eliminate erroneous data</td>
<td>Eliminate erroneous data from a NetCDF file by filtering that file with a mathematical constraint, such as (-20 C &lt; temp &lt; 100 C)</td>
</tr>
<tr>
<td>6</td>
<td>Find variables</td>
<td>Identify all of the NetCDF files in a collection that contain a specific set of variables</td>
</tr>
<tr>
<td>7</td>
<td>Split NetCDF file</td>
<td>Break a NetCDF file into a number of smaller NetCDF files</td>
</tr>
<tr>
<td>8</td>
<td>Combine NetCDF files</td>
<td>Combine the data from multiple NetCDF files into one NetCDF file</td>
</tr>
</tbody>
</table>
### Table 6.5: Use case 1–detailed

<table>
<thead>
<tr>
<th>Use Case 1</th>
<th>Get a data subset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors</strong></td>
<td>Researcher, Developer</td>
</tr>
<tr>
<td><strong>Preconditions:</strong></td>
<td>NetCDF source files are available.</td>
</tr>
<tr>
<td><strong>Postconditions:</strong></td>
<td>NetCDF file is created with desired set.</td>
</tr>
</tbody>
</table>
| **Main Success Scenario:** | 1. User opens NetCDF tool’s textual editor  
2. User indicates NetCDF input files to use  
3. User picks a particular subset of the data contained in chosen files  
4. User presses the execute workflow button  
5. User accesses the desired data set in a NetCDF file |
| **Extensions:** | 2.a NetCDF input files are an earlier version than 4.0:  
1. System shows 'Invalid version for <filename>, all input files must be of NetCDF version 4.0 or higher. <filename> is of version <number>).'  
2. User returns to step 2  
3.a data set specification is invalid:  
1. System shows failure message  
2. User returns to step 3 and corrects errors |
| | 4.a No data available:  
1. System shows 'No Data Available'  
2. User returns to step 2 or 3 |
Table 6.6: Use case 2–detailed

<table>
<thead>
<tr>
<th>Use Case 2</th>
<th>Email data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors</strong></td>
<td>Researcher, Developer</td>
</tr>
<tr>
<td><strong>Preconditions:</strong></td>
<td>NetCDF source files are available.</td>
</tr>
<tr>
<td><strong>Postconditions:</strong></td>
<td>NetCDF file is created with desired set.</td>
</tr>
<tr>
<td><strong>Main Success Scenario:</strong></td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>User opens NetCDF tool’s textual editor</td>
</tr>
<tr>
<td>2.</td>
<td>User indicates NetCDF input files</td>
</tr>
<tr>
<td>3.</td>
<td>User picks a particular subset of the data contained in the chosen files</td>
</tr>
<tr>
<td>4.</td>
<td>User specifies that they want an ASCII file with that chosen data</td>
</tr>
<tr>
<td>5.</td>
<td>User presses the execute workflow button</td>
</tr>
<tr>
<td>6.</td>
<td>User gets the ASCII file</td>
</tr>
<tr>
<td>7.</td>
<td>User emails ASCII file to someone</td>
</tr>
<tr>
<td><strong>Extensions:</strong></td>
<td></td>
</tr>
<tr>
<td>2.a</td>
<td>NetCDF input files are an earlier version than 4.0:</td>
</tr>
<tr>
<td>1.</td>
<td>System shows 'Invalid version for &lt;filename&gt;, all input files must be of NetCDF version 4.0 or higher. &lt;filename&gt; is of version &lt;number&gt;’</td>
</tr>
<tr>
<td>2.</td>
<td>User returns to step 1</td>
</tr>
<tr>
<td>3.a</td>
<td>Data set specification is invalid:</td>
</tr>
<tr>
<td>1.</td>
<td>System shows failure message</td>
</tr>
<tr>
<td>2.</td>
<td>User returns to step 2 and corrects errors</td>
</tr>
<tr>
<td>5.a</td>
<td>No data available:</td>
</tr>
<tr>
<td>1.</td>
<td>System shows 'No Data Available'</td>
</tr>
<tr>
<td>2.</td>
<td>User returns to step 1 or 2</td>
</tr>
</tbody>
</table>
Table 6.7: Functional requirements–use case traceability matrix

<table>
<thead>
<tr>
<th>ID</th>
<th>Use Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>F1.1</td>
<td>x</td>
</tr>
<tr>
<td>F1.2</td>
<td>x</td>
</tr>
<tr>
<td>F1.3</td>
<td>x</td>
</tr>
<tr>
<td>F1.4</td>
<td></td>
</tr>
<tr>
<td>F2.1</td>
<td>x</td>
</tr>
<tr>
<td>F2.2</td>
<td>x</td>
</tr>
<tr>
<td>F2.3</td>
<td>x</td>
</tr>
<tr>
<td>F2.4</td>
<td></td>
</tr>
<tr>
<td>F3.1</td>
<td>x</td>
</tr>
<tr>
<td>F3.2</td>
<td>x</td>
</tr>
<tr>
<td>F3.3</td>
<td>x</td>
</tr>
</tbody>
</table>

6.6 Technological Choices

We intended to run this application from a Web based platform in order to achieve portability and ease of access goals. In order to be able to work with current existing tools with the Nevada Climate Change Portal (NCCP) [126, 127, 139] we planned to offer the services of this tool through a web service. MDE has been at the core of our methodology and the Eclipse tool enabled us with EMF to assist with the development of a textual interface for the DSL. Furthermore, we used GEF to create the graphical interface. Other resources were Xtext, the Ecor e model (Figure 6.3), OCL, Graphical Modeling Project (GMP), and others.

6.7 High Level Design

The high level design uses UML packages (Figure 6.4) to describe the elements of the design. For the UI we have a textual and graphical representation to accommodate different types of users. It has been suggested that while novice users prefer a
Figure 6.3: Ecore model [44]
graphical representation, more experienced users prefer a textual interface [191]. The textual representation, a DSL, may also allow more flexibility in porting the software to a web service interface as well.

6.8 Medium-Level Design

In order to provide more details regarding these package descriptions, the validation package is broken out to display the class model (Figure 6.5). The Validator class is the main public interface to the module and allows users to check if their elements meet the constraints set for the validator. If something does not meet the constraints, then it does not pass that filter. A ValidationRule interface provides a way for the software to iterate through all of the known constraints and check if they pass or fail accordingly. For the operation of extracting a subset of values from many NetCDF
files and putting that subset into one or more NetCDF files, we need this Validator package. The user will be allowed to specify the subset via a list of constraints; i.e. “soil temperature depth < 3 cm”. The list of constraints must be specified correctly and exactly.

### 6.9 Data Design

The NetCDF files have been the source of data for this project (Figure 6.6) and various constructs support user actions on those NetCDF files. We have used the Common Data Model (CDM) format from Unidata (Figure 6.7) to define the NetCDF files, but due to the limited scope of this project the CDM may not be implemented in full.
Figure 6.6: Data class diagram
Figure 6.7: Unidata CDM Data Access Layer object model
In addition to the data models for the files, we have defined constraints (Figure 6.8) for the project. These constraints provide a way to filter the data from NetCDF files and enable us to run validation checks on the actual DSL script set up by the researcher.

### 6.10 Detailed Design

The details of our design are presented here in order to fill in some of the dynamic UML perspectives and the DSL specifics. The typical workflow for a user is to filter the data of some input files and write the output files (Figure 6.9). The activity diagram for the isValid() method of the Validator is shown in Figure 6.10.

### 6.11 User Interface

The graphical interface is similar to creating a workflow diagram (Figure 6.11). The input for the files allows us to get the data from already existing NetCDF files; though the combinator node is shown in Figure 6.1, it is not shown in our final application.
Figure 6.9: Typical user workflow activity diagram

Figure 6.10: isValid() activity diagram
for the sake of brevity. For the extraction functionality, we have a result of \( \{ x \mid\ constraint(x) \} \) where the \( constraint() \) function is defined by the user (Figure 6.12). Constraints may be defined using boolean algebra and operator precedence is (, \( \neg \), \( \land \), and \( \lor \). An example of a constraint would be \((precipitation \geq 20\%) \land (temperature > 34C)\). The conversion functionality to convert an ASCII file to a NetCDF file is quite straightforward as it changes the input set of binary bits into an output in ASCII bytes.

Upon using the application, the user is presented with a window with a blank canvas and a side list box containing the various elements that can be used to create a workflow on the canvas (Figure 6.13). The user can then create their specific workflow by dragging the source files and actions on the source files and creating a workflow with them (Figure 6.14). In regards to the details of creating a workflow, the user has to choose source NetCDF files from their local directories with a dialog...
Figure 6.13: Initial application state

box (Figure 6.15). In addition to this, the user has to select constraints for any Extractor functionality, as we see in Figure 6.16 where the user has scrolled down the toolbar to reveal constraint controls. When the user drags a constraint control onto the canvas, he is asked to define the constraint formally and the resulting definition is show in the icon instead of the word ‘constraint.’ Constraints can be combined with boolean operations and then connected to the appropriate Extractor icon (Figure 6.17). In order to run the current workflow, the user presses the ‘Execute’ button and the workflow execution is illustrated by highlighting appropriate icons in red as the workflow progresses (Figure 6.18).

Though we assumed that the graphical interface was easy for the novice researcher to use in order to create workflows of NetCDF file manipulations, this graphical interface also suffers from some drawbacks. First, the use of a computer mouse is high for this interface, the diagram takes up a lot of space, and our graphical in-
Figure 6.14: Workflow with files specified

Figure 6.15: Selecting a source NetCDF file
Figure 6.16: Created constraint $\text{precipitation} < 20\%$

Figure 6.17: Finished constraint construction
Figure 6.18: A running workflow

terface requires the user to be manually present to run the simulation. As a response to the stated drawbacks of the graphical interface, we are creating a textual interface which resembles writing a script to create a workflow. Users can specify which files to input, how to filter them, and where to write the output. An example of a script that might be created is shown in Figure 6.19. This frees the researcher from needing to know any of the details of a particular programming language and only requires basic knowledge of file system paths and boolean algebra expressions. In addition to that, we hope that in the future the textual interface can be expanded to deal with remote files and ontologies [120].

Two different user interfaces have been described here and though they appear separate, we have attempted to connect them together. The first UI to be developed was the textual interface and the graphical interface was developed second. As both UIs employ the same functionality, we have the graphical UI export its representation
Figure 6.19: Preliminary script

to a script which can then be processed by the program. It is in this way that we provide consistency for the UI throughout the lifetime of this program.

6.12 Tools

The approach for this project is to use a DSL to get our required functionality. To satisfy our development needs, we have used a textual DSL, and to satisfy the UI needs, we planned on using a graphical DSL. Reasons for using a textual DSL as the main part of the project include the many tools and a long history of textual methods in developing software. The graphical DSL was intended to satisfy our needs for a UI with good usability for the novice or infrequent user of the software. The textual DSL was created with Xtext on the Eclipse platform with the Ecore architecture. Xtext has been in development for a few years and has enough maturity for our purposes. The process of creating the textual DSL is shown in Figure 5.14, which consists of a number of file creations followed by relevant building of new artifacts. The first step consists of starting Eclipse and creating an Xtext project, which is followed by the writing of the grammar. After building the artifacts for the grammar and writing an Xpand template, one can build the MWE2 code generator engine. Now that the
relevant Eclipse plugins have been generated, one can start a new instance of Eclipse with which to create a new Java project and then write the Xtext script which receives text highlighting and autocompletion via the plugins. Once the script is created, one can create a src-gen file, followed by a building of the Java project. The files that have been generated can now be incorporated into a new Java project, built, and executed.

6.13 Conclusions

We have explained the application of our methodology to the creation of a program to subset NetCDF files for climate scientists. The current environment of NetCDF manipulation programs has been explained and we described our conceptual viewpoint of our program with a simple block diagram (Figure 6.1). We performed ethnographic interviews and used the resulting learnings to derive a persona for the climate scientist. For specification of the program, we delineated functional and non-functional requirements and illustrated these with a use case diagram (Figure 6.2). UML was used to describe our high-level, medium-level, and data design. The detailed design was represented with block diagrams, UML, and GUI wireframes. The tools that we used were detailed and we showed how they were used in our implementation of the design. As with any design documentation our presentation here is incomplete, but a comprehensive description has been provided regarding our design of the NetCDF tool using our prescribed methodology.
Chapter 7

Applying the Methodology

7.1 Introduction

We chose to work on a problem which affected climate scientists—that of the subsetting of NetCDF files. Here, we describe the application of our methodology to the real world. This is intended to give the reader a comprehensive view of how the methodology works in practice. The chapter is structured according to the flow of the methodology shown in Figure 7.1. Note that given my expertise in both UX [65, 67] and SE [9, 57, 66, 68, 69] in the area of climate science, I have played both the roles of UX professional and programmer.

7.2 Step 1: Identify the gaps

Here, we knew we were going to attempt to create software to ease the climate scientists job in using NetCDF files. In order to learn how the user interacted with NetCDF files, the UX professional performed a number of ethnographic interviews.
Step 1
Identify the Gaps

Step 2
Identify the Knowledge Domains

Step 3
Create a DSL

Step 4a
Create the UI

Step 4b
Create the Code

Step 5
Integrate

Figure 7.1: A simplified workflow representation of our methodology
Design ethnography is a way of studying real users to determine how they use tools and many methods to do this are provided [160], one of which are ethnographic interviews. These interviews consisted of asking the users what they currently use NetCDF files for, where they get the files, and how they process those files. With the information our UX professional collected, he created a Persona (Table 6.1 in the previous chapter) for reference and then wrote a list of steps that users performed to subset data from a NetCDF file or files. In performing these steps, they were accomplishing their goal of analyzing climate data for their research needs.

1. Search for data
2. Find data in NetCDF files
3. Download NetCDF files
4. Extract data from NetCDF
5. Analyze data in Matlab

7.3 Step 2: Identify the knowledge domains

Our UX professional and programmer got together and discussed the steps of the user’s task in order to categorize those steps as shown in Table 7.1. While we could have put the 2nd and 3rd steps into the machine domain as well, we did not in order to limit the scope of this dissertation to a manageable amount of time. With this identification of domains, we can now separate the work of the team. By clearly separating the steps into the relevant domains, we limit what will be done by those working on the domains. This really simplifies the specification of the software design and the UI design.
Table 7.1: Classifying steps into domains

<table>
<thead>
<tr>
<th>User Knowledge</th>
<th>Machine Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search for data</td>
<td></td>
</tr>
<tr>
<td>Find data in NetCDF files</td>
<td></td>
</tr>
<tr>
<td>Download NetCDF files</td>
<td>Extract data from NetCDF</td>
</tr>
<tr>
<td>Analyze data in Matlab</td>
<td></td>
</tr>
</tbody>
</table>

7.4 Step 3: Create a DSL for the domains to talk to one another

Our UX professional and programmer worked together to enhance our previous table to that of Table 7.2 to show a DSL column and the corresponding messages that will go between the domains. This DSL will later be formalized, but for now it is only important to hash out the details of exactly what needs to be communicated between the two domains.

