Context in interactive mathematical documents : personalizing mathematics

Context in Interactive Mathematical Documents

Personalizing Mathematics

PROEFONTWERP

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op donderdag 3 december 2015 om 16:00 uur

door

Rikko Verrijzer

geboren te Alkmaar

De documentatie van het proefontwerp is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr.ir. B. Koren
1e promotor:	prof.dr. A.M. Cohen
copromotor:	dr. F.G.M.T. Cuypers
leden:	prof.dr. M. Kohlhase (Jacobs University)
	prof.dr. J.H. Davenport (University of Bath)
	prof.dr. P.M.E. De Bra
	prof.dr. J.H. Geuvers (Radboud Universiteit)
	prof.dr. J.T. Jeuring (Universiteit Utrecht)

Het ontwerp dat in dit proefontwerp wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Context in Interactive Mathematical Documents

Personalizing Mathematics

Rikko Verrijzer

Printed by Dereumaux.
Cover design by Cynthia Schoorl.

NWO
Netherlands Organisation for Scientific Research

This work is part of the research project "Context in Interactive Mathematical Documents" (631.002.058) which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

A catalogue record is available from the Eindhoven Univeristy Library.
ISBN: 978-90-386-3975-8

*Stubborn people get themselves in a lot of trouble,
but they also get things done.*

— Anna Paquin

SUMMARY

Today's current information technology opens up new perspectives and possibilities for documents on the web. They need no longer be static. We can read and learn from content presented on the web in an interactive way. This offers new possibilities for mathematical documents. For instance, the opportunity arises to keep track of a framework for the mathematics that a user is working on in a mathematical document. In our thesis, we develop a new document model for mathematical documents in which the content is enriched with a context for the mathematics displayed.

The mathematical context of the document is captured by its sets of theories, symbols, and variables. All three are partially ordered by their dependencies and related through order preserving mappings. This enables soundness checks of the document. The model is made up of three parts, concerned with the mathematics, the user, and the presentation, respectively. These parts interact with each other to update the mathematical framework, to cater for user data, and to achieve adaptivity.

A context as described by this model was implemented as an extension to MathDox, a dynamic system for presenting mathematical documents on the world wide web, developed at the Technische Universiteit Eindhoven in the beginning of this millennium. This prototype enables users to customize the document to their needs and preferences. The prototype context model contains implementations of the parts concerned with the mathematical domain, user, and presentation model. These, in turn, use a theory, symbol, and variable graph, representing the above-mentioned partial orders, as well as queries and rules on these graphs. In order to demonstrate the features of our document model, a sample document was created.

The pages that a user visits, automatically adapt to the user's knowledge and preferences by dynamically selecting and presenting content. Users select their own paths through a document and create their own mathematical context by assigning values to variables to be used in the content. The system keeps track of the knowledge of the user and ensures good presentation and the soundness of the chosen paths.

The implementation of the document model and its context are generic and multi-purpose. When creating documents, authors will be able to make a selection of context components (such as theory, symbol, and variable graphs, as well as queries and rules on these graphs) and to add their own components. The standards used for the

implementation and content include OpenMath, Phrasebooks, XSLT, XForms, Orbeon, Jelly, and Java.

The prototype demonstrates *that* and *how* mathematics can be presented considerably more interactively than by classical methods.

ACKNOWLEDGEMENTS

It seems ages ago when Mark Spanbroek asked me to apply for the position of technical developer in the European WebALT project at the TU/e. This marked the start of my involvement with the MathDox format and with it my interest in the possibilities of interactive mathematics. When the WebALT project finished Arjeh Cohen and Hans Cuypers approached me for a PhD project to enrich MathDox with a context. I eagerly accepted. I would like to express my appreciation to both Arjeh Cohen and Hans Cuypers for this opportunity and with it their show of trust. Especially since it was quite clear that with my dyslectic background, long travel hours and with a newborn at home it would be a challenge. And a challenge it has been indeed.

I also want to express my gratitude to the committee members Michael Kohlhase, James Davenport, Paul De Bra, Herman Geuvers and Johan Jeuring for agreeing to serve on my PhD committee and for providing me with much appreciated feedback.

Throughout the years many people have offered me their help and assistance. The number of many friendly pointers to spelling mistakes, incorrect sentences, incomprehensible paragraphs are simply too many to count. Among these people I especially have to thank Eric Strouwen for his seemingly limitless amount of hours he spent just to keep me writing. His proofreading of my sentences and paragraphs and his suggestions as to how to change my texts to make them say what they should, have been a tremendous help. Eric even took vacation days and traveled to Bulgaria to keep me going, twice. Finding mistakes was something Fabian Groffen was eerily good at. He found a great number of them and helped out with quite a number of LaTeX issues. His help came without being asked for and whenever I opened my laptop there was a good chance I had another one of his mails waiting. Help also come from unexpected sides; cases in example were Riny Bus and Marjolein Assenbroek, colleagues of my wife at the Dutch embassy in Sofia. During our years in Sofia, it was quite easy and tempting to let things slide and quit writing. However, they simply refused to let me do just that and kept asking for status updates. Shona Yu and Julia Collins also helped out by proofreading texts and reviewing the math. They did so while being overseas. Some last minute proofreading was done by Marja Scholten, René Jonkman and Rob Baijens, colleagues at Bertrand Russell College in Krommenie. They did not blink when they were surprised with a stack of papers and a tight deadline. Of course they got their revenge when they returned those papers full with notes. Cynthia Schoorl — another colleague at Bertrand Russell College — demonstrated her skill as an artist by creating the cover of this thesis. She was helped by Martijn Versteeg, who had way too much fun playing with Lego and building the symbols one can see on the cover. During the implementation of the context in the MathDox environment, Jan Willem Knopper assisted with word and deed whenever a MathDox-related problem would occur. He was a joy to brainstorm with over solutions that would benefit both the MathDox Player and the context implementation.

I have truly enjoyed my time in Eindhoven and that is because of all the great people I have had the pleasure to work with. I want to thank Anita Klooster for always taking care of everybody and her cheerful character. During my time, I have seen her leaving the department and returning. The Discrete Algebra and Geometry group is definitely not the same without her. I also want to thank Rianne van Lieshout, who took over from Anita in her own distinct manner and was never shy of voicing her opinion. I have always appreciated her honesty. Under the supervision of Bram van Asch I discovered the joy of teaching. I am grateful for that as I am still teaching today. I also want to thank Karin Poels, Mark Spanbroek and Dorina Jibetan, with whom I worked alongside in the LeActiveMath and WebALT projects. And of course I want to thank the many PhD students from the Discrete Algebra and Geometry and Coding and Cryptology groups: Maxim Hendriks, Cristiane Peters, Dan Roozemond, Erik Postma, Ellen Jochemsz, Reza Rezaeian Farashahi, José Villegas, Tim Mussche, Jos in ’t panhuis and Andrey Sidorenko.

Last but certainly not least, I want to thank Janette, Joa and Natalja. I am painfully aware that, far more often than I should have, I took my burden into my family. I am thankful for their understanding, support and love and deeply admire their patience.

Thank you all.

CONTENTS

1 INTRODUCTION 1
- 1.1 MathDox 2
- 1.2 Web 2.0 and the Semantic Web 3
- 1.3 Context 4
- 1.4 Adaptive applications 5
- 1.5 The objective 6

2 MATHDOX AND OTHER INTERACTIVE MATH SYSTEMS 7
- 2.1 Requirements for interactive mathematical documents 8
- 2.2 Formats for interactive mathematical documents and its use 11
  - 2.2.1 Well-known formats 11
  - 2.2.2 TeXmacs 12
  - 2.2.3 Computer algebra systems 14
  - 2.2.4 Specialized mathematical formats 16
- 2.3 The MathDox format 18
  - 2.3.1 Requirements for interactive mathematical documents in MathDox 20
  - 2.3.2 The structure of MathDox documents 24
  - 2.3.3 Semantic encoding of Math 25
  - 2.3.4 User driven interactivity 27
  - 2.3.5 Programming and scripting 28
  - 2.3.6 External services 29
  - 2.3.7 Separating functionality in MathDox files 31
- 2.4 MathDox software 31
  - 2.4.1 The MathDox Player 32
  - 2.4.2 Extending the MathDox format 35
  - 2.4.3 The MathDox Formula Editor 36
- 2.5 MathDox at work 37
  - 2.5.1 MathDox examples 37
  - 2.5.2 Exercise graphs 40
  - 2.5.3 Authoring 42
  - 2.5.4 MathDox at work in various projects 43

3 CONTEXT 47
- 3.1 Adaptive systems 48
  - 3.1.1 Scalar model 50
  - 3.1.2 Stereotype model 50
  - 3.1.3 Structural model 50
  - 3.1.4 Overlay model 50
  - 3.1.5 Bug model 52
  - 3.1.6 Plan model 53
- 3.2 Adaptive applications, an overview 53
  - 3.2.1 Categories of adaptive systems 53

3.2.2 Examples of adaptive systems 54
3.3 Context of a user 58
3.4 Requirements for an adaptive mathematical system . . 61
3.5 The need for context in MathDox 63
4 A MATHEMATICAL CONTEXT MODEL 69
4.1 Domain model 71
4.1.1 The theory graph 73
4.1.2 The symbol graph 76
4.1.3 The variable graph 78
4.1.4 Synthesis 80
4.1.5 An example 85
4.2 User model 88
4.2.1 Mathematical context 88
4.2.2 Logistic context 89
4.2.3 Knowledge context 90
4.3 Presentation model 91
4.3.1 Communication with the user and domain models 92
4.3.2 Selecting and structuring of content 93
4.3.3 Presentation of content 95
5 IMPLEMENTATION OF CONTEXT IN MATHDOX 97
5.1 Implementation design 98
5.1.1 The context object 98
5.1.2 Jelly implementation 100
5.1.3 Alterations in the MathDox Player 103
5.2 Domain model 104
5.2.1 Theory graph 108
5.2.2 Symbol graph 108
5.2.3 Variable graph 110
5.2.4 Content 117
5.3 User model 123
5.4 Presentation model 134
5.4.1 Queries and rules 135
5.4.2 Pre-defined set queries and rules 140
5.4.3 The knots example 148
5.4.4 The IDA example 151
6 CONCLUSION 167
6.1 Meeting the requirements of a mathematical adaptive system 167
6.2 User evaluation 170
6.3 Further research 171

i APPENDIX 173
A MANUAL 175
A.1 How to set up a new document 175
A.2 Tutorial 178
A.3 The layout of a MathDox Context document 188