Table 7.2: Designing a DSL

<table>
<thead>
<tr>
<th>User Knowledge</th>
<th>DSL</th>
<th>Machine Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search for data</td>
<td>Filter ‘precip’ From xyz.nc</td>
<td></td>
</tr>
<tr>
<td>Find data in NetCDF files</td>
<td>xyz.csv</td>
<td>Extract data from NetCDF</td>
</tr>
<tr>
<td>Download NetCDF files</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyze data in Matlab</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7.5 Step 4a: Creating the UI

The process of creating the UI comprised a number of steps. The first step consisted in creating a prototype UI and attempting to get feedback from climate scientists regarding the prototype. The first prototype was a drag and drop GUI that allows a climate scientist to create a workflow out of components that are familiar to them (Figure 7.2). After the creation of the prototype, we proceeded to attempt to implement this idea in a real world artifact. In order to be faithful to the methodology, we identified the MetaEdit+ application which could be used as a GUI builder without requiring the user to understand programming. The resulting MetaEdit+ prototype is shown in Figure 7.3. Unfortunately, after the UI was created, it had a number of problems: (1) required user to have MetaEdit+ installed, (2) did not have a clear area for the tool icons for NetCDF files and the filters, and (3) contained all MetaEdit+ controls rather than only the controls needed to create a NetCDF subsetting model. Due to these problems, we moved on to use other methods for creating the UI.

At the time that the MetaEdit+ prototype was completed, we became informed by a usability study conducted on a separate project (Appendix A). That project was created to allow climate scientists to search for data, but it had nothing to do with NetCDF files. The result of that usability study was that a non-graphical approach is more effective than a graphical one. This inspired us to begin a search for a less graphical solution to our problem. The candidate solution that looked the most promising was a natural language output interface [35]. This led to the creation of some UI sketches to determine the usefulness of such an interface. It was determined to test three separate interfaces, the first a graphical solution much like what we created with the MetaEdit+ tool (Figure 7.4), the second a mix of graphical and
Figure 7.2: A running workflow

Figure 7.3: A prototype in MetaEdit+
text-oriented (Figure 7.5), and the third was a text-oriented interface resembling a natural language output (Figure 7.6). When tested with users, it seemed that their opinion was that the text-oriented interface would be the most optimum interface (more details are available in Appendix A).

It is at this point in the development that we decided to analyze the prototype UI sketches with GOMS. The analysis is shown in Table 7.3. Some assumptions have been made in order to make the comparison fair, such as restricting the number of constraints to one—this does not follow the sketches of the User Interfaces (UIs) presented earlier. We also assumed that the number of parameters needed to specify a constraint was $P$ for all interfaces—only the sketch for the natural language interface shows multiple parameters for a constraint. Through the total number of steps required for each UI, we were able to understand how poor our first option would have been. The consequence of this table was to drop our Workflow UI in MetaEdit+ and
Figure 7.5: A sketch of the mixed approach

Figure 7.6: A sketch of the text-oriented interface
pursue the remaining prototypes.

Table 7.3: GOMS Analysis (N = number of source files, P = number of constraint parameters, C = number of constraints)

<table>
<thead>
<tr>
<th>Steps</th>
<th>Workflow Operator Count</th>
<th>Hybrid Operator Count</th>
<th>Natural Language Operator Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Drag and drop source NetCDF files</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>2) Drag and drop constraint box</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3) Select constraint</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>4) Connect each input file to constraint</td>
<td>NC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5) Create output file</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6) Connect each constraint to output</td>
<td>C</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7) Press execute button</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total Operator Count</td>
<td>NC + C + P + N + 3</td>
<td>N + P + 2</td>
<td>N + P + 2</td>
</tr>
</tbody>
</table>

7.5.1 User Testing

At this point the UX professionals can begin to test the UI with users to begin the iterative process of evolving the UI. There are currently no techniques that substitute for testing a UI with target users. Our testing for the application is covered in Chapter 9.

7.6 Step 4b: Creating the code

We began by converting the current form of the DSL into an official syntax. The DSL which was first loosely specified in Table 7.2 was formalized. Table 7.4 shows all
messages that must be passed between the UI and Code components.

Table 7.4: The formalized DSL

<table>
<thead>
<tr>
<th>User Knowledge</th>
<th>DSL</th>
<th>Machine Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search for data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find data in NetCDF files</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Download NetCDF files</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add file</td>
<td>→<code>getInfo: x3.nc</code></td>
<td>Get list of variables</td>
</tr>
<tr>
<td></td>
<td>←<code>variable list</code></td>
<td></td>
</tr>
<tr>
<td>subset files</td>
<td>→<code>input: x3.nc</code></td>
<td>Extract data from NetCDF</td>
</tr>
<tr>
<td></td>
<td>→<code>output: out.nc</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>→<code>transform: precip &lt; 20</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>←<code>out.nc</code></td>
<td></td>
</tr>
<tr>
<td>time remaining?</td>
<td>→<code>time remaining</code></td>
<td>Estimate time remaining</td>
</tr>
<tr>
<td>notify user of time remaining</td>
<td>←<code>mm:ss</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>←<code>error: error msg</code></td>
<td>Error occurs</td>
</tr>
<tr>
<td>Analyze data in Matlab</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>←<code>exception: message</code></td>
<td>Exception occurs</td>
</tr>
</tbody>
</table>
with the Ecore architecture. Xtext has been in development for a few years and has enough maturity for our purposes. The process of creating the textual DSL is shown in Figure 5.14, which consists of a number of file creations followed by the relevant building of new artifacts. The first step consists of starting Eclipse and creating an Xtext project, which is followed by the writing of the grammar. After building the artifacts for the grammar and writing the Xpand templates, one can build the MWE2 code generator engine. Now that the relevant Eclipse plugins have been generated, one can start a new instance of Eclipse with which to create a new Java project and then write the Xtext script which receives text highlighting and autocompletion via the plugins. Once the script is created, one will create a src-gen file, followed by a building of the Java project. The files that have been generated can now be incorporated into a new Java project, built, and executed.

Using Xtext, we began by defining a grammar for our textual scripts (see Figure 7.7) which took 34 lines of written code. The entire program was then written in the Xpand language and subsequently imported into our Xtext project as output targets for our textual scripts. These templates (Figure 7.8) resulted in 476 lines of text. At this point we had defined all of the necessary information for Xtext to generate a parser, so we generated the parser code (see Table 9.1). With a completed parser, we began to write and test scripts (Figure 7.9). The DSL script was then fed into our parser and Xtext generated the resultant Java code (Figure 7.10).

7.7 Step 5: Integration

After creating the UI and Code, we worked on integrating them. This went very smoothly as our DSL served as the glue to connect these two parts and it worked
Figure 7.7: Xtext grammar

```
NcDsI.text  NcDsIGenerator.xtext

grammar org.xtext.demo.mydsl.NcDsI with org.eclipse.xtext.common.

generate nCDSI "http://www.xtext.org/demo/mydsl/NcDsI"

NcDsI:
    workflow+=Workflow*;

Workflow: 'Workflow' name=ID
    'input': '('source+=Source';')
    'output': '('sink+=Sink';')
    'transform': '('transition+=Transition';')

Transition returns Transition:
    (Extraction) // | Transformation
```

Figure 7.8: Xtext template

```
def compile(workflow e) ...```

```java
if (e.container != null)
    package [e.fullyQualifiedName]
</EMLIF>

import java.io.*;
/*
 * A file to hold the main procedure
 */
public class workflow
{
    public static void main(String[] args) {
        //open input file
        NetCdf_Source ncc = new NetCdf_Source();
        String binaryfilename = "xyz.nc";
        byte[] readBuf = ncc.readBinaryFile(binaryfilename);
        String tuple[] = null;
        try {
            BufferedReader in = new BufferedReader( new FileReader(binaryfilename));
            BufferedReader outw = new BufferedWriter( new FileWriter("out" + binaryfilename));
```
as expected. In order to describe how the integration worked, we will describe the steps that would happen as the program is run by climate scientists. First, the climate scientist finds some data sources and downloads the necessary NetCDF files to their computer. They then open our MetaEdit+ application and create a workflow representing their desired subsetting. When they press the "Execute" button, the UI creates a script (Listing 7.11). This script is then fed into our parser (Figure 7.12), which puts the parsed information into our code templates and then generates the Java source code for the specified workflow (Figure 7.13). These files are then compiled into an executable (Figure 7.14) and that executable is run to execute the workflow (Figure 7.15).

7.8 Conclusion

We have illustrated how our methodology is applied to a real world application. Each step of the process has been described in detail and a diagram of the flow of the
Figure 7.10: Generated Code

```plaintext
Workflow w01
  input: /cygdrive/c/final-draft/x3.nc;
  output: /cygdrive/c/final-draft/out.nc;
  transform: precip > 20;
```

Figure 7.11: A script, original.ncdsl, for our DSL

```plaintext
$ ls
original.ncdsl
$ java -jar ncdsl-compiler.jar original.ncdsl
Code generation finished.
```

Figure 7.12: Command to create generated java files from original.ncdsl
steps has been provided (Figure 7.1). The steps described here represent where the boundaries are between different work elements and provide a guideline for using our methodology on other projects.
Chapter 8

Evaluation of the Methodology and Related Work

8.1 Introduction

In this chapter, we focus on evaluating our methodological approach. We first introduce other existing work in this area of UI separation based architectures and UI creation techniques. Then we compare our approach to that prior work in regards to user interaction and UI creation. Finally we discuss the benefits, drawbacks, other opportunities, and impacts of our work.

8.2 Other Comparable Approaches

Several approaches that have certain similarities with our proposed approach are overviewed next.
8.2.1 Plastic User Interfaces

This method attempts to address the lack of collaboration between the human-computer interface and SE by using MDE to attain an ability to easily modify the UI. UI plasticity is defined as ‘the capacity of user interfaces to adapt to the context of use while preserving usability.’ [183] Three context models are specified as the user model, the environmental model, and the platform model. The usability of the UI is defined differently for different contexts. If a UI is intended to run in a set of different contexts (C) then we can define a set of values (V) to represent our usability. The set V can then be mapped to the set C of contexts in unique ways to preserve usability across the contexts. With this formal model of the UI with usability values V and contexts C, further conclusions may be reached such as [169]

An interactive system $S$ is plastic from a source context of use $C_i$ to a target of use $C_j$ if the following two conditions are satisfied:

1) Adaption, if needed, is supported when switching from $C_i$ to $C_j$

2) Usability (value) is preserved in $C_j$ by the adaption process

Now, with a solid base, MDE is brought in and a meta-modeling language is created [25, 26]. This method of combining MDE to generate UIs while preserving usability defines plastic user interfaces and work is still being done [36].

Current efforts in this area are being pursued; such as runtime manipulation of plastic interfaces [36]. The Context Aware Modeling for Enabling and Leveraging Effective interactiON (CAMELEON) project, which is being supported by the Eu-
European Commission, has created tools such as TERESA\textsuperscript{1} and ReversiXml\textsuperscript{2}. A framework (Figure 8.1) provides a common vocabulary to aid in discussing plastic user interfaces and many other UI generation strategies. The USer Interface eXtended Markup Language (UsiXML) consortium is working on consolidating other user interface markup languages, and on providing tools for creating UIs with UsiXML. A couple of suggestions for paying more attention to transformations and Concrete User Interface (CUI)s in plastic user interfaces are: [36]

\begin{quote}
At the CUI level, meta-modeling not only lags behind innovation, but briddles creativity.
\end{quote}

\begin{quote}
MDE as a software development methodology, has favored the dichotomy between design stages and the run-time phase which has resulted in
\end{quote}

\begin{itemize}
\item models and code can get out of sync
\item design tools are intended for software professionals
\item run-time adaption is limited to the changes of context identified as key by developers
\end{itemize}

\subsection*{8.2.2 Usability-Supporting Architectural Patterns}

Software Usability has traditionally been treated in an isolated manner by focusing on the UI as the embodiment of usability. Relatively recently, some researchers have taken an interest in the limitations imposed on the UI from the supporting architecture. Some usability concerns such as a Cancel feature generally need a considerable

\textsuperscript{1}A tool for design and development of multi-platform applications \url{http://giove.isti.cnr.it/tools/TERESA} accessed 2013-06
\textsuperscript{2}A tool that can reverse engineer a HTML file into UsiXML
Figure 8.1: CAMELEON reference framework for the development of Plastic User Interfaces [36]
amount of support external to the UI module and therefore are difficult to add later in a project lifecycle. Ways to identify these architecturally sensitive UI features of a software product during the requirements phase have been developed [105, 157]. Deriving architecture patterns from the requirements has also been described in the literature [11, 99]. Usability-Supporting Architecture Pattern (USAP)s have been created to provide insight for designers in order to deal with incorporating usability into the software architecture [12]; a pattern language has also been created with these USAPs [99]. A USAP isolates usability concerns of the UI which need a significant amount of support outside of the UI and thereby result in difficulties when attempting to add such a feature late in the development process. Research regarding the effectiveness of USAPs has found them to be effective [73]. And, finally, the concept of addressing usability concerns during the architectural and design stages of product lines has also supported the practice [178].

This method allows designers to recognize the dependencies between the UI and the code. It does not refer to a methodology as our approach does. However, our approach is subject to these USAPs as well and the USAPs may be used in conjunction with our methodology.

8.2.3 Game Maker Language

The Game Maker Language (GML) has been created to enable people to make computer games more easily. The GameMaker:Studio™ environment integrates with the language to make the creation of the scripts even easier. Users are allowed to use drag and drop to set up a large portion of their games and this is enhanced with textual

---

scripts written by the user (Figure 8.2). The purveyor of GML claims that one can create games 80% faster with their tool and scripting language. Once the game is created, it can be deployed on iOS, Android, OS X app, Windows executable, or HyperText Markup Language (version 5) (HTML5) application. YoYoGames offers tutorials and other materials to enable users to create their own games.

While the GML supports creating games, it appears that the user still needs to know quite a lot about programming as evidenced by the example script shown in Figure 8.2. Our approach instead attempts to remove the need to know programming from the UX professional—instead of just attempting to increase the productivity of programmers by offering them a higher level of abstraction to program in.

8.2.4 Linden Scripting Language

The Linden Scripting Language (LSL) allows users of the virtual 3-D world game Second Life® to write scripts to control their games. This scripting language (Figure 8.3) resembles the C programming language syntax and uses event-based programming. Objects in the virtual world can be created and imbued with behavior with this