A.3.1 File structure 189
A.3.2 Document file 189
A.3.3 Database structure 190
A.3.4 Author & inspection tools 193
A.4 Domain model graphs 196
A.4.1 Properties for nodes or edges 196
A.4.2 Defining nodes 197
A.4.3 Edges in a graph 199
A.4.4 Variable graph properties 200
A.4.5 Graph preparation script 202
A.5 Queries & rules 202
A.5.1 Custom tags 203
A.5.2 Creating new queries and rules 216
A.6 People who have worked on MathDox 219
Bibliography 221
Curriculum Vitea 233

LIST OF FIGURES

Figure 1 The DocBook example rendered 25
Figure 2 MathDox Player overview 32
Figure 3 MathDox translation pipeline 33
Figure 4 Dynamic explanation of a tangent line in Dutch 38
Figure 5 A MathDox SCORM package in Moodle 40
Figure 6 MathDox exercise graph 42
Figure 7 The flow of knowledge/information from an author, through a mathematical adaptive system, to the reader of the document. 59
Figure 8 A drawing by Escher showing mutual dependency. 72
Figure 9 Omitting redundant edges leads to a Hasse diagram. 73
Figure 10 A theory graph with nodal pages 74
Figure 11 A symbol graph on group theory. 77
Figure 12 A variable graph related to orbits. 78
Figure 13 Inter graph relations. 82
Figure 14 Used symbols and variables on nodal pages . 83
Figure 15 Running example usage 85
Figure 16 Theory graph (on the left) and symbol graph (on the right) relations Σ_S and Θ 87
Figure 17 Symbols introduced at theories ($\theta(s)$) 88
Figure 18 The schema of the presentation model. 92
Figure 19 The MathDox Player and the context add-on. . 98
Figure 20 The MathDox interpretation schema. 101
Figure 21 Output of a MathDox page with Jelly statements 102
Figure 22 A closer look at the Jelly phase and context influence on a page 103
Figure 23 A visual representation of the theory graph. . 108
Figure 24 A component of a variable graph related to orbits 112
Figure 25 A variable graph related to orbits with conditions 115
Figure 26 The MathDox Player translation pipeline adapted to cope with fragments. 120
Figure 27 A component of a variable graph related to orbits 129
Figure 28 The steps of the algorithm for a partial copy of the variable graphs 130
Figure 29 Output of code in Listing 28 142
Figure 30 The theory graph as used by the knots document 148
Figure 31 A view of a nodal page of the knot example . 149
Figure 32 The theory graph as used in IDA 152
Figure 33 The symbol graph as used in IDA 153

Figure 34 The variable graph as used in IDA 154
Figure 35 Logistic context of a user 158
Figure 36 Mathematical context of a user 159
Figure 37 Mathematical context of a user 159
Figure 38 An example with variables from the mathematical context. 160
Figure 39 An example with automatic adapted variables from the mathematical context. 160
Figure 40 A Euclid example performed by recursive fragments. 162
Figure 41 Denied access because not all required knowledge is understood. 162
Figure 42 Content made available by means of stretchtext. 163
Figure 43 A list of symbols as they occur on the theory page. 164
Figure 44 Theory nodes that fail soundness checks. . . . 165
Figure 45 The MathDox Player status page 176
Figure 46 The division nodal page 183
Figure 47 The create user tool 188
Figure 48 A theory graph with unsound nodes 194
Figure 49 The main form of a domain model graph . . . 197

LIST OF TABLES

Table 1 How well does LaTeX and TeX meet the requirements . . . 12
Table 2 How well does TeXmacs meet the requirements 13
Table 3 How well does (X)HTML and MathML meet the requirements . . . 14
Table 4 How well does WebMathematica meet the requirements? . . . 15
Table 5 How well does Sage meet the requirements? . 16
Table 6 How well does OMDoc meet the requirements 17
Table 7 How well does ActiveMath meet the requirements? . . . 18
Table 8 Requirements as met by the different formats . 19
Table 9 How well does MathDox meet the requirements 23

LISTINGS

Listing 1 An example of DocBook code 24
Listing 2 An example of OpenMath code 26
Listing 3 An example of XForms code 27
Listing 4 A Jelly Fibonacci example 29
Listing 5 Output of the Jelly Fibonaci example 29
Listing 6 An example of a MONET query 30
Listing 7 A call to a Jelly custom tag 35
Listing 8 A code example of a Jelly custom tag 36
Listing 9 An XPath expression that verifies that no addition is used. 39
Listing 10 A MathDox code example containing Jelly . . 101
Listing 11 A MathDox code example with executed Jelly 101
Listing 12 A library definition for custom tags 102
Listing 13 An example of a document file 105
Listing 14 A theory graph description file 106
Listing 15 Listing of a symbol graph 109
Listing 16 A partial variable graph 110
Listing 17 MathDox code calling a fragment 121
Listing 18 The fragment that is called 121
Listing 19 The include fragment tag 122
Listing 20 The include fragment tag 123
Listing 21 Mathematical context as stored in the user model125
Listing 22 A piece of the knowledge information 131
Listing 23 An example of knowledge information query tags 133
Listing 24 An example of a rule implemented in MathDox code 136
Listing 25 A Jelly rule being called 137
Listing 26 An example of a rule implemented by means of a Jelly custom tag 137
Listing 27 An example of a fragment implementing a rule 138
Listing 28 The use of mastered available and notReachable queries. 141
Listing 29 A retrieval of the variable from the logistical context 143
Listing 30 A fragment call a step in the Euclidean algorithm. 144
Listing 31 An example of a stretch text. 145
Listing 32 Calling the styling rule for a multiple choice exercise. 146
Listing 33 Adding knowledge events to the knowledge context. 147

Listing 34 Retrieving the set of nodes leading towards the goal node. 147
Listing 35 An example of a template as constructed with fragments 149
Listing 36 Contents of the template fragment rule 150
Listing 37 A node declaration in a theory graph 151
Listing 38 A node declaration in a symbol graph 153
Listing 39 A node from the variable graph 154
Listing 40 A document file. 156
Listing 41 Knowledge events stored in the database . . . 157
Listing 42 A fragment step in the Euclidean algorithm. . 161
Listing 43 The layout of a theory graph 179
Listing 44 The properties of a theory graph 180
Listing 45 The nodes of a theory graph 180
Listing 46 The edges of a theory graph 180
Listing 47 Nodal page group 182
Listing 48 Fragment code intro 183
Listing 49 The contents of the document.xml file 186
Listing 50 The contextvariables.xml 188
Listing 51 A knowledge XML document in eXist 190
Listing 52 An example of a variable XML document in eXist192
Listing 53 The start of a theory graph file 197
Listing 54 Definition of graph nodes 197
Listing 55 Definition of theory graph edge 199
Listing 56 An example of the OMExpression property . . 200
Listing 57 A conditions example 201
Listing 58 A fragment call with arguments set in the body 216
Listing 59 A fragment call with arguments in a name value map 216
Listing 60 An example of a custom tag. 217
Listing 61 The MathDox Context tag library class. 218
Listing 62 A snippet from the status page code with a call to a custom Jelly tag. 219

1 INTRODUCTION

Today's current information technology, with the internet, Semantic Web, Web 2.0 and Cloud Computing, to mention just a few innovations, opens up new perspectives and possibilities for mathematical documents. We no longer have solely static documents or textbooks from which to read, understand and learn, now we also have the opportunity to read and learn from interactive content. This offers new perspectives for mathematical documents and their readers, as they too benefit from dynamic and interactive material as it helps documents to make the step from a general audience focus towards a focus on smaller audiences. With the help of the computer we can now create documents that are optimized for smaller and smaller target audiences. Proper design of content makes it easier for an author to reuse content and have that content adapted to its new environment. This results in faster and more elaborate document creation and potentially more users while still being able to adjust to the needs of the few.

It is now time for the next step, where the target audience for a document shrinks even further. Instead of making documents aimed at an entire class of students we focus on an audience of only one: an individual student. This way attention can be paid to the needs of the individual student, making it easier for this reader to comprehend the contents of the document. The step from a large audience with a general focus to a smaller audience and then to the individual, requires that the author knows increasingly more about the smaller audience's needs in order to make the document fully connect with them.

Whereas it involves dedication on the side of the author to write a document targeting a small audience, it becomes virtually impossible for an author to create documents so as to target different individuals, unless the author creates a document that automatically adapts itself towards an individual. Such documents needs to modify their contents through collection and analysis of data about the user so it knows what to adapt and change.

The result of this collection and analysis is the user context, or just context for short. The context keeps track of all important data concerning the reader of the document and is to be used to adapt and specialize a document that is specific in content for the needs of a reader. In this manner specifically generated documents for readers are created much in the same fashion as interactive websites, such as e-banking systems, a web e-mail interface or an e-learning system. In these examples, there are data about the user, that are subsequently used to alter the web page to reflect the data stored in their profiles; for example, their current account balance, number of new e-mails waiting, or the current state of a course and such. We want to create a similar adaptive environment for mathematical documents.

1.1 MATHDOX

The Discrete Algebra and Geometry group at Eindhoven University of Technology has been working on the MathDox project [146, 61, 136, 62] for some time. MathDox is a format as well as a set of software tools for creating interactive mathematical documents. Such documents can be delivered over the internet and be viewed via a web browser. MathDox enables the creation of interactive mathematical content that is accessible through the internet. One of the most popular applications of MathDox is e-learning; several showcases exist, such as Wortel TU/e [112], MathAdore [60], IDA [148], WebALT [193], TELMME [99], and the on-line knot theory course notes [50]. MathDox may also be incorporated in e-learning environments such as Moodle [158], OLAT [74], and Claroline [176]. MathDox does not limit itself to just e-learning, it aims to make interactive mathematics accessible to as many people as possible. As such it can also be used for mathematical articles and documents.