```gml
var xNew, yNew, nNew;
if can_move = true then begin
    can_move := false
    alarm[0] := 5
    xNew := x + lengthdir_x(14, direction)
    yNew := y + lengthdir_y(14, direction)
    nNew := instance_create(xNew, yNew, obj_car)
    with nNew begin
        speed := obj_car.speed + 3
        direction := obj_car.direction
    end
end
```

Figure 8.2: GML example
language. The game interface is used to program and therefore offers some similar capabilities as an IDE in working with LSL. The Second Life® game was built by the users through the use of LSL. It has been reported that there were an average of 54,000 concurrent users of the game in May 2010⁴, so the game has become quite large.

Although the LSL does offer a programming interface to users, that interface still requires knowledge of programming as evidenced by the example script shown in Figure 8.3. It seems to us that the market for LSL could be greatly expanded by simplifying this interface. In contrast, we are not offering users a way to program, but are seeking instead to enable a UX professional to work independently of code creation in order to improve the productivity of the software development process. Although UX professionals may want to create a UX for the user to be able to program, they generally don’t and therefore our methodology does not concern itself with end-user programming.

8.2.5 User Interface Markup Languages

The declarative creation of a GUI has gotten a fair amount of attention of researchers, but the focus seems to be entirely on the declarative creation of the GUI, focusing on separation of concerns rather than usability [71, 72]. Others focus on creating a UI for multiple platforms in a flexible way [14, 47, 48, 142]. Some focus on using a EXtensible Markup Language (XML) and EXtensible Stylesheet Language (XSL) to enable the user to customize the UI for their own purposes [104].

According to Wikipedia\(^5\), there are 42 platform specific markup languages. The UsiXML is an effort to generalize the UI markup language jungle. These languages appear to offer simpler ways to address multiple platforms and thereby save effort in reduplicating code for multiple interfaces for multiple platforms. While research has been done in this area [83, 138, 143], much of it focuses on removing the need to program multiple interfaces by using modeling and code generation techniques instead.

A story here is provided to illustrate the current difficulty that UI markup languages face despite their theoretical advantages. The work 'native' describes an application that runs on a native platform, such as Windows 7, iOS, and others. A mobile web app works through the web and can be viewed through a browser. Due to the time required to create and maintain multiple native applications, many software producers may choose to go with a mobile web app in order to only have to code one interface for multiple platforms. However, choosing producer time efficiency over an application that may work better is not always the best choice. For example, LinkedIn first designed their mobile site via mobile web app technology, but eventually turned

to native apps in order to improve performance\textsuperscript{6}. They explained that extending the memory life of mobile devices was the main reason, but they also benefited from improved smoothness of their animations. And LinkedIn is not the only company to favor native applications, as in the case of Xero\textsuperscript{7} who state that they found developing a fast HTML5 app was hard, HTML5 apps take more time to develop than native apps, and native apps can benefit from specific capabilities offered by a native platform. Even Facebook has moved from HTML5 to native apps\textsuperscript{8} and they cite user responsiveness as being a primary reason. From this data, we infer that native apps can provide some advantages to mobile web apps for some companies. What the stories here indicate is that there are multiple ways to make a UI technologically and that these different ways are significant. An abstracted mobile web app approach fails for some users due to limitations in the abstraction to deal with specific differences, a lack of tools for creating fast mobile web apps, and the extra time needed for communication over the web. Therefore a real world solution must not only be strong in theory, but it must be able to compete with other methodologies in reality as well.

In contrast to a UI markup language, we are focusing on a DSL to create an interface based on role specialization. Our method can support the use of a UI markup language, but is by no means required to use them. In fact, we do not believe


Figure 8.4: Snow example

that creating multiple UIs for different platforms is necessarily a waste of effort if it provides a better experience for the users in doing so. Although UI markup languages promise to generate multiple interfaces, we believe that they still apparently fail to do so adequately in many cases, as shown by the recent experience of some companies making mobile web applications.

As a practical matter, we will now discuss one UI markup language, Snow\(^9\), to give the reader a feel for what these languages look like. Snow uses Lisp syntax (Figure 8.4) and is a declarative language. The language was created to overcome some difficulties in other declarative GUI languages that use XML, such as XUL or XAML.

Though Snow allows someone to program a GUI, it does not address other

\(^9\)http://common-lisp.net/project/snow/
aspects of the software development process. In addition, it addresses creating interfaces for multiple devices through using a DSL. In contrast our DSL is used to cement an interface between the UX professional and the coder in order to improve communication and role specialization.

### 8.2.6 Intentional Programming

This idea began in the 1990s and has progressed to today. Whereas much of software today is created using OOP techniques, this method departs from traditional OOP by modifying the editor, the compiler, and introducing the concept of intentions. With OOP we create classes, which are viewed by a textual editor or IDE and then compile these classes into an executable. With Intentional Programming, a class is created, but in addition to the class itself, a viewer for the class must be created along with a parser, a version control component, etc. By including with the class a bunch of functionality that is now concentrated in tools, we can create a higher level of programming IDE. For example, a decision class could be represents at a table instead of text, and a graph class can be represented with a graphical diagram. The advantages here are that now different representations can easily be mixed and matched to create a program that have better overall comprehensibility—instead of attempting to read graph textual code, we can just look at the actual graph picture.

The company Intentional Software\(^\text{10}\) promotes the concept of intentional programming. Their overall view is that this method will allow one to have a domain expert program the software on a domain level and the programmer will write and generate the underlying code for the domain. An example of their approach is shown in the pension workbench (Figure 8.5).

\(^{10}\text{http://www.intentsoft.com}\)
The overall approach of intentional programming is very much like our methodology. However, the focus here is on getting the domain expert a way to code an application themselves. The domain representation is now the interface between the code and the domain language. In contrast, our approach is to focus on the usability of the method by involving a UX professional to create the UI instead of a having a domain expert to code the application.

### 8.2.7 Separation-Based Architectural Patterns

Separation-Based Architecture Pattern (SBAP) isolate usability concerns to a UI component of the software architecture. Some attention has been paid to using a DSL to communicate between the GUI and the actual application. Microsoft has registered a patent for using Windows Powershell commandlets for a UI \[154\]. Traditionally, a method of wrapping a GUI around a command line tool to create a simpler interface has been called GUI Wrapper programs. An essential source of this pattern is that the UI code is conceptually different than the machine code and that inspires the

---

developers to separate these parts.

8.2.8 GUI Builders

GUI Builders have provided a simple graphical building block interface for GUI interfaces. They have provided a high degree of ease of use in creating a GUI. Though there are many of these builders in existence, there do not seem to be simple GUI builders where programming is not needed in order to create the final GUI. And though some GUI builders do pay attention to usability, such as Microsoft Expression Web’s validator tool, usability is not a focus of any builder we are aware of.

Our focus is aligned with the overall concept of a GUI builder. A GUI builder does free the programmer from knowing quite a lot of programming in order to construct the GUI. However, there is still quite a bit of programming needed. While the general GUI builder adds a helpful graphical interface for GUI construction, our methodology attempts to remove the need to understand programming from the construction of the entire GUI. And, although we have not found an optimal GUI builder to allow us to implement our method in this way, our GUI builder would be graphical and require only knowledge of UX and the DSL interface for that particular application.

8.2.9 Other Research

A number of applications have attempted to use a DSL to increase the flexibility of a traditional application. Some people used DSLs to create an elevator application and allow people to manipulate that application with a DSL [195]. As enterprise applications are typically composed of multiple independent units, some researchers have
addressed this by creating a DSL for these independent units to communicate [174]. Others have used DSLs to integrate multiple applications [16].

8.2.10 Comparison

Here we compare and contrast the various approaches identified in the previous section to provide an overall picture of how our methodology is positioned in relation to existing methods. A listing of the different approaches with brief descriptions is shown in Table 8.1 for easy reference. We decided to compare the roles of people involved with the related software development and use this comparison in order to illuminate the reader as to the differences between the various approaches. However, take note that we are by no means experts in any of the other approaches and that this comparison table represents our current limited knowledge of these methods. Table 8.2 shows that our method is the only one to specify a UX Professional as the one to create the UI and this, we believe, is the key distinguishing characteristic of our approach. The closest to our method in respect to who creates the GUI is the Game Maker Language which contains a simplified builder for video games. In fact, we looked for a GUI builder which would exclude the need to know programming and we see a need for a new type of GUI builder which excludes the user from a need to understand code—the Game Marker Language GUI and other game creation platforms of this ilk are a good representation of the type of GUI builder that we require. So the UI creator and the GUI Builder are closely related and the game creation platforms in existence may provide a fertile source of ideas for how to create this new type of GUI builder. As it can be seen, our proposed method is more generic, or flexible, as it can involve using various types of GUI builders.
<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plastic User Interfaces</td>
<td>This method involves using MDE to generate UIs, while preserving usability</td>
</tr>
<tr>
<td>Usability-Supporting Architectural Patterns</td>
<td>A list of patterns to discuss during design stage to avoid a late-stage need for usability modifications</td>
</tr>
<tr>
<td>Game Maker Language</td>
<td>Provides a DSL for creating a video game and subsequently port to different platforms using MDE</td>
</tr>
<tr>
<td>Linden Scripting Language</td>
<td>Allows users of the Second Life game to write scripts to control their games</td>
</tr>
<tr>
<td>User Interface Markup Languages</td>
<td>Attempts to make a DSL for creating UIs</td>
</tr>
<tr>
<td>Intentional Programming</td>
<td>Focused on creating a platform to enable domain experts to program</td>
</tr>
<tr>
<td>Separation-Based Architectural Patterns</td>
<td>Isolate usability concerns to a UI component</td>
</tr>
<tr>
<td>GUI Wrapper</td>
<td>Uses a GUI to control a program through a console-based command language</td>
</tr>
<tr>
<td>GUI Builder</td>
<td>Allows drag and drop creation of a significant portion of a UI</td>
</tr>
<tr>
<td>Our Method</td>
<td>Uses a DSL with a separation-based UI architecture</td>
</tr>
</tbody>
</table>
Table 8.2: Approach comparison: L=Layman, P=Programmer, DE=Domain Expert, GD=Game Designer, UXP=UX Professional

<table>
<thead>
<tr>
<th>Test</th>
<th>Who is the user?</th>
<th>Can the user program?</th>
<th>Who creates the UI?</th>
<th>How is it created?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plastic User Interfaces</td>
<td>L</td>
<td>No</td>
<td>P</td>
<td>MDE</td>
</tr>
<tr>
<td>Usability-Supporting Architectural Patterns</td>
<td>L</td>
<td>No</td>
<td>P</td>
<td>Code</td>
</tr>
<tr>
<td>Game Maker Language</td>
<td>GD</td>
<td>Some</td>
<td>GD</td>
<td>Builder</td>
</tr>
<tr>
<td>Linden Scripting Language</td>
<td>L</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>User Interface Markup Languages</td>
<td>P</td>
<td>Yes</td>
<td>P</td>
<td>Script</td>
</tr>
<tr>
<td>Intentional Programming</td>
<td>DE</td>
<td>No</td>
<td>DE</td>
<td>Workbench</td>
</tr>
<tr>
<td>Separation-Based Architectural Patterns</td>
<td>L</td>
<td>No</td>
<td>P</td>
<td>Code</td>
</tr>
<tr>
<td>GUI Wrapper</td>
<td>L</td>
<td>No</td>
<td>P</td>
<td>Code</td>
</tr>
<tr>
<td>GUI Builder</td>
<td>L</td>
<td>No</td>
<td>P</td>
<td>Drag-n-Drop with Code</td>
</tr>
<tr>
<td>Our Method</td>
<td>L</td>
<td>No</td>
<td>UXP</td>
<td>Builder</td>
</tr>
</tbody>
</table>
8.3 Discussion: Benefits, Drawbacks, Other Opportunities, and Impacts

The approach presented here provides the following benefits. Communication is improved between the UX designers and the programmers by focusing on the DSL as an intermediate language (i.e. avoids those with low communication ability from playing games via jargon). Increased role specialization helps to increase the productivity of UX designers and programmers, which may improve the product. The method is (1) compatible with web applications by using the DSL as a protocol between UI client and the application server, (2) accommodates the increasingly diverse UIs such as mobile, voice, tactile, etc. by allowing separate UIs to be created without affecting the application code, (3) supports testing at the DSL level, and (4) provides for simple tracing of requirements.

Some drawbacks are inherent in our proposed approach. It requires a UX professional and a programmer. In approaching the problem of platform diversity, the UI will typically not be generalizable such as UsiXML, but would instead be a GUI builder that is simple enough for the UX professional to understand. It may be difficult to use this method if one is dependent upon some software framework such as dotNet, because the use of this method may invalidate some advantages of any particular software framework. Using the DSL with code generation may invalidate the ability to finely tune the code (i.e. for special speed or other considerations) or the UI for any specialized purpose beyond its general realization. The DSL now becomes an important artifact in the software application and therefore the DSL needs to be designed well and may require a language designer if things get complex. Certain cross cutting USAPs will still be a problem if they have not been considered during
the design phase.

Currently, multiple existing options to this approach are used. Programmers may code the UI but are trained to be submissive to UX desires and not use confusing jargon. The dotNet Framework, Java Webstart, DJANGO and other platforms for development provide tools to reduce the typical workload. Many web development platforms have begun to work from a Model-View-Controller (MVC) conceptual framework which separates the UI from application code. Agile techniques are also an option which have gained a lot of popularity and focus on short increments and cooperation among the employees working on the application rather than a defined process or tool.

If a team decides to use this methodology they can expect the following impacts on the applications. The long term impact will be improved usability through the UX approach and improved programmer productivity through role specialization and clear communication of application requirements via the DSL. Overall, the team will accomplish UI flexibility through freedom from a specific platform, and improved communication between designers and programmers. In the short term, the team may experience confusion in adapting to this methodology, especially if they have not previously done UX, traditional design then code methods, DSL creation, or code generation. Breaking from the current framework paradigm of a majority of software producers today could cause a good bit of discomfort in departing from the well known and supported frameworks.