Interactive mathematical formats differ from traditional paper documents and other static formats by their capacity to employ computer algebra systems allowing for real-time computations. This provides for a user experience with mathematical concepts that would not have been possible in any traditional document. For instance, examples that work with parameters as values — for example values that are given by users — increase the variation and number of possible different occurrences significantly. Therefore users can set out to satisfy their curiosity by experimenting with the values they enter, and can see what kind of effects these have on computations.

In case of exercises, MathDox allows parameter values to be randomized, surprising the user with many different exercises based on the same template. But MathDox can go even further and take the input from a user on a given page to use it for the selection of the next MathDox page to show to the user, or confront the user with their own input values by showing for instance a page that addresses the (in)correctness of the input. Analysis of answers may lead to detection of common errors, offering the opportunity for users to find out where possible misconceptions are located in their understanding of a mathematical notion.

Other possibilities of MathDox are the inclusion of interactive objects such as graphs or applets that allow readers, for instance, to play with trigonometry in a visual way [159, 178]. MathDox also possesses a formula editor [62] allowing users to enter in a simple way complex mathematical formulae. In short, the MathDox format is a scripting language aimed at constructing web pages that serve interactive mathematics, also called interactive mathematical documents.

To enrich the MathDox format with more interactivity and adaptation we introduce the concept of context into MathDox documents. The context will enable MathDox to store data pertaining to visitors of the web pages, and act upon the stored state of these users. This makes MathDox more suitable for intelligent tutoring, as it enables students to be guided by a document that adapts to the users and offers them the opportunity to create their own mathematical context.

1.2 WEB 2.0 AND THE SEMANTIC WEB

Let us take a moment to reflect on interactive mathematical documents written in MathDox, with the addition of context, in relation to the already existing trends on the internet, namely *Web2.0* and the *Semantic Web*.

Web 2.0 is largely responsible for the recent increase of interactive sites. Despite the lack of a precise definition of Web 2.0. In general Web 2.0 is considered to be a distinct movement on the internet to personalize content and make it interactive. As such, an emphasis lies in content presentation and user interfaces, making Web 2.0 an ideal candidate for the presentation layer in cloud computing, where locally installed software is replaced by interfaces to software running on remote servers instead. Despite the fuzziness of the term Web 2.0, it is generally accepted that Web 2.0 is a term for a collection or combination of trends such as collaborating user, creating of content by users, sharing of content, and interacting.

Examples of Web 2.0 applications are web logs (blogs), social networks (such as Facebook [24], Instagram [39] and LinkedIn [53]), webmail applications (e.g. Gmail [30], Hotmail [35] and Yahoo [119]), RSS-feeds [88] and many others.

The Semantic web is older and quite different from Web 2.0. Where Web2.0 aims to personalize the internet, the Semantic web tries to give content on the internet a meaning [179]. The Semantic web wants to enable software to be able to reason about content on the internet on behalf of the user [165]. For this it is necessary for software to understand what content is and what it means. Software lacks the human notion of a context to do so properly without some extra help in the form of metadata or data with a precise meaning (semantic data). By adding metadata to content where meaning is otherwise not precisely defined, it becomes possible to define relations between various sources of content and for software to start to reason about the meaning of content.

The MathDox format uses elements of both trends, and combines them into a format that has both a meaning and interactiveness. However, the MathDox format currently lacks the sense of a user and the notion of a context, or as it is called in Web2.0, a profile.

1.3 CONTEXT

Context is a word used in many different settings. A dictionary will give the following meanings of the word context.

> *The circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood.*
> *—Oxford Dictionary of English [195]*

but also

> *The parts of something written or spoken that immediately precede and follow a word or passage and clarify its meaning.*
> *—Oxford Dictionary of English [195]*

We use *context* to denote the relevant circumstances of a user that is reading an interactive mathematical document. These circumstances will include:

- Logistic information. All *personal* information of relevance for an interactive mathematical document, such as language, profession, etc.
- Knowledge information. To adjust content in order to make it comprehensible for a reader, an interactive mathematical document needs to have an idea about the knowledge of the user. For instance the theorems or symbols that are understood.
- Mathematical context of the interactive mathematical document. This includes all visible variables and their values as used in the interactive document.

Note that context as described here fundamentally differs from the notion of context as used in Dutch high schools[1].

The notion of context we use does not describe a mathematical problem, instead it describes the state of the user. As such it is comparable with a user profile as seen in a Web2.0 application or a user model as found in an adaptive system.

1 A mathematical exercise with context on Dutch high schools is regarded as description of a situation with in which one or more mathematical problems occur that are to be solved by the students. For example, an typical exercise context would be:

> *A tree has 23 rings in white, and 35 rings in brown. Each ring indicates a year, however in the first three years no rings where created by the tree. How old is the tree?*
> *Answer:* $23 + 35 + 3 = 61$.

An advantage of having a context is the fact that a mathematical document can adjust its contents to the circumstances of the user. On the basis of references in the context, a document will decide what to include, exclude, or how to adapt any specific parts of the content. For example beginners in the field will be offered an elaborate explanation of the mathematical notions as they encounter them in the document, while more advanced readers will be shown a reduced document with only the basic definitions of new concepts. Another example would be the representation of the imaginary symbol i which is either left untouched or adapted to j if the user has a background in electronics or physics, where the symbol i is commonly used to indicate amperes. Also a context allows readers to set their own favorite values for variables occurring in running examples throughout the document, enabling readers to make a document their own.

1.4 ADAPTIVE APPLICATIONS

Enriching MathDox with a context and allowing MathDox code to use this context to adapt the content as it is presented to the user will make MathDox effectively an adaptive mathematical system. Our study in making adaptive mathematical content is not alone in its effort. Over the past years several (although no mathematical) adaptive application models have been proposed [151, 131, 161] and developed, among these AHA! [132, 133], the LAOS framework [152] and the more recent GRAPPLE [135]. These systems however do not focus on and are not suitable for mathematics as they have trouble presenting mathematics, accepting input and are missing a connection to computational engines like a computer algebra system, criteria in our view required for a mathematical interactive document. Instead they aim to be a general adaptive platform.

Panta Rhei [184, 185, 183] is an adaptive mathematical system that is being built on top of OMDoc [174] and focuses on adaption of notation in the content presented to the user. Panta Rhei can transform mathematical texts into documents using the mathematical notation preferred by the reader. To select the right notation Panta Rhei groups its users into communities of practice (CoPs [199]).

ActiveMath [182, 1] is an intelligent tutoring system aimed at teaching mathematics to its users. It focuses on finding misconceptions in the knowledge acquired by its users and aims to repair these misconceptions. For this purpose ActiveMath takes into account the current knowledge of a user, and the expected approaches of the user to solve specific problems.

Our goal is to make it possible for authors to have their mathematical content connect to their readers by letting the content adapt to the specific needs of the users. This will increase the efficiency of the reading and understanding of the texts by the user, mainly by reducing the time a user loses when trying to figure out the mismatches in knowledge between the user's and author's assumption. Panta Rhei collects individuals into groups, and loses the capability to adapt documents to each specific individual. Our study focuses on the individual within a group and creates the opportunity for content to adapt itself to that specific user.

1.5 THE OBJECTIVE

The objective of this thesis is designing a model which supports interactive mathematical documents and that adapts automatically towards the user with the help of the context.

To achieve this objective we first inspect the current existing (interactive) mathematical formats in Chapter 2. In Chapter 3 the different aspects of context are discussed as well as their influence on adaptive systems. Chapters 2 and 3 will lead up to a set of requirements a mathematical adaptive system should meet.

The requirements and the conclusions drawn in these chapters are used in Chapter 4 where we will introduce the three models *domain model*, *user model*, and *presentation model* that make up our context model. We will also pay attention to soundness so as to prevent possible undesired mistakes in adapted content. An implementation of the context is described in Chapter 5, which discusses two prototype implementations of interactive mathematical documents, one about *Knots*, and another about *Group theory*, a conversion of the first chapters of IDA [148]. We will end this thesis with conclusions in Chapter 6.

2 MATHDOX AND OTHER INTERACTIVE MATH SYSTEMS

Web pages about mathematics are hard to create and therefore often static in the sense that they do not ask the user for any information and therefore do not use such information to adapt the content. This also includes pages with non-static elements if these do not accept user input, such as movies. These pages do not invite the user to play and experiment with the math they present and do not give the user a chance to interact with the material. In this sense these web pages are little more than a traditional math book.

We see three main reasons for this. First of all, interactivity in mathematical content often requires non-trivial computations that are not easily performed without the assistance of specialized mathematical software, e.g. computer algebra systems like Mathematica [63], Maxima [69] or GAP [28], or specialized applets such as Geogebra [29] or JavaScript packages like JSXGraph [49]. Not being able to make calculations removes most if not all possibilities for interactivity, as these web pages cannot react to math input from the user.

Second, displaying mathematics properly on a web page can be challenging. There are a number of methods to render mathematics on a web page, such as MathML [65], CSS [128], or by images. Unfortunately, all of these are not intuitive to write for an author. MathML is not yet natively supported by all major browsers[1], CSS style sheets are not always accurate and uniform on different browsers, images are slow to create and cannot be easily generated on the fly. MathJax [67] offers a way out by using JavaScript to render MathML and LaTeX into suitable fonts for the browsers without MathML support.

Third, user input of mathematical expressions is difficult. It is a lot of work for users to enter complex or deeply nested formulae into a web page with the standard set of XML elements, such as XHTML, OpenMath or MathML. Without any aid –for instance of palettes with mathematical symbols– it is nearly impossible for a user to enter any complex mathematical expressions an in HTML input field.