8.4 Conclusions

We have identified our method as having the benefit of improving communication among UX designers and programmers, improving productivity of UX designers and programmers, accommodating diverse platforms for UIs, supporting use over the internet, supporting requirements tracing, and supporting automated testing. Although our method solves many problems, other beneficial options to our approach are to assign a UI programmer to implement the UX designer’s wishes, use a modern framework to reduce workload, follow a MVC pattern, or incorporate Agile methods. Many of the drawbacks of our approach seem to be related to the size of the team to work on the software in that we require a UX designer, a programmer, and possibly a language designer. Another potential drawback is the lack of easy to use UI builders that do not require the user to know how to program. Additionally, the use of frameworks with this method may defeat some of the inherent benefits of the approach by constraining certain choices. The use of a code generation framework will make the code more general, but also less efficient and this can be a problem in real-time or other constrained environments. We expect that the impacts of this method to improve the productivity of developing software. Our comparison of similar approaches to development really highlights the somewhat unique idea of having a UX designer design the UI with a tool that does not require any knowledge of programming. Our evaluation is based on our knowledge of the software engineering field and therefore our analysis here may be incomplete due to the impossibility of knowing everything about such a large and ever expanding field as software engineering.
Chapter 9

Evaluation of the Application

9.1 Introduction

In this chapter, we evaluate our proposed methodology by assessing the results of applying the methodology, that is, by evaluating the NetCDF application created following our approach. Specifically, this application is for extracting subsets of data from NetCDF files (for brevity, we call this application the NetCDF application or just application throughout this dissertation). The components of this evaluation include a usability study that informed our approach, a usability analysis, specific results of the code generation component of our approach, and a comparative analysis of tools that could have been used to create the NetCDF application.

9.2 Informative Usability Study

The NetCDF application we designed consists of giving users the ability to see what is in a NetCDF file and the ability to subset a NetCDF file. We saw these two
characteristics to be quite similar to those of the NCCP data search and download web page. Allowing users to see what is in a NetCDF file is provided in NCCP with the search page options. And, users subsetting a NetCDF file is analogous to users downloading a subset of data from the NCCP. It is due to this similarity that we were able to get insights from a usability study performed with the NCCP. The search page of the NCCP has also been the subject of an analysis regarding accessibility [67] and usability.

For the usability test, we worked with actual climate scientists. This usability test was an A/B test comparing an all text version and a text/graphics version of the search page. Users followed an Institutional Review Board (IRB) approved protocol while being timed and observed. The timings and observations were recorded, compiled, analyzed, and reported. Our results indicated that the all text version was satisfying for users to work with and made them quicker at executing searches. More details about this study are provided in Appendix A.

9.3 Usability Analysis

Once we had performed the informative usability test, we attempted to create a search UI that included our learning from that test. In addition to that, during our research with search UIs, we found an interface called a natural language output interface [35]. This interface was developed following our methodology and we tested it against the text-oriented search UI from the informative test (details are provided in Appendix B). From the test we found that the natural language output version is more satisfying to climate scientists than the text-oriented search UI. We also found out that climate scientists can execute tasks up to twice as fast on the natural language output search
UI. The result of this test is that we will choose a natural language search UI in our final application artifact. Overall, the test results provide positive indications of the benefits of applying our approach. These results also provide direction for others working with climate scientists and search UIs thereby having a broader impact than just this application.

9.4 Code Generation

We have two stages of code generation to address here. The first stage is in the Eclipse IDE with Xtext, which generates our parser generator for the second stage of code generation. This first stage requires us to define the grammar and write the associated code templates and after building we get a Java Jar file which accepts DSL scripts as input and generates Java code as output. Table 9.1 summarizes the Lines of Code (LOC) measurement which results in a code generated:written ratio of 10:1. The Java Jar file can then be used to read scripts and create Java source code files (Table 9.10). The code generated:written ratio at this second stage was 25:1.

9.5 Comparative Analysis

To give the reader an understanding of how our climate science tool (created using our approach) relates to other such tools, we present several tools that are currently available to help climate scientists work with NetCDF files and compare them with ours. In fact, there are a great number of tools which manipulate NetCDF files and not all of them are listed here as the time it would take to compare all of them would be quite large, but we do provide a spectrum of the free tools existing today. We
Table 9.1: Parser code generation

<table>
<thead>
<tr>
<th>artifact</th>
<th>Details</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>written code</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grammar definition</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>templates</td>
<td></td>
<td>476</td>
</tr>
<tr>
<td>total lines written</td>
<td></td>
<td>530</td>
</tr>
<tr>
<td><strong>generated code</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>org.xtext.demo.mydsl.ncDsl</td>
<td>Table 9.2</td>
<td>282</td>
</tr>
<tr>
<td>org.xtext.demo.mydsl.ncDsl.impl</td>
<td>Table 9.3</td>
<td>2005</td>
</tr>
<tr>
<td>org.xtext.demo.mydsl.ncDsl.util</td>
<td>Table 9.4</td>
<td>363</td>
</tr>
<tr>
<td>org.xtext.demo.mydsl.parser.antlr</td>
<td>Table 9.5</td>
<td>35</td>
</tr>
<tr>
<td>org.xtext.demo.mydsl.parser.antlr.internal</td>
<td>Table 9.6</td>
<td>2238</td>
</tr>
<tr>
<td>org.xtext.demo.mydsl.serializer</td>
<td>Table 9.7</td>
<td>241</td>
</tr>
<tr>
<td>org.xtext.demo.mydsl.services</td>
<td>Table 9.8</td>
<td>467</td>
</tr>
<tr>
<td>org.xtext.demo.mydsl.validation</td>
<td>Table 9.9</td>
<td>12</td>
</tr>
<tr>
<td>total lines generated</td>
<td></td>
<td>5643</td>
</tr>
</tbody>
</table>

Table 9.2: Code generated for org.xtext.demo.mydsl.ncDsl

<table>
<thead>
<tr>
<th>generated code</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.xtext.demo.mydsl.ncDsl</td>
<td></td>
</tr>
<tr>
<td>AsciiFile.java</td>
<td>7</td>
</tr>
<tr>
<td>Comparator.java</td>
<td>7</td>
</tr>
<tr>
<td>Extraction.java</td>
<td>10</td>
</tr>
<tr>
<td>GetDetails.java</td>
<td>9</td>
</tr>
<tr>
<td>GetTimeRemaining.java</td>
<td>7</td>
</tr>
<tr>
<td>GetVar.java</td>
<td>9</td>
</tr>
<tr>
<td>NcDsl.java</td>
<td>10</td>
</tr>
<tr>
<td>NcDslFactory.java</td>
<td>21</td>
</tr>
<tr>
<td>NcDslPackage.java</td>
<td>147</td>
</tr>
<tr>
<td>NetCdfFile.java</td>
<td>7</td>
</tr>
<tr>
<td>Port.java</td>
<td>7</td>
</tr>
<tr>
<td>Sink.java</td>
<td>7</td>
</tr>
<tr>
<td>Source.java</td>
<td>7</td>
</tr>
<tr>
<td>Transformation.java</td>
<td>9</td>
</tr>
<tr>
<td>Transition.java</td>
<td>5</td>
</tr>
<tr>
<td>Workflow.java</td>
<td>13</td>
</tr>
<tr>
<td>lines generated</td>
<td>282</td>
</tr>
</tbody>
</table>
Table 9.3: Code generated for org.xtext.demo.mydsl.ncDsl.impl

<table>
<thead>
<tr>
<th>generated code</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.xtext.demo.mydsl.ncDsl.impl</td>
<td></td>
</tr>
<tr>
<td>AsciiFileImpl.java</td>
<td>84</td>
</tr>
<tr>
<td>ComparatorImpl.java</td>
<td>84</td>
</tr>
<tr>
<td>ExtractionImpl.java</td>
<td>133</td>
</tr>
<tr>
<td>GetDetailsImpl.java</td>
<td>133</td>
</tr>
<tr>
<td>GetTimeRemainingImpl.java</td>
<td>105</td>
</tr>
<tr>
<td>GetVarImpl.java</td>
<td>133</td>
</tr>
<tr>
<td>NcDslFactoryImpl.java</td>
<td>132</td>
</tr>
<tr>
<td>NcDslImpl.java</td>
<td>156</td>
</tr>
<tr>
<td>NcDslPackageImpl.java</td>
<td>310</td>
</tr>
<tr>
<td>NetCdfFileImpl.java</td>
<td>84</td>
</tr>
<tr>
<td>PortImpl.java</td>
<td>84</td>
</tr>
<tr>
<td>SinkImpl.java</td>
<td>105</td>
</tr>
<tr>
<td>SourceImpl.java</td>
<td>105</td>
</tr>
<tr>
<td>TransformationImpl.java</td>
<td>128</td>
</tr>
<tr>
<td>TransitionImpl.java</td>
<td>17</td>
</tr>
<tr>
<td>WorkflowImpl.java</td>
<td>212</td>
</tr>
</tbody>
</table>

lines generated                   2005

Table 9.4: Code generated for org.xtext.demo.mydsl.ncDsl.util

<table>
<thead>
<tr>
<th>generated code</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.xtext.demo.mydsl.ncDsl.util</td>
<td></td>
</tr>
<tr>
<td>NcDslAdapterFactory.java</td>
<td>174</td>
</tr>
<tr>
<td>NcDslSwitch.java</td>
<td>189</td>
</tr>
</tbody>
</table>

lines generated                   363

Table 9.5: Code generated for org.xtext.demo.mydsl.parser.antlr

<table>
<thead>
<tr>
<th>generated code</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.xtext.demo.mydsl.parser.antlr</td>
<td></td>
</tr>
<tr>
<td>NcDslAntlrTokenFileProvider.java</td>
<td>9</td>
</tr>
<tr>
<td>NcDslParser.java</td>
<td>26</td>
</tr>
</tbody>
</table>

lines generated                   35
Table 9.6: Code generated for org.xtext.demo.mydsl.parser.antlr.internal

<table>
<thead>
<tr>
<th>generated code</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.xtext.demo.mydsl.parser.antlr.internal</td>
<td></td>
</tr>
<tr>
<td>InternalNcDslLexer.java</td>
<td>949</td>
</tr>
<tr>
<td>InternalNcDslParser.java</td>
<td>1289</td>
</tr>
<tr>
<td>lines generated</td>
<td>2238</td>
</tr>
</tbody>
</table>

Table 9.7: Code generated for org.xtext.demo.mydsl.serializer

<table>
<thead>
<tr>
<th>generated code</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.xtext.demo.mydsl.serializer</td>
<td></td>
</tr>
<tr>
<td>NcDslSemanticSequencer.java</td>
<td>202</td>
</tr>
<tr>
<td>NcDslSyntacticSequencer.java</td>
<td>39</td>
</tr>
<tr>
<td>lines generated</td>
<td>241</td>
</tr>
</tbody>
</table>

Table 9.8: Code generated for org.xtext.demo.mydsl.services

<table>
<thead>
<tr>
<th>generated code</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.xtext.demo.mydsl.services</td>
<td></td>
</tr>
<tr>
<td>NcDslGrammarAccess.java</td>
<td>467</td>
</tr>
</tbody>
</table>

Table 9.9: Code generated for org.xtext.demo.mydsl.validation

<table>
<thead>
<tr>
<th>generated code</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.xtext.demo.mydsl.validation</td>
<td></td>
</tr>
<tr>
<td>AbstractNcDslValidator.java</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 9.10: Script code generation

<table>
<thead>
<tr>
<th>artifact</th>
<th>Details</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>written code</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>dsl script</td>
<td>Figure 7.7</td>
<td></td>
</tr>
<tr>
<td>total lines written</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>generated code</td>
<td></td>
<td>252</td>
</tr>
<tr>
<td>NetCdf.java</td>
<td></td>
<td>122</td>
</tr>
<tr>
<td>NetCdf_Sink.java</td>
<td></td>
<td>99</td>
</tr>
<tr>
<td>Predicate.java</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Workflow.java</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>total lines generated</td>
<td></td>
<td>252</td>
</tr>
</tbody>
</table>
have chosen tools which are close to ours in size and scope or they are tools that were mentioned by the climate scientists we have worked with. And though we do not know what software methodology was used to create these other tools, we will discuss the various advantages of our tool and how these advantages came to be as results of our methodological process.

Since we are not climate scientists, our evaluation of these tools is based on user concerns rather than the detailed abilities offered to climate scientists and our tool does not aim to compete with these various capabilities but instead it aims to provide a better interface for the user and to illustrate the effects of using our prescribed methodology. The limited comparison we presented here serves that purpose and gives the reader a good view of how our tools UX features compare with currently available tools.

9.5.1 ncdump and ncgen

A command line program, ncdump\(^1\), allows users to extract data from a NetCDF file and export it to a network Common Data form Language (CDL) or NetCDF Markup Language (NcML) format. A second program, ncgen\(^2\), can be used to create a NetCDF file from CDL.

Now, we provide an example of using these tools. In using ncdump, we first want to know what variables exist in our sample.nc file (Figure 9.1). Next, knowing what variables are there, lets get only the ‘precip’ variable and put it into a CDL file (Figure 9.2). And we subsequently create our sample-precip.nc from the CDL file

\(^1\)Unidata. ncdump. UCAR Community Programs and the University for Atmospheric Research. [www.unidata.ucar.edu/software/netcdf/docs/ncdump-man-1.html](http://www.unidata.ucar.edu/software/netcdf/docs/ncdump-man-1.html)

\(^2\)Unidata. ncgen. UCAR Community Programs and the University for Atmospheric Research. [www.unidata.ucar.edu/software/netcdf/docs/ncgen-man-1.html](http://www.unidata.ucar.edu/software/netcdf/docs/ncgen-man-1.html)
Ubuntu:$ ncdump -k sample.nc
classic
Ubuntu:$ ncdump -h sample.nc
netcdf sample {
  dimensions:
    leveldim = 1 ;
    latdim = 139 ;
    londim = 139 ;
    timedim = 1 ;
  variables:
    float precip(latdim, londim) ;
      precip:units = "mm" ;
    float tmin(latdim, londim) ;
      tmin:units = "K" ;
    float tmax(latdim, londim) ;
      tmax:units = "K" ;
    float tmean(latdim, londim) ;
      tmean:units = "K" ;
    float qmean(latdim, londim) ;
      qmean:units = "kg.kg^-1" ;
    float u10mean(latdim, londim) ;
      u10mean:units = "m.s^-1" ;
    float v10mean(latdim, londim) ;
      v10mean:units = "m.s^-1" ;
    float swdownmean(latdim, londim) ;
      swdownmean:units = "W.m^-2" ;
}

Figure 9.1: sample.nc contents

(Figure 9.3).

9.5.2 MATLAB

Offers a function based extension to their language for NetCDF files\(^3\). The user must be familiar with the MATLAB language (see Figure 9.4). We see here that the users


Ubuntu:$ ncdump -v precip sample.nc > sample.cdl

Figure 9.2: Using ncdump
need to be able to compose a list of commands to get the information they want from
the NetCDF file.

The cdfread function could shorten the program above by allowing one to read
a NetCDF file into an array in one step (Figure 9.5).

9.5.3 EverView Slice and Dice Tool

EverVIEW is an attempt by Joint Ecosystem Modeling Group (JEM) and the United
States Geological Survey (USGS) to create a tool to work with NetCDF files\(^4\). The
project is supported by the USGS and has three goals: subset NetCDF files, view
tabular data, and convert to Comma-Separated Values (csv) files. Figure 9.6 shows
the interface for viewing the table data from a NetCDF file. The project is currently
in the Beta stage of development [1,163]. The focus of the tool is to subset a NetCDF
file with regards to time or geography\(^5\).

9.5.4 Ncview

The Ncview application\(^6\) runs on Unix systems and provides a graphical picture of the
contents of a NetCDF file. Upon starting the program, the user is confronted with a
command screen (Figure 9.7). If the user chooses one of the variables, such as ‘precip’
then a plot of that variable is shown (Figure 9.8). Subsequently clicking on any point

\(^5\)JEM www.jem.gov/Modeling/SliceAndDice accessed 2012-06-08
pierce/ncview_home_page.html accessed June 2013
ncid = netcdf.open('example.nc','NOWRITE');

//get information about the variables in the file
info = cdflib.inquire(cdfid)

info =
    
    encoding: 'IBMPC_ENCODING'
    majority: 'ROW_MAJOR'
    maxRec: 23
    numVars: 6
    numvAtts: 1
    numgAtts: 3

//get details about a specific variable
info = cdflib.inquireVar(cdfid,0)

info =
    
    name: 'Time'
    datatype: 'cdf_epoch'
    numElements: 1
    dims: []
    recVariance: 1
    dimVariance: []

// Get the name of the first variable.
[varname, xtype, varDimIDs, varAtts] = netcdf.inqVar(ncid,0);

// Get variable ID of the first variable, given its name.
varid = netcdf.inqVarID(ncid,varname);

// Get the value of the first variable, given its ID.
data = netcdf.getVar(ncid,varid)

data =
    6.0221e+023

// Determine the data type of the output value
whos data
Name     Size      Bytes  Class Attributes
       data    1x1        8  double

netcdf.close(ncid);

Figure 9.4: Matlab code for opening and inspecting a NetCDF file
```matlab
var_long_lat = cdfread('example.cdf','Variable',{"Longitude","Latitude"});
```