Web pages that do contain interactive mathematics, often include applets [102, 29] or flash applications [125] as demonstrated by websites such as [129, 14]. The various applets and flash items tend to be

1 Of all major browsers Internet Explorer [41] and Chrome [10]. do not support MathML natively. MathML support for Internet Explorer is obtained by installing the MathPlayer plugin [58] in versions up to Internet Explorer 9. In Internet Explorer 11 the MathPlayer has been disabled. Chrome has a limited support for MathML. Work towards MathML support had started but was canceled due to concerns about the quality of the code. FireFox [72], Safari [90] and Opera [80] do support MathML natively, but in varying degrees of completeness.

and remain separate items on a page and do not come together with the remainder of the content nor do they form a consistent interactive mathematical document. Any expression or value occurring in these items is not used in the surrounding page or document. These items also tend to emphasize focus on graphical aspects of the mathematics involved.

To address these issues the Discrete Algebra and Geometry (DAG) research group at TU/e [15] has developed the MathDox system [61, 150, 146]. It consists of the MathDox format and a set of tools including the MathDox Player. The MathDox format is an XML based language for interactive mathematical documents. As such MathDox documents can be interpreted and transformed into web pages by the MathDox Player, software designed for this sole purpose. The resulting web pages are dynamic and interactive, support rendering of mathematics, offer easy access to mathematical services including computer algebra systems, and are equipped with a convenient mathematical input system, the MathDox Formula Editor [62]. MathDox shows its potential when demonstrating the workings of an algorithm, testing readers' skills with exercises, or explaining new concepts with on-screen step by step calculations. However, it can also be used in a more traditional way for publishing static mathematical documents on paper (by export to PDF) or on the web (by export to HTML).

The following sections will introduce the MathDox format and software and other mathematical formats. In Section 2.1, we describe what criteria we feel a mathematical interactive format should meet. We discuss other available mathematical formats in Section 2.2. In Section 2.3, we examine the different XML formats that are combined within the MathDox format. Then, in Section 2.4, we discuss how the MathDox system works. In Section 2.5, we give examples of various projects in which MathDox has been used in practice.

2.1 REQUIREMENTS FOR INTERACTIVE MATHEMATICAL DOCUMENTS

The quality of an interactive mathematical format depends upon a number of factors, listed here as a set of requirements. With the help of these requirements we are able to compare the different formats. We discuss in Section 2.2 how these requirements are met by some of the available formats [2]. In Section 2.3 we discuss how MathDox meets these requirements.

2 By "format" we mean the grammar of the source text for such documents. It may be necessary to apply programs to the source text in order to obtain presentable mathematical documents. For instance a web browser that converts HTML code into a viewable document in the browser, or a program to convert a LaTeX file into a PDF file.

The following six requirements are pertinent to interactive mathematical documents.

Interactivity.
A user needs means of interacting with the mathematical content of a document. For example, the user should be able to answer a math question or to supply data for an example and to study the effects. In short the user needs ways to change parameters and there has to be a reasoning mechanism that can act on and work with those parameters. Non-mathematical interactive web pages use reasoning techniques like JSP [43], Java Servlets [44], PHP [84], or ASP [6].

Mathematical applets, flash applications are interactive but often lack the opportunity to communicate and cooperate to form a consistent page with the other elements. Flash applications or Java applets run on the client browser and are sandboxed [3] by default, as such they lack the capability to access anything outside the applet itself. At best JavaScript [45] can retrieve values in the clients browser. But by doing so an applet is considered untrusted and is run inside a sandbox. JavaScript is only available at the client when the original web page has already been constructed and sent to the client, severely limiting the use of JavaScript for content adaption.

Usable in multiple formats.
An interactive mathematical document only benefits from interactive aspects if it is served in an electronic environment. Web browsers are very suitable for this task as they are already installed on virtually every computer and do not require extra software installations by the user. However the need may arise to view such a document on different media than a normal computer, such as a mobile phone or more traditionally on paper. It is important that such formats are supported from the same set of source files, although a different format may limit the functionality of the original document. No source code changes should be required to allow for a change of display formats; all that is needed is a different program or setting to transform the source into the desired format for presentation. In other words, content needs to be rich in semantics. A development in the field of computer science addresses the same issue. There is a desire for separation of content and presentation in general. Examples include XHTML documents with CSS style sheets [128], where applying a different CSS style sheet can dramatically change the presentation of any XHTML page.

Semantics.
Semantics in non-mathematical content provides a starting point for

3 A sandbox is a confined area in which software is run and separated from other software. A sandbox will not allow software to access anything outside the sandbox like memory, connected devices and hard disk. This will protect other processes and data on the computer from possible unstable or malicious code.

transformations into different presentations, see the previous bullet point. However the need for semantic meaning extends further than just the content. The mathematics occurring in a mathematical document requires an even higher level of semantic meaning as to enable a variety of purposes, such as calculations that are to be performed by the computer. But it also allows (parts of) mathematical expressions — for instance entered by the user — to be analyzed and reused in other expressions and calculations. The semantics also provide a starting point for transformations from the mathematics as used by the computer to mathematics as read and understood by a user.

Being extendable.
Mathematics is often regarded as a language that is used in various other fields such as physics, electronics, computer science, economics, etc. The usability of any mathematical format is greatly enhanced if the format allows for field specific adaptations that would extend the functionality of the mathematical format and make it suitable for specialized applications. This means that if a document is written for a specific target audience, the format must be able to include extra (non-standard) functionality to serve that target audience in the best possible way. Extensions may include applets, web services, or changes/extensions to the format itself.

Representation of mathematics.
Any mathematical format that aims to show mathematics to its users, requires a good representation of mathematics. Before mathematics expressions are rendered on the screen it can have various other formats less readable by users. Examples are LaTeX [171, 170], OpenMath [77], or MathML [65], which are — in unprocessed form — not the kind expressions one would expect in a mathematical text. It is paramount that any reader is only served a proper rendering of mathematics.

Availability of a dedicated computational engine.
To support interactivity in relation to mathematics it is necessary to be able to (re)compute (complex) mathematical expressions when values which they depend on change. Computer algebra system are specialized in complex mathematical computations and are a must have in interactive mathematical document.

Ease of usage.
The success of a format is mostly given by the ratio of richness of the format to the ease of usage. A format that does not offer much but is still easy to use will not gain much popularity. The same goes for a rich format that is very hard to use. The users involved here are both the authors and the readers of the documents. For instance, users should not have to install any extra programs or plugins in order to

read mathematical documents. At the same time authors should not have too much difficulty creating the mathematical documents. Creating an interactive web page is often a programming task that many authors have trouble with. Therefore an interactive mathematical format should be designed so as to make this task as easy as possible for authors. For instance with authoring tools or by using well known (sub)formats authors are familiar with.

Use of open standards.
By following existing standards, a format benefits from other tried and proven technologies. It also helps to lower the learning curve considerably as compared to a completely new format. Which again directly assists in the previously discussed *ease of usage* of the format. Also the format itself and any programs needed for compilation or translation should be open, so that every author will have direct access to all the specifics of the format used and is free to make adaptations if necessary. This will further expand the opportunities the format offers. Another benefit of open standards is the opportunity to verify inner workings. Especially in the field of mathematics but also in the academic world in general, validation of correctness is paramount.

2.2 FORMATS FOR INTERACTIVE MATHEMATICAL DOCUMENTS AND ITS USE

There are a variety of different applications and formats available that are capable of dealing with mathematics in a varying degree. They range from well-known general formats like XHTML [189, 126] and LaTeX [171, 170] to computer algebra systems like Mathematica [63] and Sage [91], to specialized formats like OMDoc [174] and ActiveMath [1], and to e-learning systems as WeBWorK [107] and EMILeAstat [21]. We have categorized these formats, selected some of each category for discussion and compared them with our set of criteria for interactive mathematical documents as given in Section 2.1.

2.2.1 *Well-known formats*

In this section we discuss well-known formats that can be used for mathematics, but which may not have been primarily designed for that purpose. As a consequence we see that they do not meet all of our requirements for interactive mathematical documents, especially where math interaction is involved. Two examples of this category are LaTeX and (X)HTML + MathML.

Interactivity	Limited
Usable in multiple formats	Usually only PDF
Separation of content and presentation	Math is non semantic
Being extendable	Yes
Representation of Mathematics	Very good
Availability of a computer algebra system	No
Ease of usage	Easy to use
Use of open standards	Yes

Table 1: How well does LaTeX and TeX meet the requirements

2.2.1.1 *LaTeX and TeX*

LaTeX [171, 170] is a markup format for typesetting of (mathematical) documents, known by most (if not all) mathematicians and very suitable for static mathematical documents. LaTeX is built upon TeX [171] and focuses upon presentation aspect of the document; the mathematical expressions in LaTeX do not hold semantic value to interpret the mathematics unambiguously. Although TeX is a Turing complete programming language, LaTeX is not built for computations. Nevertheless, LaTeX is easy to use and can create documents in postscript and PDF, which both render mathematics very well. As such the LaTeX format has been the standard for non-interactive mathematical documents for quite a while. Some interactivity is obtained by CTAN packages [12] like the Exerquiz Package for exercises, the Eforms Package for PDF forms, and the Insdljs package and dljslib Package for JavaScript support, but limits the resulting document to Acrobat Reader [123]. The gained interactivity is very limited and does not support mathematical in or output.

2.2.2 *TeXmacs*

The TeXmacs [100] format competes with TeX and LaTeX but is less known. A big difference between TeX and TeXmacs is that TeX uses a regular text editor and requires a compiler to process the TeX files, whereas TeXmacs uses a WYSISWYG editor. Documents written in TeXmacs are saved in either XML, Scheme [93] or the TeXmacs own format. These are then convertible to PDF and postscript and with converters also to TeX and HTML/MathML. TeXmacs documents may also include plugins and connections to other programs, such as computer algebra system as Maxima and Sage. TeXmacs is part of the GNU Project [32] and therefore open source.

Interactivity	Good
Usable in multiple formats	PDF, postscript, LaTeX and HTML/MathML
Separation of content and presentation	Math is non semantic
Being extendable	Yes
Representation of Mathematics	Very good
Availability of a computer algebra system	Yes
Ease of usage	Easy to use
Use of open standards	Yes

Table 2: How well does TeXmacs meet the requirements

2.2.2.1 *(X)HTML & MathML*

(X)HTML and MathML [65] are both recommendations of W3C [103], the World Wide Web Consortium. Both are therefore well documented and (X)HTML documents are easy to create and use. Enriching the XHTML with MathML takes some more effort if one is not familiar with MathML, but it is an effort that pays in respect to the rendering of the mathematics in a web browser. Despite being a W3C recommendation MathML is not yet supported by all browsers.

MathML exists in two variants: MathML-Presentation and MathML-Content. MathML-Presentation is used to render the mathematics inside users' web browsers. MathML-Content is similar in purpose to OpenMath [77] and both are meant as semantically rich formats, but both need to be converted into MathML-Presentation before they can be used by a web browser. MathML-Presentation, MathML-Content and OpenMath all get increasingly difficult to type and edit when expressions become more complex.

Recently, work has been completed on new versions of MathML (MathML 3.0 W3C[66]) and OpenMath. These formats are interchangeable with each other. As a result the differences in use are negligible.

(X)HTML and MathML are content formats and therefore any documents written in these formats will lack interactivity without the addition of server sided software. This absence limits the possibility to adapt the content sent to the user to the specific needs or demands of said user, although some code adaption is still possible with the use of JavaScript. Files containing (X)HTML and MathML can be dynamically generated by server-side applications like Java-Servlets [44] (and JSP [43]), PHP [84] or ASP [6] scripts. By dynamically generating the (X)HTML/MathML files the content can be made to adapt to the needs of the user and interactivity is introduced in this way.

Interactivity	Requires server side code
Usable in multiple formats	No
Separation of content and presentation	Yes
Being extendable	Yes
Representation of Mathematics	Good
Availability of a computer algebra system	No
Ease of usage	Well known, easy to use
Use of open standards	Yes

Table 3: How well does (X)HTML and MathML meet the requirements

2.2.3 *Computer algebra systems*

A Computer Algebra System (also called CAS) is an application specialized in (exact) mathematical calculations. It is able to receive an expression by means of an interface (i.e.. command line or web interface) and will return a response to it. Such a response may be the result of a calculation or simply the confirmation of having received the expression, which may be called upon in a following expression. Computer algebra systems are a necessity for any interactive mathematical format that needs to perform calculations, as these calculations can quickly become so complex that the — in comparison — simple computation possibilities of a normal programming language are quickly exhausted. The complexity of computations even forces computer algebra systems to specialize in specific fields. For instance GAP [28] focuses on group theory where Maxima [69] specializes in manipulation of symbolic and numerical expressions. Some computer algebra systems support a feature called notebook. A notebook is a (web) interface with predefined expressions, usually example programs that aim to show and or teach certain mathematical specifics.

Among the various computer algebra systems that exist, we discuss Mathematica [63] and Sage [91] as representatives.

2.2.3.1 *Mathematica, webMathematica*

Mathematica [63] is a well-known computer algebra system. There are multiple interfaces that connect to Mathematica, among which a GUI, a command line interface, the Wolfram Workbench (based on Eclipse [19]) and with webMathematica [106] and Wolfram Alpha [111] also a web browsers. Our interest goes to webMathematica, as webMathematica is capable of producing interactive mathematical documents on the web. It is based upon the Java Servlet framework and offers some servlet functionality to the author of a webMathematica document.

Interactivity	Possible with Notebooks
Usable in multiple formats	Very limited
Separation of content and presentation	Limited
Extendable	No
Representation of Mathematics	Very good
Availability of a computer algebra system	Yes
Ease of use	Limited
Use of open standards	No

Table 4: How well does WebMathematica meet the requirements?

WebMathematica pages are written in a similar fashion to how JSP pages are written. HTML code is enriched with Mathematica Server Page statements in order to get the desired combination of web page and mathematical interaction.

Mathematics in webMathematica can be rendered by using images or MathML. As the name suggests, webMathematica is a web based interface to the computer algebra system Mathematica.

webMathematica is not an open format and is therefore more difficult to adapt to other applications. webMathematica does however cooperate with JavaScript, applets, Active-X controls, browser plugins, and Excel.

Mathematica also provides the Mathematica Player [64]. This is an application in which the Mathematica engine is embedded and allows the user to view interactive documents. The Mathematica Player does not need prior installation of a servlet container or a Mathematica installation and can be used without a browser. The Mathematica Player is a successor to MathReader, which handled only static documents. Mathematica Player uses manipulation functions for interaction. These functions are graphical components that can be used to input or alter settings. These settings are then passed to the Mathematica Player engine. The Mathematica Player does not allow the input of mathematical expressions without the use of one of the manipulation functions.

2.2.3.2 *Sage*

Sage [91] is an open source computer algebra system that combines different mathematical open source software into one computer algebra package. It offers a command line interface as well as a web interface.

Some software packages that are included are: GAP [27], PARI/GP [82], Macaulay2 [54], Maxima [69], Octave [31], and Singular [96]. Closed software is not included, however, support for Magma [55],

Interactive documents	Possible with Notebooks
Usable in multiple formats	Very limited
Separation of content and presentation	Limited
Extendable	Yes
Representation of Mathematics	Very good
Availability of a computer algebra system	Yes
Ease of use	Easy
Use of open standards	Yes

Table 5: How well does Sage meet the requirements?

Maple [56], Mathematica [63], and MATLAB [68] is included as to allow users who have these installed on their computers to use them through Sage as well. Sage is written in Python [86] and Cython [13] and uses close to 100 open source libraries.

Sage provides Notebooks, which are mathematical documents that can be viewed by and edited in a web browser. These are written in Python and Cython. Notebooks aim at collaboration between people allowing, for example, students to work together on their homework assignments. The languages that can be used in the Notebooks are Python and LaTeX. The former is used for complex free-form calculations and plotting, whereas the latter is useful for displaying mathematical formulae. Since there are no interactive documents there is no need for separation of content and presentation.

2.2.4 *Specialized mathematical formats*

Over time, various research groups have developed their own mathematical formats. These formats differ quite a bit from each other as they have been build with different design goals. For instance while OMDoc aims to be a semantically rich format, WeBWork [107] puts more emphasis on presentation instead. We will discuss OMDoc and ActiveMath (related to OMDoc) in more detail.

2.2.4.1 *OMDoc*

The OMDoc [174] format is an open format which focuses on the semantic meaning of math in its documents. The strict distinction between semantic meaning and presentation form was one of the design goals of the format. The semantic meaning of mathematics in OMDoc is achieved by the use of OpenMath and Content MathML.

Transformation from the XML OMDoc format to another format easier to read by humans is done in two steps. First knowledge about the user is taken into account which results in in- or exclusion of ma-

Interactivity	No
Usable in multiple formats	XHTML, LaTeX, PDF
Separation of content and presentation	Yes
Extendable	Yes
Representation of Mathematics	Very good
Availability of a computer algebra system	No
Ease of use	Moderate
Use of open standards	Yes

Table 6: How well does OMDoc meet the requirements

terial. This can include matters like language in the document itself (provided the document has more than one language to offer) as well as knowledge already known by the user and therefore not needed by this user. Next a series of XSLT style sheets [118] will transform OMDoc into HTML (+MathML) or LaTeX. The transformation of the latter to PDF is trivial with pdflatex from the LaTeX suite.

The OMDoc format has no interactivity and therefore no means for entering mathematical expressions. This is however something ActiveMath adds to OMDoc. Also, changes in users' context and the lack of interactivity requires OMDoc to start new transformations to reflect presentation changes.

The modular design of OMDoc enables extending the format into other formats. OMDoc is used for e-learning by ActiveMath [1, 182, 181] although its emphasis is more on the formal, semantic and logical treatment of mathematics and less on the interactivity of mathematics. OMDoc is also extended to be used in a physics environment with the format PhysML [168].

Because of the semantical and logical structure of OMDoc it enables meaningful communication between various software systems, including mathematical web services. Software also exists for translation to theorem provers and computer algebra systems such as Coq [11] and Maxima [69].

2.2.4.2 *ActiveMath*

ActiveMath [180, 1, 182, 181, 122] is a mathematical intelligent tutoring system developed by DFKI and Saarland University. ActiveMath is built upon OMDoc and therefore requires content — which is fragmented into separate notions — to be written in OMDoc entities. An document[4] is then constructed by a compilation process that selects entities and transforms them into a document. The generated document is organized around a collection of concepts such as definitions, axioms, assertions, proofs, algorithms, etc. This is further extended

4 also called a book or a course

Interactivity	Good
Usable in multiple formats	HTML, Latex, PDF, SVG
Separation of content and presentation	Yes
Extendable	Yes
Representation of Mathematics	Very good
Availability of a computer algebra system	Yes
Ease of use	Moderate
Use of open standards	Yes

Table 7: How well does ActiveMath meet the requirements?

with additional items like explanations, elaboration and exercises. The selection and transformation process also personalizes the content for individual students on the basis of user data. This user data is gotten from questionnaires about learning goals, background and preferences of the student and is then stored in the user model.

Because of its connection to OMDoc, ActiveMath has similar characteristics, such as the semantics of content, including mathematics which is written in OpenMath. The semantic meaning also makes it possible to create a course in another format such as PDF or SVG. As ActiveMath is a tutoring system it also supports means for interactivity such as exercises, and access to computer algebra systems like Mathematica [63] and GAP [28].

This modular design together with the opportunity to link client tools and loosely coupled components and its open source license allows for extending and specializing the ActiveMath format. Ease of usage is moderate as the format used is not well known.

As intelligent tutoring systems are a subpart of adaptive systems, we will discuss the adaptive capabilities of ActiveMath in more detail in Section 3.2.2.6.