```
whos var_long_lat
Name      Size    Bytes  Class       Attributes
var_long_lat  1x2    128    cell
```

Figure 9.5: Matlab code to read a NetCDF file

Figure 9.6: EverVIEW file contents view
of the plot brings up a detailed 2D graph of the ‘precip’ variable (Figure 9.9).

### 9.5.5 Data Basin

Data Basin\(^7\) allows users to analyze and map data from NetCDF files. This is an online tool and one needs to have an account to use the tool and special permission to be able to upload one’s own NetCDF files rather than work with the data that

\(^7\)Data Basin. Conservation Biology Institute [databasin.org](http://databasin.org) accessed June / 2013
Figure 9.8: Ncview ‘precip’ plot

Figure 9.9: Ncview ‘precip’ detailed graph
Data Basin provides (see video\textsuperscript{8}). The tool is free and it does not require sophisticated computer knowledge of programming languages and such to use.

The process of using Data Basin requires the user to first upload a file, then they have to select from a number of settings (Figure 9.10 and Figure 9.11). Results are then presented to the user (Figure 9.12).

9.5.6 Interactive Data Language

Interactive Data Language (IDL) was created to enable users to analyze data. It supports the use of many types of files used in the scientific fields such as Hierarchical Data Format 5 (HDF5), NetCDF-3, NetCDF-4, Common Data Format (CDF), Hierarchical Data Format - Earth Observing System (HDF-EOS), Hierarchical Data Format 4 (HDF4), GRidded Binary (GRIB), and others. The language is ready to access data in large files and can even access data from remote locations via the Hyperbassin. NetCDF Data in Data Basin. YouTube. \url{http://www.youtube.com/watch?v=L3ydfi5BqEk} accessed June / 2013

\textsuperscript{8}Hyperbassin. NetCDF Data in Data Basin. YouTube. \url{http://www.youtube.com/watch?v=L3ydfi5BqEk} accessed June / 2013
Figure 9.11: Data Basin import settings, page 2

Figure 9.12: Data Basin results
perText Transfer Protocol (HTTP) and File Transfer Protocol (FTP). Figure 9.13 shows a sample of IDL to get data from a NetCDF file.

9.5.7 Comparison

When comparing tools, we do so from a user perspective because our knowledge of how these tools were designed or coded is very limited. By comparing from a user’s point of view we also hope to achieve a picture of the actual utility of these tools to a user rather than a gross listing of various features. Table 9.11 shows all of the tools and the presence or absence of specific user related features. We broke down the required information by the user into different mental loads that are pushed onto the user when using a tool. So, if a tool requires command line knowledge then the user must know how to use some textual commands in addition to understanding the basic model of how a console works (where the output goes, the limits of ASCII, etc.). If programming knowledge is required then the user must know commands and basic programming flow concepts such as for loops. If the user needs to know about NetCDF files then he/she must understand the concept of a file and details about how the NetCDF format works. We chose these dimensions of comparison to elucidate the UX for the climate scientists using the tools. Though the tools have many detailed capabilities, our comparison is limited in line with our understanding of the tools and our goal of an exposition of our methodologies effects. By looking at the table, we can see that our tool compares favorably to other tools in the amount of user knowledge demanded for use. The comparison is favorable to our method because it stems from our proposed approach.
cdfid = ncdf_open(File)
inq= ncdf_inquire(cdfid)
;Read the attributes of the flight. the amount of attributes is in
  the inq structure
for i=0,inq.NGATTS-1 do begin
  attname = ncdf_attname(cdfid, i, /GLOBAL)
  ;print, attname
  ncdf_attget, cdfid, attname, attvalue, /GLOBAL
  ;print, attname, attvalue
  ;Save this attributes
endfor
names=strarr(inq.nvars)
longname=strarr(inq.nvars)
units=strarr(inq.nvars)
dim=strarr(inq.nvars)
for i=0,inq.nvars-1 do begin
  info=ncdf_varinq(cdfid,i)
  names(i)=info.name
  print(names(i))
  FOR j=0,info.natts-1 DO BEGIN
    attname = ncdf_attname(cdfid,i,j)
    IF (attname EQ 'units') THEN BEGIN
      ncdf_attget, cdfid, i, attname, attvalue
      units(i)= string(attvalue)
      print(units(i))
    END
    IF (attname EQ 'long_name') THEN BEGIN
      ncdf_attget, cdfid, i, attname, attvalue
      longname(i)= string(attvalue)
    END
  ENDFOR
ENDFOR
ncdf_varget,cdfid,'level',levelshum
ncdf_varget,cdfid,'lat',xlat
ncdf_varget,cdfid,'lon',xlong
ncdf_varget,cdfid,'time',times
retrivevar='shum'
ncdf_attget, cdfid, retrivevar, 'add_offset', add_off
ncdf_attget, cdfid, retrivevar, 'scale_factor', xscale
ncdf_varget, cdfid, 'level', levelshum
+ncdf_varget, cdfid, retrivevar, tos; offset=[lonswhere(0),latswhere(0),0], count=[n_elements(lonswhere),n_elements(latswhere),n_elements(times)]
shum=xscale*tos+add_off
+;ncdf_varget, cdfid, retrivevar, tos, offset=[lonswhere(0),latswhere(0),0], count=[n_elements(lonswhere),n_elements(latswhere),n_elements(times)]
cdf_close, cdfid

Figure 9.13: Interactive Data Language sample
<table>
<thead>
<tr>
<th>Test</th>
<th>Who is the user?</th>
<th>Requires command line knowledge?</th>
<th>Requires programming knowledge?</th>
<th>Requires knowledge about NetCDF format?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncdump and ncgen</td>
<td>CS</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>MATLAB</td>
<td>CS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>EverView Slice and Dice</td>
<td>CS</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ncview</td>
<td>CS</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Data Basin</td>
<td>CS</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Interactive Data Language</td>
<td>CS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Our Method</td>
<td>CS</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Table 9.12: Benefits and Drawbacks

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) UI based on usability studies</td>
<td>1) Complex build process</td>
</tr>
<tr>
<td>2) Code generation aids in productivity</td>
<td>2) Development tool dependency</td>
</tr>
<tr>
<td>3) Favorable comparative analysis</td>
<td>3) Climate scientist awareness</td>
</tr>
<tr>
<td>4) Simple UI which does not rely on information outside of the climate science realm</td>
<td>4) Development tools are quickly evolving and require developers to learn them.</td>
</tr>
<tr>
<td>5) Novice climate scientists will find our tool simpler than learning programming</td>
<td>5) Development tools, such as Xtext are in a niche area and skills will not be as transferable thereby decreasing developer motivation</td>
</tr>
</tbody>
</table>

9.6 Discussion: Benefits, Drawbacks, Other Opportunities, and Impacts

We performed a benefit / drawback analysis of the application to help assess its current state of development (Table 9.12). The benefits offered by this application are a search UI based on real usability studies, code generation aiding in productivity gains, a favorable comparative analysis, and a simple UI which does not require knowledge outside of the realm of climate science. The drawbacks are a complex build process regarding generation and the need to understand how to write grammars, dependency on Xtext and Eclipse, and the awareness of climate scientist that their current tools could be improved upon.

Other options are available to users besides using a tool such as described here. There are many tools that can be used to subset NetCDF files and those in this comparison are a small sample. Some climate scientists may feel great independence and freedom in being able to program and therefore resist changing to a simpler application. If a particular organization is large enough, they may assign a NetCDF
expert to do the work of subsetting NetCDF files and consequently there would not be as much of a productivity gain by the use of our application.

There are a few expected impacts of creating a final version of our tool to release to climate scientists. First, we expect that some climate scientists may find the tool helpful. The use of the tool may at first be quite limited due to awareness of its existence in the quite filled world of NetCDF tools. Even if scientists become aware of our tool, it will not necessarily be to their liking as they might not want to learn something new. At the same time, in order for the tool to slowly increase in users, it will need to be updated with advances in NetCDF files, new operating system platforms, error fixes, and change requests. Due to a period of a low number of users and a need to fix errors and other code changes, we expect that the application would need a fair amount of sweat equity or money to keep it alive and progressing in the market of NetCDF tools.

\section*{9.7 Conclusions}

In order to analyze our application, we have performed a set of analyses. First, we relied on a usability study of the NCCP to inform our directions in creating a search UI for NetCDF files. Then, we checked the results of applying the methodology with a second small usability study. To get an understanding of the benefits from our code generation, we also measured the ratio of code generated to code written. We have also provided a comparative analysis of our tool with several related tools. Finally, we provided a detailed discussion of the benefits and drawbacks of the tool created following our proposed methodology.
Chapter 10

Conclusions and Future Work

10.1 Introduction

We have presented a new methodology, the concepts and steps behind the creation of that methodology based on prior work, and provided an application of the methodology. We have also evaluated the methodology through comparison with related work, usability studies, and analysis of its results. Our hypothesis has been that a separation-based UI architecture can enhance the process of software development. A key to our proposed approach is the use of a DSL to bridge the gap between UI design and writing code. Much of the research, development, and testing conducted has been promising regarding the truth of our hypothesis, but inherently more work remains to be done to fully prove and evolve our methodology.
10.2 Conclusions

The current practice of software engineering is to create a software application to meet functional and non-functional requirements. Once the requirements are known, architectures of the software are specified to satisfy those requirements. To enhance the usability of applications, a UX professional will be put onto a design team to inform and lead the development of the UI. A programmer is then responsible for coding the solution’s architecture and specified UI to bring the software artifact into existence.

In analyzing this process of software development, we came up with a key question regarding the difficulties of UI implementations. A number of technical constraints exist depending upon the framework used, platform used, and tools used. In addition to technical constraints, there are developer limitations due to programmer experience, whether they are responsible for the entire UI creation or working with a UX professional. Could the developer problems be addressed with an architecture that is not only focused on satisfying the functional and non-functional requirements, but one that also satisfies the needs and limitations of the developers themselves? In this work, we have shown how a separation-based UI architecture can be enhanced by considering role specialization and thereby leading to a potentially more suitable development process and consequently a better final software product.

The results we have developed in this dissertation show that it is possible to consider the role specialization of developers and to architect a software solution that enhances appropriate role separation. Our derived architecture starts with the widespread separation-based UI architecture and we successively incorporated our premises regarding role specialization and communication gaps to arrive at an
enhanced separation-based UI with a DSL between the UI and Code components (Figure 5.9). While prior work has made great gains in making UI production easier and more flexible, all of the work we surveyed concentrated on the UI being created by a programmer whereas we focused on the UX professional creating the UI. Our solution does require specialized developers; a programmer, a UX professional, and a DSL designer which, for the moment, we assumed to be the programmer. The solution also incorporates sophisticated techniques involving code generation and MDE. Our approach also complicates communication between the UI and Code by using a DSL rather than the more typical API. While some parts of development are more complex in our approach, we believe that our enhanced separation-based UI via role specialization has shown promise in addressing UI considerations in a new development architecture.

To validate our proposed architecture, we created an application (a NetCDF subsetting application) following our method’s steps. The implementation of our approach was comprehensively illustrated via that application. What became evident during this implementation was a dearth of UI builders that do not require knowledge of programming, however, by assigning a programmer to be directed by our UX professional the problem was dealt with. In addition to this, the methodology is quite close to the way software is developed today and therefore does not require a lot of new learning tasks for current developers. During the creation of the application, the UI and Code development was quite independent and therefore met our objectives of allowing different experts to specialize and not have to learn someone else’s specialty. Our evaluation of the proposed methodology has generally confirmed our expectations. This evaluation has taken the forms of comparing the proposed method with related work, as well as assessing the results of applying the method through usability
tests, evaluation of code generation efficiency, and comparison with several tools that could have been used to create a similar application.

The importance and diversity of UIs has been progressing as electronics continue to shrink and increase in computational power. This means that the UX knowledge base is expanding alongside UI progress. At the same time, programmers are facing a need to focus on a more diverse world than the previous WINdows InTEL (Wintel) desktop dominated environment. Not only do we have a ubiquitous web presence, but ever more devices are now being connected to the web and these devices require paying attention to battery life, limited memory, and communications. As we have shown, the specialized skills of UX professionals and programmers are quite different and therefore we think our methodology provides an avenue for others to follow in order to improve the effectiveness of their development process. Our separation-based UI architecture is also very well suited to operating from a client/server and other distributed environments which are becoming more prevalent with the internet.

10.3 Future Work

In considering what remains for this work, we must discuss both work on the architecture and the developer’s experience. In the case of the UI component, we did not find a suitably flexible GUI builder that would not require the need to understand code. Having a suitable UI builder could really make our approach easier to implement. Though we have shown how to create the DSL, we still want to find a generic solution for passing the DSL messages back and forth from the UI and Code components. And, with regards to the Code component, we could work with different code
generation techniques to find a best of breed. For this dissertation, we had one person (the author) play the roles of UX professional and the programmer, but we need to test our methodology on separate people playing those different roles. By testing with developers, we can gain a valuable feedback loop to inform the development of the methodology. Further testing may additionally specialize the architecture, with three major roles rather than just two: UX professional, DSL designer, and code programmer.

The software environment is expanding every year due to more devices and different services that are offered. Though our methodology looks promising in the climate science environment, we still need to test to determine if it is effective in a number of other areas.

Our intention in the UX area was to support the specialization of professionals and to free them from needing to learn programming in order to get their jobs done. In the climate science domain, we see some opportunity to address this concern. Our application in this dissertation was to subset NetCDF files, but there are other files such as GRIB files that climate scientists must also deal with. While we have spent some time experimenting with Backus-Naur form (BNF) grammars to create software for reading scientific data files, much work can still be done here to take advantage of current code generation tools. The scientist must still download NetCDF files in order to use our tool, but in the future we might be able to entirely eliminate the users’ need to know anything about NetCDF files by performing a search of NetCDF repositories for them. In addition, many other applications both inside and outside the climate science domain could be developed to further evolve and fine-tune our proposed methodology.
Appendices
Appendix A

NCCP Usability Study

A.1 Introduction

This usability study is included as an appendix to provide a detailed account of tests performed in creating software tools for the climate science research. The study here actually represents a web portal used to provide scientists access to a store of data. The essential relation to our climate science tool is in that both interfaces need to provide climate scientists with a way to search for data. These search UIs are non-trivial and contain many intricate factors and we felt that usability testing was one avenue that provided substantial feedback with which to design an effective interface. All of the results here were used to inform the design of our proposed approach and ideas generated from the study were also fed back into the web portal to improve it as well.
A.2 Details

The Nevada Climate Change Portal (NCCP), see Figure A.1, is a project funded by the NSF Experimental Program to Stimulate Competitive Research (EPSCoR) to fund the study of climate change and how it affects Nevada. The site has been created and is currently being used to offer climate data to its users. As the maintainers and developers of the NCCP, we have embarked on a series of user studies in order to assess the effectiveness of parts of the NCCP. The information in this appendix contains the results of one of those studies. The usability study\(^1\) contained herein was conducted to evaluate the effectiveness of different search implementations on the NCCP website. The study provided empirical data of the effectiveness of these search pages with regards to real climate scientist users. The NCCP has many slightly different modifications of the search page and we used a text-based version (Figure A.2) along with a map-based version (Figure A.3).

The NCCP usability tests were conducted with 10 climate scientists who were working at Desert Research Institute (DRI). The test consisted of a series of 3 tasks instructing the user to search and download specific sets of data from the website. Users were both observed by the tester and they were timed. In addition to this, the users were asked to fill out questionnaires regarding their experience with the NCCP. The results are reported in two parts, a quantitative analysis and a qualitative analysis. First, we describe the quantitative analysis of user satisfaction and timing of users. The second qualitative part of this report describes the observations by the tester as to stumbling blocks that caused users difficulties during the tests. The final section of this report is the conclusion and it provides an opinion on what can be done

\(^1\)Nevada Climate Change Portal Usability Study, IRB protocol # S12-063, sponsored by NSF EPSCoR
Figure A.1: The NCCP homepage
Figure A.2: The text-based search page for the NCCP
Figure A.3: The map-based search page for the NCCP
to improve the site and additionally describes actions that might assist in conducting usability studies in the future.

### A.3 Quantitative Analysis

The comparison here uses statistical procedures to judge the significance of the findings. Overall, the significance is low because the two versions of the search page were quite close and the number of user tested was not able to easily show any difference.

#### A.3.1 Overall SUS

After finishing the three tasks for one version of the search page, the users would fill out a System Usability Scale (SUS). This questionnaire gives us some qualitative opinions of users satisfaction with the site. The questionnaire is scored on a 100 point scale and a paired-t test was used to compare versions of the search page. The average difference between Text and map was 9.25 (Figure A.4) in favor of Text with a confidence interval of (-1 to 19). The margin of error was 10.2%. These numbers are with respect to an 80% confidence interval.

#### A.3.2 Timing Test Results

Table B.1 shows the results for 80% confidence intervals. The high margin of error provides little confidence in these numbers. The map and text versions are so close that we need more users in order to distinguish the difference between them. One problem that is showing up here, as well, is the difficulty to get a 30 minute interval of the queried data. Users struggled with this, as seen by the observer during testing, and it provided a significant stumbling block for the majority of users. Only Tests
Figure A.4: SUS scores
2 and 3 requested a 30 minute interval and thereby it can be seen that this may be causing a significant part of the slowdown (average difference) in the table below. Another observed slowdown was in figuring out how to correctly choose the sites in the map version, but “Test 1” has a small 2.5 second difference. On subsequent Tests, most users already understood how to access the Site drop-down list and it would not be significant on Tests 2 and 3. With the large margins of error, any attempt to compare the observations to these timings appears to be doomed.

Table A.1: Timing results

<table>
<thead>
<tr>
<th>Site</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Difference (seconds)</td>
<td>2.5</td>
<td>32.5</td>
<td>88.67</td>
</tr>
<tr>
<td>Confidence Interval</td>
<td>-66 to 66</td>
<td>-139 to 72</td>
<td>-139 to -38</td>
</tr>
<tr>
<td>Margin of error</td>
<td>64%</td>
<td>106%</td>
<td>50.5%</td>
</tr>
<tr>
<td>p-value</td>
<td>0.96</td>
<td>0.64</td>
<td>0.05</td>
</tr>
<tr>
<td>N</td>
<td>6</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

A.3.3 Completion Rates

The following table of values were calculated with a 90% confidence interval using a McNemar Exact Test. The only information that might be gleaned here is that it appears that the Text version of the site allows users to learn it and perform queries more quickly with it as their learning improves. A significant slowdown on Test 3 resulted from the user needing to uncheck many data boxes in order to get one specific soil Temperature and this may be one reason for the poor performance of the map version in Test 3.
Table A.2: Completion rates

<table>
<thead>
<tr>
<th>Site</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chance the Completion rates are different</td>
<td>50%</td>
<td>75%</td>
<td>93.75%</td>
</tr>
<tr>
<td>Chance that Text has higher completion rate</td>
<td>75%</td>
<td>87.5%</td>
<td>96.88%</td>
</tr>
<tr>
<td>Chance that Map has higher completion rate</td>
<td>25%</td>
<td>12.5%</td>
<td>3.13%</td>
</tr>
</tbody>
</table>

A.3.4 Correctness Rates

Users were instructed to download the data resulting from their searches. This data was then analyzed to determine if they had successfully downloaded correct information (Table B.3). When comparing Text to Map using sampling we get Table B.4.

Table A.3: Correctness rates

<table>
<thead>
<tr>
<th>Counts</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Percentage</td>
<td>100%</td>
<td>67%</td>
<td>70%</td>
<td>75%</td>
<td>71%</td>
<td>83%</td>
</tr>
</tbody>
</table>

Table A.4: Correctness comparison

<table>
<thead>
<tr>
<th></th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chance the correctness is different</td>
<td>75%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Chance that Text has more correctness</td>
<td>87.5%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>Chance that Map has more correctness</td>
<td>12.5%</td>
<td>50%</td>
<td>50%</td>
</tr>
</tbody>
</table>
A.3.5 Mouse Measurement

We used a program called Mousotron\(^2\) to measure the distance the mouse traveled, the number of clicks, and the key strokes while performing the tasks on the search pages. This revealed a way to compare the amount of user effort needed to perform the searches (Table A.5). In each case an expert user was assumed and therefore the table here represents a minimal set of clicks, strokes, and movement to accomplish a task. For Test 3, the Map needs far mouse use than the Text version and this is due to the way that the Soil Temperature subcategories are all checked by default and require the user to uncheck all unwanted subcategories. Another anomaly is the difference in the number of key strokes for Text and Map in Test 3. This was due to the inability to enter the date via the keyboard for the end time, so it was done by clicking through the calendar instead. This was some particular unknown error of the interface and since it did not skew the test it was not regarded as a significant problem.

Table A.5: Mouse measurement

<table>
<thead>
<tr>
<th></th>
<th>mouse movement</th>
<th>left button</th>
<th>right button</th>
<th>key strokes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test 1:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>29</td>
<td>7</td>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td>Map</td>
<td>22</td>
<td>12</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td><strong>Test 2:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>30</td>
<td>11</td>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td>Map</td>
<td>26</td>
<td>12</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td><strong>Test 3:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>37</td>
<td>18</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>Map</td>
<td>186</td>
<td>147</td>
<td>0</td>
<td>11</td>
</tr>
</tbody>
</table>

A.4 Qualitative Analysis

During user testing, numerous difficulties became obvious. Firstly, it was noticed that climate scientists were much slower than the computer science students used for the pilot study. A computer science student took about 15 minutes to complete a test, but the climate scientist took 20-25 minutes to complete a test. The Climate Scientists appear to be much more conscientious of what they are doing than the computer science students and this may be part of the reason for the slowdown.

A list of difficulties observed is provided below and they are ordered in regards to the frequency of occurrence. Higher occurrence problems are listed first.

A.4.1 Contrast

Some of the buttons background and text seem to be very slight in contrast. Test the color contrasts and modify if too low.

A.4.2 “Download Data” Button

The Download Data button does not follow the flow of the users. This button is at the top of the screen, but the user progresses from top “Start Time” to the bottom search query choices. When they have made their choices, it takes them some time to find the button for downloading data. It is suggested that moving the download button into the natural flow of the user will enhance the usability of the web page.

Additionally, the Download button is constantly enabled even if the user has not specified any query terms. It might be better to enable this button only when the user has specified a valid query.
A.4.3  Query Errors

Errors in the Query are not shown directly and instead require the user to determine that there is a problem and then go search through their choices to ensure that the query choices are valid. For example, if we execute the following actions, we see problems with the resulting dialog.

1. The user chooses START TIME > END TIME (Figure A.5).

2. When the user clicks the “Download Data”

3. A dialog appears (Figure A.6).

The errors in this dialog box are that the “Start Formatting” button is enabled and appears clickable. It is suggested that the Start Time and End Times could be highlighted in red if there is such a conflict, immediately notifying the user of the exact problem. It is also suggested to not allow the user to download data if the query formulation is not complete.

A.4.4  Map location

The Map appearing on the right side of the search page causes confusion. Users would often attempt to choose a site by zooming in on the map. When they did not see any names for the sites, they began using their memory or deductive powers to determine which site was which. Although the majority eventually discovered the “Site List” drop down, one user did not; instead he would click on a site in the map and then attempt to download data in order to see the name of the site he had clicked, and by using trial and error he thereby found the site he wanted. I would suggest making
Figure A.5: Start Time > End Time
Figure A.6: Download dialog
this much smaller and not putting the “Site List” on it, as that takes away from the user’s flow of direction.

A.4.5 Aggregate Data Every box

The users were asked to get some data at 30 minute intervals. To attempt to do this, they entered 0.5 Hours into the “Aggregate Data Every” box (see Figure A.7). Now, when they hit the “Start Formatting” button, nothing happens. Nothing will happen until they either (a) put 1 into the “aggregate data every” box, or (b) click “cancel” and reopen the download dialog without clicking on “Aggregate Data Every.” I would suggest to disallow the entry of 0.5 or make the “Aggregate Data Every” box capable of dealing with 0.5 and other fractions—one user entered 0.1667 here to get 10 minute data.

Addition questions here are, what is the difference between the “Measurement Intervals” and the “Aggregate Data Every” box? If they refer to the same thing then I would suggest combining them.

A.4.6 Choosing Temperature

For the new search, the user has difficulty when choosing the Temperature from “Available Properties.” (Figure A.8) If the user chooses this, it adds many measurements which the user may not want. This requires the user to tediously go through and uncheck many boxes, resulting in dissatisfaction. One user refused to uncheck all of the boxes and gave up on the search instead.
**Download Data**

**Measurement Intervals:**
- 1-minute
- 10-minute

**Measurement Types:**
- Average

**Aggregate Data Every:**
- 0.5

<table>
<thead>
<tr>
<th>Site Name</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Altitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sheep Range</td>
<td>36.572808</td>
<td>-115.204059</td>
<td>2064.7152</td>
</tr>
<tr>
<td>Pinyon-Juniper</td>
<td>066677</td>
<td>660135</td>
<td></td>
</tr>
</tbody>
</table>

**Figure A.7: 0.5 Hours error**
Figure A.8: Adding temperature adds many checkboxes
A.4.7 Small Screen Error

The new map search page does not show up well on a small screen such as a laptop. “Available Properties” is too small and is not easily expandable to see anything other than ‘Atmosphere’ values. As seen in Figure A.9, no scroll bar appears on the right side of the screen and no scroll bar appears in the “Available Properties” box. Without scroll bars, the user is unaware that there are more choices in the “Available Properties” box or that a “Selected Properties” box exists underneath the “Available Properties” box. This was observed with screen resolutions of 1366x768 and 1440x900.

Figure A.9: From Mac Airbook 1440x900 resolution
A.4.8 Zero Records Error

If a query is specified which results in zero records (Figure A.10), then the progress box continues as if it is “retrieving data” (Figure A.11) and some users do not notice that there is a problem. They sit and wait for the data to be retrieved. “Processed 0 out of 0 records” may be interpreted to mean that the application is still working and has not yet processed or gotten the first record. It is suggested to notify the user more strongly that there are no records with a specific message rather than showing a ‘0 out of 0 records’ progress box.
Figure A.11: Later picture of download progress box
A.4.9  Gray Out of Start Time

Sometimes the “Start Time” box would become grayed out and would not allow the user to change it. Not sure what caused this problem. Pressing the “Reset” button on the search page did not fix the problem. Refreshing the entire web page would eliminate the problem.

A.4.10  Numeric Keypad Failure

Aggregate data entry does not work with numeric keypad.

A.4.11  Location Obfuscation