2.3 THE MATHDOX FORMAT

MathDox has been developed at the Discrete Algebra and Geometry group at the TU/e in Eindhoven. The MathDox format is a mathematical format that compares with OMDoc and ActiveMath. The MathDox format combines several XML formats. The combination of XML formats is then translated upon a user's request to form an interactive mathematical document accessible through the web. Each format contributes a useful facet for an interactive mathematical document. The XML formats used in MathDox are:

Table 8: Requirements as met by the different formats

Requirements	LaTeX	TeXmacs	MathML and (X)HTML	(Web)-Mathematica	Sage	OMDoc	ActiveMath
Interactivity	Limited	Good	Requires server code	Possible with Notebooks	Possible with Notebooks	No	Good
Usable in multiple formats	Usually only PDF	PDF, postscript, LaTeX and HTML/-MathML	No	Very limited	Very limited	XHTML, LaTeX, PDF	HTML, LaTeX, PDF, SVG
Separation of content and presentation	Math is non semantic	Math is non semantic	Yes	Limited	Limited	Yes	Yes
Being extendable	Yes	Yes	Yes	No	Yes	Yes	Yes
Representation of Mathematics	Very good	Very good	Good	Very good	Very good	Very good	Very good
Availability of a computer algebra system	No	Yes	No	Yes	Yes	No	Yes
Ease of usage	Easy to use	Easy to use	Well known easy to use	Limited	Easy	Moderate	Moderate
Use of open standards	Yes	Yes	Yes	No	Yes	Yes	Yes

- *DocBook.*
  DocBook [16] is used to structure documents. It is an open markup format that adds semantics by specifying the different roles document parts have. Enabling translation into multiple presentation formats. See also Example 1.

- *OpenMath.*
  Semantic encoding of mathematics is done with OpenMath [77]. Thanks to the semantics of OpenMath, mathematical expressions are used for computations but also for presentation.

- *XForms.*
  User-driven interactivity, and user input in general, is made possible with XForms [113]. XForms is an open standard and it replaces the combination of HTML-Forms [36] and JavaScript [45] with a new XML based and open format.

- *Jelly.*
  Jelly [47] is a programming and scripting XML format and is responsible for acting upon user input. Content depending on conditional actions, loops or external web services require Jelly code.

- *XInclude.*
  The inclusion of other pieces of content from different files enables separation of responsibilities and enlarges the possibilities for maintenance and adaptation. XInclude [117] makes this possible.

These formats will be discussed individually in the Subsections 2.3.2 to 2.3.7 below.

MathDox documents are served to users by the MathDox Player, which will be discussed in Subsection 2.4.1.

2.3.1 *Requirements for interactive mathematical documents in MathDox*

We discuss how MathDox meets the criteria of each of the design features listed in Section 2.1.

2.3.1.1 *Interactivity*

MathDox combines the interactivity as found on web pages with the power of mathematical computations, resulting in interactive mathematics. The interactivity of web pages is realized within the MathDox documents and MathDox Player by use of the scripting language

Jelly, that adds conditional logic capable of responding to user input from XForms elements [113]. The Jelly format also offers interactivity through inclusion of (specialized) web services. Interactivity finally is boosted by means of included applets on the page. The Jelly scripting language [47] also offers the opportunity to interface with external software, the most notable among these being computer algebra systems. XForms and applets react to user input without the need to refresh the web page. XForms can even hide, show and adapt existing content on a web page to certain limits. This results in web pages that adapt to user input and can contain complex mathematical expressions that are run-time computed.

2.3.1.2 *Usable in multiple formats*

Mathematical documents written in the MathDox format can be translated from the XML source into XHTML, but with a different set of XSLT style sheets also into PDF and LaTeX. By creation of yet more XSLT style sheets other formats also become possible. These transformations ensure that, besides the web browser based form of the MathDox format, other electronic or even paper versions of MathDox documents are possible.

2.3.1.3 *Separation of content and presentation*

The MathDox Player serves MathDox documents as XHTML documents to the user and uses CSS style sheets for styling issues. Any further distinction between content and presentation is achieved by means of semantic data. The semantic data is then used to interpret the content in the correct way during the translation process to a viewable document, usually an XHTML document.

2.3.1.4 *Being extendable*

MathDox is extendable. External applications can easily be embedded into MathDox by means of web-service calls. In this way MathDox documents are made suitable for almost any task involving external software. This approach is also used to connect various computer algebra systems to MathDox documents, so as to ensure that the user is not bound to one particular system, but can select the best suited computer algebra system for each computation. In fact, the invocation of a computer algebra system can be done in essentially two different ways: either the instructions for the computer algebra system are written in the universal language OpenMath [77], and, by use of external phrasebooks [142], encoded into — and later the results are decoded from — specific commands for the computer algebra system; or the instructions within MathDox are written straight in the specific command language for the computer algebra system.

Other features also support extendability; CSS files, applets, the open source license on the MathDox software and the modularity of the MathDox Player implementation. Flexibility is highlighted in more detail in Section 2.4.2. In Section 2.5.4 we will discuss projects that have made use of MathDox and demonstrated its flexibility.

2.3.1.5 *Representation of Mathematics*

Mathematics can be represented in OpenMath [77], MathML [65] and LaTeX. Via XSLT transformations, OpenMath and MathML can be transformed to LaTeX, which, with the help of MathJax [67], can be rendered within a web browser. MathJax translates LaTeX into either locally available suitable fonts, or images of appropriate sizes. Other approaches to render mathematics in a web browser are to make use of MathML (and skip translation to LaTeX) or to translate mathematics to images. MathML is not yet truly supported by Internet Explorer [41] and requires an additional plug-in before web pages with MathML are displayed properly. The need for a plug-in by Internet Explorer requires a lot of users to install it. This requires extra work and some users may lack the skill or the right permissions to do it. The need for a plug-in almost always disrupts the 'plug and play' feeling MathDox is aiming for. The Opera [80] and Mozilla Firefox [72] browsers have native support for MathML and do not require a plug-in. Chrome relies on MathML implementation in WebKit [105], which is work in progress. The translation to images, as done by MathJax, requires a dynamical translation of the mathematics. Moreover, images are known for their lack of scalability. The use of MathJax allows MathDox document authors to use either OpenMath, Presentation MathML or LaTeX for mathematics they want to be rendered in the web browser. See the Section 2.4 for more information on this process.

2.3.1.6 *Easy to use*

MathDox documents should be easy to access and to use. Users should not have to go through extra trouble to view a MathDox document in a web browser. This consideration played a major part in the decision (see Section 2.3.1.5) not to use MathML or images in the generated HTML pages destined for the users' web-browsers. *Easy to use* also means that the MathDox documents are provided with a formula editor [62] that enables the entering of mathematical expressions in an easy way and does not require existing knowledge about the underlying representation of math in LaTeX, MathML or OpenMath. More about this mathematical editor can be found in Section 2.4.3.

The use of existing standards helps authors to familiarize themselves with the MathDox format. To even further assist authors in the

Interactivity	Good
Usable in multiple formats	HTML & PDF
Separation of content and presentation	Good
Being extendable	Yes
Representation of Mathematics	Very good
Availability of a computer algebra system	Yes
Ease of usage	Limited
Use of open standards	Yes

Table 9: How well does MathDox meet the requirements

creation of MathDox documents, several editors have been developed, these are explained in detail in Section 2.5.3.

2.3.1.7 *Usage of open standards*

By combining existing standards MathDox profits from other techniques that have already proven themselves. This approach saves a lot of unnecessary work and helps new MathDox authors who are already familiar with some of these standards to lower the learning curve of the MathDox format. Authors also benefit from having existing documentation available if they are not yet familiar with these standards. Often tools working on sources with these standards are readily available.

In the same spirit, the MathDox software is available under the open source license LGPL [52].

2.3.1.8 *How well does MathDox meet the requirements*

The MathDox format produces documents that are meant for interaction with the user and as such scores well in this regard. With the addition of a context to MathDox, the user interaction will improve further. The MathDox documents are meant to be viewed in a web browser, therefore its main format is HMTL but PDF is also supported. The open source subformats used by the MathDox format are semantically rich — together with CSS files — allowing for a good separation of content and from. MathDox was designed with extendability in mind and practice this mindset in the way it connects to computer algebra systems. Representation of mathematics is done by MathML and MathJax and works well. Authoring a MathDox document involves a learning curve for new authors.

2.3.2 *The structure of MathDox documents*

The MathDox format needs a structure that allows MathDox documents to be processed in a logical and semantic manner.

The required structure is provided by DocBook. DocBook adds semantic structure to a document and is an open source XML format.

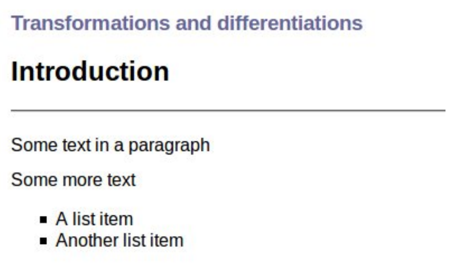
DocBook also has some disadvantages. It is more extensive than what was sought for and is mainly focused on static documents and lacks support for mathematics. The first disadvantage is minor, as no MathDox author is forced to use DocBook to its full extent. The second disadvantage is countered by the inclusion of other formats into MathDox such as XForms and Jelly: see Section 2.3.4 and Section 2.3.5. Inclusion of other formats also solves the lack of math support, namely OpenMath or MathML.

Listing 1: An example of DocBook code

```
<book>
  <title>
    <ulink url="introduction.md">
      <phrase role="title1">
        Transformations and differentiations
      </phrase>
    </ulink>
  </title>
  <subtitle>Introduction</subtitle>
  <sect1 name="introduction.mb" role="introduction">
    <title/>
    <para class="">
      Some text in a paragraph
    </para>
    <mediaobject>
      <imageobject>
        <imagedata fileref="image.jpg"/>
      </imageobject>
    </mediaobject>
    <para>Some more text</para>
    <itemizedlist>
      <listitem>A list item</listitem>
      <listitem>Another list item</listitem>
    </itemizedlist>
  </sect1>
</book>
```

Example 1:

See Listing 1 for an example of DocBook code. In this example the root tag is a *book* (it could also have been an *article*) element grouping the contents of a MathDox Document together. Next is a *title* element which contains a link (*ulink*) and a *subtitle* element stating the titles of this document. This document contains just one section which is grouped together in the *sect1* element. Within this section some paragraphs (*para*) exists. Also, a *mediaobject* is inserted which contains everything necessary for an image to be displayed. Finally, we see an *itemizedlist* which demonstrates an enumeration.



Transformations and differentiations

Introduction

Some text in a paragraph

Some more text

- A list item
- Another list item

Figure 1: The DocBook example rendered

2.3.3 *Semantic encoding of Math*

Within MathDox, mathematics appears in various forms: mathematics as used in computations by software packages or mathematics solely meant for the user to read or sometimes for both the computer and the user. For each type of usage one wants mathematics to have specific properties. In general a user will be able to grasp the meaning behind a, possibly ambiguous, mathematical expression, often helped by the context in which the expression appears. Computer software, however, will need its mathematics to be completely unambiguous, since it cannot benefit from the context in the way a mathematically skilled reader does in order to solve gaps caused by incompleteness and ambiguity. OpenMath [77] was chosen as the main format for mathematics within MathDox documents. It is well suited because it is semantically rich, unambiguous, XML-based and can easily be transformed into other formats, such as MathML and LaTeX, which are better suited for presentation. Ambiguity in mathematical expressions are solved in OpenMath by explicitly specifying the operator that is intended. As most operators are grouped into categories, it suffices to just name the group (called a *content dictionary*) and the desired operator. Different operators have either different content dictionaries, different names, or both. The display of OpenMath expressions is left to translators such as XSLT style sheets. These translators decide on the right presentation form.

Example 2:

To give the reader an idea of the OpenMath encoding of the semantics of a mathematical expression, we present a snippet of OpenMath in Listing 2 representing $1 + \sin(x)$:

Listing 2: An example of OpenMath code

```
1  <OMOBJ>
2    <OMA>
3      <OMS cd="arith1" name="plus"/>
4      <OMI>1</OMI>
5      <OMA>
6        <OMS name="sin" cd="transc1"/>
7        <OMV name="x"/>
8      </OMA>
9    </OMA>
10 </OMOBJ>
```

In the listing we find an OpenMath Object (`<OMOBJ>`) which is used as container to hold the OpenMath expression. Each OpenMath expression should start and end with this tag, although incorrect it is also sometimes omitted. Every expression contained by an `<OMOBJ>` then starts with an OpenMath Application (`OMA`). To indicate what kind of operator is required the OpenMath Symbol (`OMS`) is used with the attributes `cd` and `name`. In the listing the attributes of the first `OMS` (line 3) indicate that the used symbol is the plus operator from the content dictionary (cd) arith1. Each content dictionary, much like XML namespaces, defines its own definitions for the symbols they contain. This makes each combination of `cd` and `name` distinct and together with the definition semantically rich. Moreover, it prompts OpenMath Symbols to be abbreviated to the `cd` and `name` values, in this case: arith1.plus The OpenMath Integer (`<OMI>`) then specifies the integer value of 1. The second argument for the arith1.plus operator is another OpenMath Application, containing an OpenMath Symbol transc1.arith1 [5]. and the OpenMath Variable (`OMV`) x.

5 The definition for arith1.sin can be found in the content dictionary [78] and is as follows: This CD holds the definitions of many transcendental functions. They are defined as in Abromowitz and Stegun (ninth printing on), with precise reductions to logs in the case of inverse functions. Note that, if signed zeros are supported, some strict inequalities have to become weak. It is intended to be 'compatible' with the MathML elements denoting transcendental functions. Some additional functions are in the CD transc2.sin
Role: application
Description: This symbol represents the sin function as described in Abramowitz and Stegun, section 4.3. It takes one argument.
Commented Mathematical property (CMP): $\sin(x) = (\exp(ix) - \exp(-ix))/2i$

> Note that if different definitions for symbols are required, alternative definitions can be created in a new different content dictionary.

MathDox documents use the OpenMath representation where semantics are important and can switch to MathML or LaTeX when the mathematics need to be presented to the user either on screen or on paper. The semantic structure offered by OpenMath also allows for easy manipulation of a mathematical expression, i.e. to select, verify, evaluate, (re)use it in computations or to present it without the need of the surrounding expression and without losing its meaning.

Mathematics solely meant for presentation and not needed for computations does not necessarily have to be written in OpenMath and may be written directly in MathML or LaTeX. Likewise mathematics only used by a specific computer algebra system may be written in the language of the used computer algebra system.

2.3.4 *User driven interactivity*

XForms [113] is a development which became a W3C recommendation [114] in October 2003 for version 1.0. XForms 1.1 received a W3C recommendation [115] in October 2009.

XForms is meant to replace the current HTML forms in web pages. For this purpose XForms offers new form elements that take over the role of the existing HTML form [36] elements and the need of JavaScript [45]. Any data collected by XForms elements is stored in an XML form that is sent along with a submit event. Also XForms allows rules, for example bindings, that determine when elements should become visible, active, or disappear. Just like JavaScript, XForms is a client based format, enabling actions to be taken without communication with the server.

XForms in MathDox documents provide means of interaction with the reader of the MathDox document. It allows for many interaction (form) elements which are used for user input, such as text fields, radio buttons, and drop down menus.

A simple example of XForms code is shown in Listing 3. The upper listing is the model, an XML document, used by XForms to store values as they are set by the XForms controls on the HTML page. When a submit occurs, this model is being sent back to the server, notifying the MathDox Player of the new settings. Note that the <*a*> element contains a default value of 1. The second listing contains two *input* elements. These represent input fields directly linked to the elements in the model just described. Initially the first input field linked to *a* will show the default value of 1. The *submit* element at the bottom of the listing represents a button labeled *submit*. When this button is pressed the (modified) model will be sent to the server.

Listing 3: An example of XForms code

```
<xforms:model>
  <xforms:instance>
    <variables>
      <a>1</a>
      <b/>
    </variables>
  </xforms:instance>
  <xforms:submission method='post' action='aWebpage'
    id='submission'/>
</xforms:model>

<xforms:input ref='a'>
  <xforms:label>a</xforms:label>
</xforms:input>
<xforms:input ref='b'>
  <xforms:label>b</xforms:label>
</xforms:input>

<xforms:submit submission='submission'>
  <xforms:label>submit</xforms:label>
</xforms:submit>
```

XForms is included into the XHTML 2.0 draft specifications [126]. XHTML 2.0 was meant to become the successor of XHTML, however the development was canceled in favor of HTML5 [198]. Work to support XForms natively by browsers has been done [116, 26], but stalled after the cancellation of XHMTL 2.0. Examples of plugins are the formsPlayer [26], a plug-in for Internet Explorer browser and the Mozilla XForms project for Mozilla web browsers.

For now the XForms code in MathDox documents is translated automatically by the MathDox Player into HTML forms supplemented by JavaScript, making sure it works whether or not the browser has a plug-in for XForms support. This translation is done by Orbeon Forms [81], see Section 2.4.1.

2.3.5 *Programming and scripting*

To specify and fine-tune reactions of a MathDox document to user input, an author needs programming constructs. Most notably a MathDox page needs *if* statements to test conditions, variables to set and store values — both normal and XML values — to be used on the page, the means to work with XML snippets, and constructs to call other web services (see Section 2.3.6). As Jelly [47] offers these properties and is both an XML format and open source, Jelly has been included in the MathDox format.

Jelly is a JSP-like [43] XML-language, and has been developed as part of the Apache project [5]. Jelly can be used for conditional statements, loops, variables, and for calls to Java objects and web services.

Example 3:
See Listing 4 for an example of some Jelly code. This example starts by declaring two variables a and b and setting their values to 1. Then a *while* loop will be entered with as condition that it continues as long as $a < 100$. In the *while* loop, variable a will be printed. Next, the variable c will get the sum of the values of a and b; then a will get the value of b and b will get the value of c. This little example calculates all Fibonacci numbers less than 100.

Listing 4: A Jelly Fibonacci example

```
<c:set var='a' value='1'/>
<c:set var='b' value='1'/>
<c:while test='$(a &lt; 100)'>
  ${a}
  <c:set var='c' value='${a+b}'/>
  <c:set var='a' value='${b}'/>
  <c:set var='b' value='${c}'/>
</c:while>
```

Listing 5: Output of the Jelly Fibonaci example

```
1 1 2 3 5 8 13 21 34 55 89
```

Example 3 is a rather simple example for sake of clarity; however, Jelly offers many more possibilities for a MathDox page. Just imagine what scripting can do with parameters from a user or random chosen set of values, when applied to an example or exercise.

2.3.6 *External services*

The MathDox Player facilitates the use of external services. Whenever a MathDox document requires an external service, it makes a call to that service and incorporates the reply in its output. The web calls required for these external services are made by a SOAP [97] call, facilitated by the Jelly format.

Examples include the call to the Natural Language Generator [73] as performed in the WebALT project [193, 104]. Here an external service translates a semantically rich abstractly formulated sentence encoded in OpenMath into a natural language sentence written in the language chosen by the user. The MathDox document then displays this sentence to the user. Translating exercises from a meta format into multiple different natural languages.

Another more commonly used application is the use of external computer algebra systems. The computer algebra systems currently supported by MathDox are; Mathematica [63], Maple [56], GAP [27], Maxima [69], WIRIS [108], Magma [55] and Singular [96]. The interoperability between these systems is achieved by making use of OpenMath and OpenMath phrasebooks [6]. There is a separate OpenMath phrasebook [142] for each of the aforementioned computer algebra systems. These phrasebooks handle the translation from OpenMath to the specific computer algebra system language and vice versa. This effectively enables MathDox documents to communicate in OpenMath with different computer algebra systems.

Since it is very common in MathDox to have a computer algebra system perform computations and because an author should not be bothered with the specifics of each computer algebra system, a specialized format MONET [70], was selected to assist in calling the different computer algebra systems in a uniform way. OpenMath expressions that need computation are therefore encapsulated in MONET expressions.