The map locations are clickable, but they obfuscate each other. It is very difficult to uncheck / check some of the obfuscated sites. I would suggest modifying the map in order to make this simpler. And, if it is not practical to choose sites due to occlusion (Figure A.12) and not showing names, you may want to eliminate ‘choosability’ from this map in order to make it clear to users that they are not expected to know the location of their sites on a map to use the search page.

A.4.12  Download to a File

A couple of users had a problem saving the downloaded data to a file. They copied the data, pasted it into Microsoft Excel, and then saved in in Excel. They asked for a ‘save as’ button.
A.4.13 Download Link Failure

A couple of times, when users clicked on “Click here to download your formatted data” nothing happened. I believe this may have been due to some popup blocker, but have been unable to reproduce the problem.

A.5 Conclusions

The comparative testing of the text and map versions of the NCCP search page resulted in many insights regarding the usability of the NCCP site. The search pages were so closely matched that we would need to test more users in order to achieve a strong opinion of which version is better. Though the quantitative section of this report shows a slight advantage of the text version of the search page, it is not a big advantage. In addition, the user tasks were originally created for the text version.
and therefore resulted in a smoother experience for the users with the text version. Overall, the site may be improved by addressing the problems listed in the Qualitative section of this report.

For future usability tests, I would suggest that the site be tested more often with less time consuming tasks. Improving the frequency of feedback will allow problems to be addressed continually. Fixing these problems is deemed to be beneficial. Another possible improvement is to access a larger population for participation in usability studies by using a remote testing service such as UserZoom.com.
Appendix B

Usability Testing

B.1 Introduction

This section contains information to validate our research regarding how to make a user search interface usable. Though we were creating an application to subset NetCDF files, we saw that on a higher level the task was really to create a search based UI. Our previous experience contains search UI artifacts and accompanying usability studies, so we were able to take what we learned from these previous attempts and apply it to our current effort. Here we describe the comparison of our new natural language output UI to a previous search UI done for the Nevada Climate Change Portal (NCCP). We intend to answer the following research questions (RQs) here.

- RQ1: Are users more satisfied when completing tasks?
- RQ2: Are users able to complete tasks more quickly?
- RQ3: Are users able to complete tasks more successfully?
- RQ4: Are users able to complete tasks more correctly?
The NCCP [127] has been in operation for years and is a result of a close collaboration with the climate science field. The project has been funded by the NSF EPSCoR assist in studying climate change in Nevada. In order to iteratively improve the site, we have performed accessibility and usability [65,67] testing on the site. A major focus of the site is to allow climate scientists to find and download data collected from sensors, which has been provided via a search UI (Figure B.1).

In our efforts to create a NetCDF subsetting application, we have constructed a search UI for this purpose. This search UI (Figure B.2) was created using the
Figure B.2: The search page subsetting NetCDF files

ASP.Net Web forms for the purpose of testing the UI with users before constructing the full application. It is a result of our previous work and a natural language output search UI [35].

B.3 Quantitative Analysis

Our report here contains a quantitative analysis using numerical data from our testing with users. The user population consisted of climate scientists, a geological sciences graduate student, three computer science students, and a working software engineer for a total sum of 8 users. During the quantitative analysis, we test if there is a significant difference between the climate science users and computer science users to determine if we should only include the climate scientists for that particular compar-
ison or all of the users.

B.3.1 SUS

With this analysis of the SUS, We answered RQ1. The comparison of the CS students to the climate science students / professionals illustrated that there was no significant statistical difference between these two groups. Therefore, we used all of the subjects and compared them to our previous study of the NCCP search page—the textual version of that page was chosen as it had the highest SUS scores. The results of this comparison are shown in Figure B.3. A 2-Sample t-test indicates an 87.5% chance that the mean SUS scores are different and there is a 94% chance that our natural language output search page has a higher SUS score than NCCP’s text-based search page.

B.3.2 Timing Results

In order to answer RQ2, we timed users while they were performing their tasks. We found a statistical difference between our CS and climate science users, therefore we compare the climate science subset of our population to that of the NCCP tests. Only completed tests were compared, whether they got correct results was not addressed in this timing analysis. Confidence levels of 95% were used for all of the comparisons listed here. All of the timing values are listed in seconds and were truncated. Test 1 represents that we have a 72 second improvement for the natural language output design, and this is approximately 2/3 of the time it takes a user to navigate the NCCP text-oriented search page. Tests 2 and 3 only improve the results of the natural language output against the text-oriented search page. Overall, these results
Figure B.3: SUS scores
indicate that the natural language output is a significant improvement over previous NCCP search pages.

Table B.1: Timing results

<table>
<thead>
<tr>
<th>Site</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Difference (seconds)</td>
<td>72</td>
<td>102</td>
<td>83</td>
</tr>
<tr>
<td>Difference Ratio</td>
<td>2/3</td>
<td>1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>Confidence Interval</td>
<td>8 to 137</td>
<td>21 to 183</td>
<td>36 to 130</td>
</tr>
<tr>
<td>p-value</td>
<td>0.98</td>
<td>0.97</td>
<td>0.99</td>
</tr>
<tr>
<td>N</td>
<td>9,4</td>
<td>9,4</td>
<td>10,4</td>
</tr>
</tbody>
</table>

B.3.3 Completion Rates

We used the N-1 Two Proportion Test and the Fisher Exact Test to analyze the completion rates in order to answer RQ3. All percentage values were rounded to their nearest integer equivalent representation. Table B.2 shows that although the completion was higher for the natural language output, there is not enough data to make a good determination. All measurements used a 95% confidence level. The NCCP text-oriented search page was testing against our natural language output search page and both CS and climate science users were included in the study. Per the table, we cannot conclude anything decisively, besides to say that we cannot detect a real difference in completion rates between our two user populations.
Table B.2: Completion rates

<table>
<thead>
<tr>
<th>Site</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chance the Completion rates are different</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Chance that NL has higher completion rate</td>
<td>56%</td>
<td>56%</td>
<td>N/A%</td>
</tr>
<tr>
<td>Chance that Text has higher completion rate</td>
<td>44%</td>
<td>44%</td>
<td>N/A%</td>
</tr>
</tbody>
</table>

B.3.4 Correctness Rates

By instructing users to get a specific subset of data and confirming that they got the correct data set, we were able to answer RQ4. The data compared here used a N-1 Two Proportion test with a 95% confidence interval. All of the percentages represented were rounded to their nearest whole number equivalent. We used both CS and climate science students and professionals to account for completion rates. Table B.3 displays the raw data for both versions of our search interface, while Table B.4 uses our proportional test to compare the completion rates. Overall, it appears that the natural language output search interface contributes to less errors made by users.

Table B.3: Correctness rates

<table>
<thead>
<tr>
<th>Counts</th>
<th>Text Test 1</th>
<th>NL Test 1</th>
<th>Text Test 2</th>
<th>NL Test 2</th>
<th>Text Test 3</th>
<th>NL Test 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Total</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Percentage</td>
<td>100%</td>
<td>78%</td>
<td>67%</td>
<td>100%</td>
<td>70%</td>
<td>75%</td>
</tr>
</tbody>
</table>

B.4 Qualitative Analysis

During the testing, we were observing the users and we learned of several difficulties that they had. One of these difficulties was with choosing the correct variable in the
Table B.4: Correctness comparison

<table>
<thead>
<tr>
<th></th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chance the correctness is different</td>
<td>53%</td>
<td>92%</td>
<td>18%</td>
</tr>
<tr>
<td>Chance that Text has more correctness</td>
<td>26%</td>
<td>4%</td>
<td>41%</td>
</tr>
<tr>
<td>Chance that NL has more correctness</td>
<td>74%</td>
<td>96%</td>
<td>59%</td>
</tr>
</tbody>
</table>

hierarchical organization of drop down lists. The problem was that the words and groupings we used sometimes conflicted with the user’s expectation of how such a variable would be named and grouped. Another problem was that one of the drop down lists was inconsistent in that if there were multiple choices it defaulted to the first one while all of the other drop down lists defaulted to a blank if there were multiple choices. We also asked users for comments about their experience using the search UI and they mentioned the problems previously described as well as the absence of labels for some of the drop down lists. Though all of these problems occurred, none was a significant problem during testing and due to the prototypical nature of this search UI, we are not going to address any of these problems until we create the next version of our search UI.

B.5 Conclusions

The text-oriented and natural language output versions of our search UI have been compared and contrasted in order to answer a number of research questions. We are 94% confident that users were more satisfied using the natural language output version thereby answering RQ1. The natural language output UI also was navigated up to twice as fast as the text-oriented version which answered RQ2. The completion
rates were similar and therefore the natural language output UI did not show any advantage in users completing their tasks (RQ3). The answer to RQ4 was that there is no significant differences between the two UIs in regard to completing tasks correctly. Overall, this report definitively answered our research questions using a prototype created in a short amount of time and therefore we rate this usability study as a successful one in moving our design forward.
Appendix C

User Interfaces to Enable Searches

C.1 Introduction

This appendix has been included to provide some details on the depth of UX knowledge in the area of searching information. Though many people are familiar with performing web searches on the internet, few understand how much research has been done regarding such interfaces. UI-based information search is both paramount to the NetCDF application that we have created and a significant component for many UI designs.

C.2 Details

In any small storage system, the human brain can be used to remember where things are and thereby retrieve them quickly. As the storage grows, it outgrows the ability for a human mind to keep track of things, and therefore we create systems to track items. Library card catalogs provide an indexing system to reference the location of
large numbers of books in a library, a phone book lists names and numbers, and a product number system can keep track of items in a store. Electronic storage systems are now widespread due to technologic advances and people have begun to study the different search methods available to electronic storage systems.

Electronic searches have progressed with technology, starting from rigorously specifying search criteria to ensure that precious computer time is used efficiently, to today where searches are fast and interactive. Browsing interfaces and the closely associated directory systems have also been used. Text based search has become common and can be very powerful as seen through Google’s success. A Boolean search enables users to sophisticatedly describe the information that they want to search for. Another area in which electronic searches have progressed is that of improving the data by adding metadata or formalizing the terms used. Even image searching is now not only being done with metadata added to images, but also by Content-Based Image Retrieval (CBIR) which uses computer vision techniques.

The area of Search User Interfaces (SUI) encompasses many areas of expertise (Figure C.1). To emphasize the depth of the subject, the author of one book [196] states:

UX practitioners, however, are experienced in refining systems in general, to achieve a good user experience, but they may not have search-domain specific skills for nuanced SUI design

Types of searches include:

- Multimedia Document Searches
  - image search
  - map search
Figure C.1: Disciplines for Search User Interfaces [196]

Figure C.2: Stages of user search
– design or diagram search
– sound search
– video search
– animation search

• Advanced Search Interfaces

– filtering with complex Boolean queries
– automatic filtering
– dynamic queries
– faceted metadata search
– query by example
– implicit search
– collaborative filtering
– multilingual searches
– visual field specification
Abstract Syntax Tree Metamodel (ASTM)

An OMG standard to support a direct 1 to 1 mapping of source code statements into lower level software models. It is a complementary specification with KDM.

Agile MDA

An MDA approach that forgoes the creation of an entire analysis or design model and instead uses modeling to incrementally create an implementation by making models executable at all times.

American Standard Code for Information Interchange (ASCII)

A defined character set for 127 characters of the English language.

Android

An operating system designed by Google for mobile devices such as a phone or tablet computer. It is a Linux based open source application.

ArgoUML

An open source modeling IDE tool that uses UML and generates Java, C++, and other coding languages (http://argouml.tigris.org).
Artificial Intelligence (AI)
A method which allows computers to learn from experience. 45

Aspect Oriented Programming (AOP)
A programming method for separating cross-cutting concerns using weaving compilers. 30

Backus-Naur form (BNF)
A metalanguage to define context-free grammars (http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form). 157

Beta
Specifies a stage of the software development lifecycle where the software meets its functional requirements but still has inconvenient bugs, performance, and other nonfunctional deficiencies. Users are made aware of this and are asked to provide feedback. 141

binary file
A file that stores data in bytes, but does not follow any specific file standard for these bytes, such as the ASCII standard. 73

Business Process Model and Notation (BPMN)
A graphical notation to support the representation of business processes. 31

Cascading Style Sheets (CSS)
A language to specify presentation semantics for documents written in some markup language, such as HTML. 22
Comma-Separated Values (csv)
A file format that stores data in table form where a comma demarcates columns and a carriage return demarcates rows. 141

Common Data Format (CDF)
A self-describing data format for storing multi-dimensional data in a platform-independent and discipline-independent way cdf.gsfc.nasa.gov. 146

Common Data Model (CDM)
A data model from Unidata that merges the NetCDF, Open-source Project for a Network Data Access Protocol (OPeNDAP), and HDF5 data models (http://www.unidata.ucar.edu/software/netcdf-java/CDM/). 86

Common Warehouse Metamodel (CWM)
An OMG standard to model elements in a data warehousing environment such as relational, multi-dimensional, and other objects. 31

Complete Meta Object Facility (CMOF)
A model for defining the architecture of a modeling language. 29, 30

Computer-Aided Software Engineering (CASE)
A CASE tool provides assistance in any stage of the software engineering process. Tools such as data flow diagram editors, code generators, code completion, and IDEs are some examples of CASE tools. 25

Concrete User Interface (CUI)
A UI implementation. 115

Content-Based Image Retrieval (CBIR)
An image search which uses computer vision in order to match an image with those in some storage location. 192

Context Aware Modeling for Enabling and Leveraging Effective interactiON (CAMELEON)

A project whose goal is to support the creation of user interfaces which can change slightly to adapt to a different context while supporting usability concerns (http://giove.isti.cnr.it/projects/cameleon.html). 114

Desert Research Institute (DRI)

A environmental research organization of the Nevada System of Higher Education (NSHE) (http://www.dri.edu/). 160

Domain Driven Design (DDD)

A software development methodology which first creates a domain model and subsequent stages are derived from this model. 8

Domain Specific Language (DSL)

A computer language that is targeted to a specific domain and is limited in what it can express. 2, 3, 19, 21–23, 28, 32, 58–63, 71, 83, 85, 89, 96, 101, 106, 108, 121, 122, 124, 126, 127, 130, 131, 135, 153, 155–157, 203, 212

don’t repeat yourself (DRY)

A principle of software engineering which states that ‘Every piece of knowledge must have a single, unambiguous, authoritative representation within a system’ (http://en.wikipedia.org/wiki/Dont_repeat_yourself). 2

Eclipse

An open source IDE. 83, 96, 97, 107, 108, 151, 212
Eclipse Modeling Framework (EMF)

A modeling framework that supports code generation for the intent of making tools to allow for manipulation of the model. 32, 83, 199, 201

Ecore

A meta model architecture to describe models (similar to the UML meta model), support persistence, support change notification, and manipulate EMF objects. 30, 83, 96, 108

Enterprise Distributed Object Computing (EDOC)

A UML profile to support enterprise computing, distributed computing, and Service-Oriented Architectures. 31

Essential Meta Object Facility (EMOF)

A model for defining the architecture of a modeling language. 29, 30

Executable UML (xtUML)

A design methodology which starts with expressing the program as a model and usually includes the generation of code from the created model. 19

Experimental Program to Stimulate Competitive Research (EPSCoR)

A program funded by the NSF to further competitive research in 27 USA states. 160, 183

eXtensible Application Markup Language (XAML)

A Microsoft XML language to declaratively describe objects and events. 22, 122

EXtensible Markup Language (XML)
A machine readable markup language designed for use in documents. Also used as a data serialization format. 120, 122, 199, 206, 211

**EXtensible Stylesheet Language (XSL)**

A family of languages to transform XML documents into other representations. 120

**External Data Representation (XDR)**

A serialization format to enable the representation of information on different computing platforms. 206

**Feature Driven Development (FDD)**

An Agile software development methodology which was introduced with the Java Modeling in Color UML book. 8

**File Array Notation (FAN)**

A program to extract and manipulate data from files of NetCDF format ([http://www.unidata.ucar.edu/software/netcdf/fan_utils.html](http://www.unidata.ucar.edu/software/netcdf/fan_utils.html)). 74

**File Interpolation, Manipulation, and EXtraction (Fimex)**

A program to extract, manipulate, and interpolate files of NetCDF format ([https://wiki.met.no/fimex/start](https://wiki.met.no/fimex/start)). 74

**File Transfer Protocol (FTP)**

A protocol used to transfer files over the internet. 148

**Game Maker Language (GML)**

A scripting language to enable computer games to be created more easily. 117, 118
GOMS

Stands for (Goals, Operators, Methods, and Selection rules). This is a measurement technique for HCI (http://en.wikipedia.org/wiki/GOMS). 104

grammar

The collection of rules used to describe the syntax of a language. 62

Graphical Editing Framework (GEF)

Provides software to build graphical editors and views for the EMF. 32, 83, 201

Graphical Modeling Project (GMP)

A set of tools to make graphical editors for EMF and GEF. 83

Graphical User Interface (GUI)

An interface to a computing system that uses graphical icons. 2, 8, 35, 36, 43, 60, 63, 97, 102, 120, 122, 124–127, 130, 156, 211

Graphviz

A software package that supports visualization of graphs. It is open-source. 21, 22

GRidded Binary (GRIB)

A meteorological data format for storing weather data. 146, 157

Hierarchical Data Format (HDF)

A collection of file formats to help store numerical data along with libraries to allow the use of the format on multiple platforms. These formats are self-describing. 201
Hierarchical Data Format - Earth Observing System (HDF-EOS)

A file type that adds certain conventions and data types to Hierarchical Data Format (HDF). 146

Hierarchical Data Format 4 (HDF4)

A HDF file format (see HDF5). 146

Hierarchical Data Format 5 (HDF5)

A HDF file format that includes improvements from the HDF4 format including an improved type system, access via Portable Operating System Interface for UNIX (POSIX) syntax, and some simplifications. 146, 197, 201

Human-Computer Interaction (HCI)

A field which studies how to design an interaction between people and computers. 39, 44–46, 57, 200

Integrated Development Environment (IDE)

A software application to assist in developing software by incorporating multiple tools together, such as an editor, compiler, GUI builder, etc. 2, 28, 119, 123, 135, 195, 198, 203

Integrated-CASE (I-CASE)

A CASE system that provides prototyping, modeling, and logic diagrams to support the generation of code from those specifications. 25

Interaction Design (IxD)

Design of the interaction between humans and a digital product. 2, 39, 45, 63, 75
Interactive Data Language (IDL)
A computer language that is used to create visualizations from data. 146, 148

iOS
An Apple Computers operating system designed for mobile devices such as their iPhone or iPad. 118

IRI/LDEO Climate Data Library
A program to manipulate large datasets and model I/O. It allows one to read, write, combine, or filter NetCDF files (http://iridl.ldeo.columbia.edu/). 74

Ivan Shmakov’s NetCDF Tools
Software tools to extract and manipulate data from files of NetCDF format (http://freecode.com/projects/netcdf-tools). 74

Jackson System Development (JSD)
A software development method created by Michael Jackson that was for the development of entire systems and not just one individual program (http://www.jacksonworkbench.co.uk/stevefergspages/jsp_and_jsd/index.html). 17

Java
An object-oriented programming language. 97, 108

JetBrains Meta Programming System (MPS)
An IDE that allows for the creation of a DSL and associated editor for that DSL. 32
Joint Ecosystem Modeling Group (JEM)

A group that was established to address the communication divide between ecological model developers and ecological model users (www.jem.gov). 141

Knowledge Discovery Metamodel (KDM)

An OMG standard to represent software artifacts and the environments that they operate in. 31

Linden Scripting Language (LSL)

A scripting language for use in the virtual game world of Second Life®. 118, 119

Lisp

A programming language. 122

Make

A build utility that allows developers to declaratively specify how to compile, test, package, and deploy a specific computer program. 22

MATLAB

A high-level language that enables visualization and numerical manipulations. (www.mathworks.com/products/matlab/). 140

Meta Object Facility (MOF)

A model for defining the architecture of a modeling language. 29, 31

MetaEdit+

A program that enables the user to model an application without writing code (http://www.metacase.com/mwb/). 102, 104, 110
Model Driven Architecture (MDA)

A MDE methodology, defined by the Object Modeling Group. 15, 19, 20, 23–26, 30, 32, 59

Model-Driven Development (MDD)

A term that describes any of a number of software methodologies that are based on abstract models. 15, 19

Model-Driven Engineering (MDE)

A design methodology which starts with expressing the program as a model and usually includes the generation of code from the created model. 3, 15, 16, 18, 19, 21, 23–32, 34, 47, 59, 73, 74, 78, 83, 114, 115, 155

Model-Driven Software Development (MDSD)

A term that describes a software methodologies where a model and code are kept consistent. 15, 19

Model-View-Controller (MVC)

An architecture pattern which describes the responsibilities and interactions of a model, view, and controller components which are used to abstract the presentation from the underlying model. 130, 131, 208, 210

Modeling Workflow Engine 2 (MWE2)

A code generator engine. 62, 96, 108

MOF Model to Text Transformation Language (MOFM2T)

A language to allow a person to write mapping rules to transform a PSM to a textual representation, namely that of a particular programming language such as Java. 21
National Science Foundation (NSF)
A government organization of the U.S. which provides funding for basic science research and education. 72, 160, 183, 199

NetCDF add-in for Microsoft Excel
An add-in for Microsoft Excel to provide for the manipulation of NetCDF files (http://code.google.com/p/netcdf4excel/). 74

NetCDF Markup Language (NcML)
An XML format for NetCDF metadata. 139

NETwork Common Data Form (NetCDF)
A data model, data file format (see External Data Representation (XDR)), and a set of libraries used to hold array-oriented data for science applications. It is meant to enable interoperability among different scientific applications. 73–75, 77, 78, 80–82, 85, 86, 89, 91, 92, 97, 98, 100, 102, 110, 133–135, 139–141, 144, 148, 151, 152, 155, 157, 182, 183, 191, 197, 200, 203, 205, 206, 211

Network Common Data Form 64-bit offset (NetCDF 64)
Refers to the NetCDF file format for versions after 3.6.0, but before version 4.0. 74

Network Common Data Form Classic (NetCDF Classic)
Refers to the NetCDF file format for versions before version 3.6.0. 73, 74

network Common Data form Language (CDL)
A language used to describe NetCDF datasets (www.unidata.ucar.edu/software/netcdf/docs/netcdf/Data-Model.html). 139
Network Common Data Form version 3 (NetCDF-3)

Refers to the NetCDF data model for versions 3.x. 146

Network Common Data Form version 4 (NetCDF-4)

Refers to the NetCDF data model for versions 4.x. 146

Nevada System of Higher Education (NSHE)

An organization created to supervise higher-educational Nevada institutions which receive support from the state (http://system.nevada.edu/Nshe/). 198

Object Constraint Language (OCL)

A language used to express constraints on UML models. 31, 83

Object Management Group (OMG)

A consortium which develops enterprise integration standards. Some of their current standards are UML, MDA, SysML, and others. (http://www.omg.org). 19, 21, 24, 29–31

Object Oriented Analysis (OOA)

A stage of object oriented development which translates the problem into a solution that addresses the functional requirements and sometimes other requirements. (Lahman 2011). 17, 19, 32

Object Oriented Design (OOD)

A stage in Object Orientation development that comes after OOA and addresses the non-functional requirements. (Lahman 2011). 17, 19, 32
**Object Oriented Programming (OOP)**

A programming methodology based on combining data and behavior into components called classes. 17, 123

**ontology**

A vocabulary and taxonomy for a specific domain with associated relationships among objects in that domain. This structure for data gives the ability to support reasoning. 72

**Open-source Project for a Network Data Access Protocol (OPeNDAP)**

A framework for research scientists to aid in networking scientific data (http://www.opendap.org/). 197

**pattern language**

A language which applies to a complex activity other than communication which forms a basis for describing that activity. The vocabulary consists of a number of design patterns, the syntax illustrates how something fits into the larger design, and the grammar describes how to construct phrases in the language. 117

**Platform Independent Model (PIM)**

A model that does not depend on any specific platform. 19, 20, 59

**Platform Specific Model (PSM)**

A model that represents an implementation for a specific platform. 19–21, 24, 59

**Portable Operating System Interface for UNIX (POSIX)**
A set of Institute of Electrical and Electronics Engineers (IEEE) standards for ensuring compatibility across different UNIX systems. 202

**Programmable Interrupt Controller (PIC)**

A microchip that receives and handles interrupts. 8

**Query/View/Transformation (QVT)**

A number of languages that are intended to allow a person to write mapping rules to transform a PIM to a PSM. 20

**Search User Interfaces (SUI)**

A UI created to perform searches of data. 192

**Semantics of a Foundational Subset for Executable UML Models (FUML)**

A derivative of UML version 2.1 intended to simplify UML and increase the ability to create executable models. 31

**Separation-Based Architecture Pattern (SBAP)**

A software architecture that supports usability by isolating UI concerns to a UI component such as the View in MVC. 124, 210

**Software Engineering (SE)**

An engineering discipline concerned with the medium of software. 48, 57, 98, 114

**Software Requirements Specification (SRS)**

A requirements document containing the user requirements, both functional and non-functional, and the system requirements. 61
System Usability Scale (SUS)
A form used to assess the subjective views of users during a usability test. 164, 185

taxonomy
A classification of items which shows the relationship among those items, such as a hierarchical arrangement of biological organisms. 72

Technology Acceptance Model (TAM)
A model that attempts to explain the way in which users come to accept the use of a particular technology. 44

Third-Generation Language (3GL)
A language that is above assembly language. 15, 16, 59

UML Profile
A mechanism to refine the semantics of UML models for a specific domain or platform. 28

UML Profile for Enterprise Application Integration (EAI)
An effort by the OMG to define and publish a metadata standard to allow the interchange of information regarding application interfaces. The focus is to allow enterprise applications to work with each other and with legacy applications. 31

Unidata
A community of many institutions that provides tools and specifications for achieving data interoperability. 72, 73, 86, 197
Unified Modeling Language (UML)
A widely used software modeling language used for object oriented designs. 13, 20, 24–29, 31, 32, 59, 83, 89, 97, 195, 199, 206

United States Geological Survey (USGS)
An agency that studies the landscape, natural resources, and natural hazards of the United States of America. 141

Usability Professionals’ Association (UPA)
An organization for usability professionals (www.upassoc.org). 37

Usability-Supporting Architecture Pattern (USAP)
A software architecture that supports usability which is not otherwise supported by the SBAP such as MVC. 117, 130

User Experience (UX)

User Interface (UI)

User Interface eXtended Markup Language (UsiXML)
A UI markup language for generating multiple user interfaces in different con-
texts such as Character user interfaces, GUIs, Auditory user interfaces, etc. (www.usixml.org). 115, 120, 130

vocabulary
A set of words. 72

WebWinds
A analysis and visualization software package for NetCDF files (http://www.openchannelsoftware.com/projects/WebWinds/). 74

Workflow Management Facility (WfMF)
A standard to define business workflow management systems in order to make them interoperable. 31

XML Metadata Interchange (XMI)
A language that can be used to transform a MOF model into another type of MOF model. 20, 29, 31

XML User Interface Language (XUL)
An XML language to declaratively create a GUI; created by the Mozilla Project. 22, 122

Xpand
A DSL to write templates for implementing code generation, especially with the Eclipse platform. 96, 108

Xtend
A Java dialect which provides operator overloading, lambdas, and other non-Java language features. 62
Xtext

A DSL and programming language creation framework which is offered for the Eclipse platform. 62, 83, 96, 97, 107, 108, 135, 151
Acronyms

API
Application Programming Interface. 23, 155

HTML
HyperText Markup Language. 9

HTML5
HyperText Markup Language (version 5). 118

HTTP
HyperText Transfer Protocol. 146

IRB
Institutional Review Board. 134, 160

LOC
Lines of Code. 135

NCCP
Nevada Climate Change Portal. 83, 134, 152, 160, 182, 183, 185, 187
SQL
Structured Query Language. 9, 21

W3C
World Wide Web Consortium. 29

Wintel
WINdows InTEL. 156
Bibliography