Listing 6: An example of a MONET query

```
<monet:query xmlns:monet="http://monet.nag.co.uk/monet/ns">
  <monet:classification>
    <monet:directive-type
      href="http://mathdox.org/phrasebook/mathematica#eval"
    />
  </monet:classification>
  <monet:body>
    <monet:output>
      <x:copyOf select="$monetExpr/*" xmlns:x="jelly:xml"/>
    </monet:output>
  </monet:body>
</monet:query>
```

Example 4:

In Listing 6 an example MONET query as used in a MathDox page is shown. This code snippet begins with the root element `<monet:query>`, which in turn contains an `<monet:classification>` and `<monet:body>`. Inside the `<monet:classification>` tag is the directive-type element `<monet:directive-type>` with the attribute *href*. The *href* attribute specifies what needs to be done, here it reads

6 A phrasebook is a wrapper around a computer algebra system. It is responsible for translating OpenMath expressions into the format of the computer algebra system, and the other way around. Upon receiving an OpenMath expression it will forward it to the computer algebra system and returns the result again in OpenMath.

`http://mathdox.org/phrasebook/mathematica#eval`. The first part tells which phrasebook to use, in this case the Mathematica phrasebook. The last part asks for an evaluation from this phrasebook. the `<monet:body>` contains a element called `<monet:output>` which contains the OpenMath expression that needs to be computed.

An observant eye notes that in this example there is no OpenMath expression. At least, not directly listed in the body. The OpenMath expression as required by the `<monet:output>` element is contained in the Jelly XML variable called `monetExpr`. The tag `<x:copyOf/>` copies the content of this variable into the MONET query. As such it demonstrates how OpenMath expressions can be stored, before they are used either for display in the web-browser or for computations, such as in this case.

The MathDox group participated in the SCIEnce [94] project (Symbolic Computation Infrastructure for Europe) funded by the European Commission. SCIEnce was backed by four computer algebra teams: GAP, KANT, Maple, and MuPAD. The project developed a new communication protocol with computer algebra systems: SCSCP [175] (Symbolic Computation Software Composability Protocol). SCSCP supports OpenMath and may therefore be a likely successor of the current MONET approach.

2.3.7 *Separating functionality in MathDox files*

MathDox documents do not need to be a single big file. There are cases in which it is advantageous to split documents into smaller parts. The resulting set of parts will increase possibilities for reuse and support maintainability. XInclude is used to include and group together XML components from different files. XInclude is a W3C recommendation [117].

2.4 MATHDOX SOFTWARE

So far we have only discussed the MathDox format and mentioned the server-side software responsible for the execution of this format. In Section 2.4.1 we will explain how the server sided MathDox player works. The possibilities of customizing the MathDox software to the specific needs of alternative applications which make use of the MathDox system are discused in Section 2.4.2. We conclude this chapter with Section 2.4.3 about the MathDox Formula Editor.

2.4.1 *The MathDox Player*

The MathDox Player is responsible for making MathDox documents accessible over the web. Its task is similar to that of a web server or rather an application server in the sense that both a web server and the MathDox Player offer stored documents from the server to the outside world. A web server offers (ready-made) HTML files, while an application server provides a web server with dynamically created HTML files. The conversion of the MathDox documents into HTML pages is the main task of the MathDox Player. We will now explain how this process works.

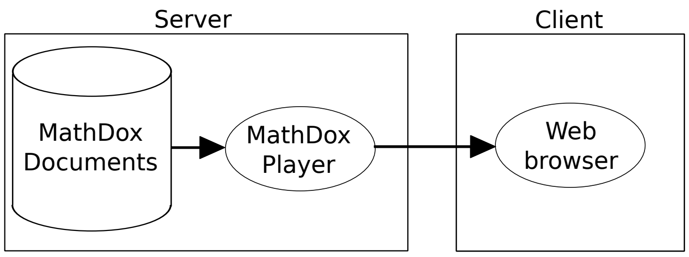


Figure 2: MathDox Player overview

The MathDox Player is implemented in Orbeon Forms [81], a Java Servlet application. Servlet applications are run within an application server that is designed to the Java Servlet specification [44], these application servers are called Java Servlet containers. The best-known Java Servlet containers are Apache's Tomcat [101] and JBoss [46]. Orbeon Forms offers some useful tools for the translation process. The most notable is an XSLT [118] processor. XSLT is a format especially suited for transformations from an XML format into another data format, often but not necessarily XML. XSLT is used extensively by the MathDox Player for transformations on the MathDox format.

The MathDox format is a combination of several different XML formats (see Section 2.3). Each XML format used within MathDox requires a separate XSLT translation. For instance, there is an XSLT transformation that translates the DocBook structure into an HTML structure, while other scripts translate the mathematics in OpenMath to MathML and LaTeX.

Just before the start of the translations, a copy of the MathDox document is created and stored in memory. The XSLT-translations are then performed sequentially and each of them places the results of the translation to this local copy in memory. In this way the MathDox document gradually changes into a HTML web page. An overview of all steps needed in the translation process is presented in Figure 3. In each step one of the subformats is converted into one of the remainder formats.

1 Macro substitution.
Macros used by the user need to be substituted for the MathDox

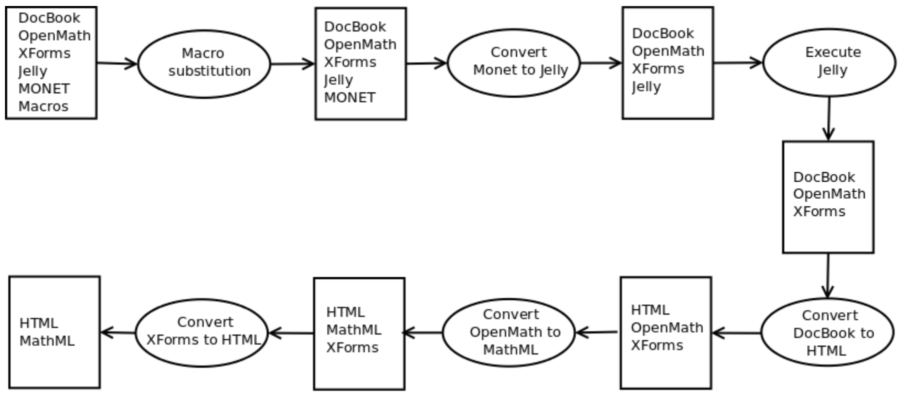


Figure 3: MathDox translation pipeline

code they represent before the translation from MathDox to HTML can start. This makes the substitution step the necessary first step of the translation process.

2 *Conversion of MONET to Jelly/JavaScript.*
Since the actual calls to web services are done by Jelly code, an important step in the translation process of MathDox is the translating of MONET queries used for calls to computer algebra systems into Jelly code. The queries translated to Jelly code will then be executed together with the other Jelly code in the next step. For instance, when a MONET query is encountered that requests evaluation of an expression, it does not matter if that expression is $\log(2+4x)$ or $\log(2x+1)+\log(2)$ because both are usually considered equal by a computer algebra system, the generated Jelly code will contain the statements for calling a specific computer algebra system that is configured for use with the MathDox Player, and that can handle the evaluation.

Alternatively MONET code can also be translated into JavaScript to facilitate communication with applets on a MathDox page since both JavaScript and applets are client based. Only applets that know what to do with these mathematical instructions should be addressed by these MONET queries. Geogebra [29] is an example of such an applet, see also Section 2.4.2.

The translation of MONET into Jelly or JavaScript code is handled by XSLT transformations.

3 *Execution of Jelly code.*
After the translation of MONET queries into Jelly code, all Jelly code is executed. This includes the just translated MONET expressions and the Jelly code that was already present in the MathDox document. It is the Jelly code that decides what should be presented on a page, and takes care of conditional statements and calls to web services. It

is through execution of the Jelly code that the content of a web page created by a MathDox document is largely formed.

4 Conversion of DocBook to HTML.
Next in line is the conversion of DocBook. DocBook is used for document structure and is translated into HTML, again with the use of an XSLT script.

5 Conversion of mathematical expressions.
The transformation of OpenMath in order to render mathematics in a web page is a bit more complex. First, one should know the mathematics expressions still on the MathDox page at this stage are solely meant for presentation. Mathematical expressions meant for computations have already been processed by a computer algebra system by this point. If the mathematics in OpenMath needs to be shown on the screen (as is the case at this point), one would normally translate OpenMath into MathML-Presentation code. There is a freely available XSLT script [79] that does exactly this. Although MathML is supported either natively or by plugins by the most popular web browsers, too many users seem unwilling or incapable to install the required plugin, and the browsers themselves have different approaches to using these plugins. See Section 2.3.1.5 for a previous discussion on this subject. It was therefore decided not to use MathML directly in the users' web browser for now, but to continue with a translation from MathML to LaTeXwith another XSLT script [118]. As native MathML support in web browsers increase, this step may be omitted in the future. The JavaScript program MathJax [67] is used for the final translation at the users' web browser to a suitable font installed in the users' browser. The use of MathJax circumvents the use of plugins, and results in the rendering of mathematics without any action required from the user.

Note that all mathematics meant for presentation on the web page is translated first into MathML-presentation and next into LaTeX. Since these translation steps are always executed in this sequence, it is possible to use MathML-presentation or LaTeX directly instead of OpenMath. As long as it is only used for mathematics that is going to be rendered, and not for mathematics that is needed for computations, no harm is done.

6 XForms conversion to HTML.
The final step is converting XForms to HTML and sending the constructed HTML page to the user. This step is executed by Orbeon Forms [81]. It first translates all XForms elements to JavaScript and HTML and then sends the entire translated page, written in HTML and containing some JavaScript code, to the web browser of the user who requested the page.

2.4.2 *Extending the MathDox format*

The MathDox system has been designed in such a way that it is easy to extend. This is especially useful when MathDox is used for more specialized environments or tasks.

MathDox can be extended in at least four different ways. These approaches are not mutually exclusive and combinations are possible.

First we describe an approach we have already discussed is the use of external web services. External web services as used in MathDox documents to connect to computer algebra systems are not limited to computer algebra systems alone; any web service application can be called from within a MathDox document. Incorporating the results of these services within the MathDox document enriches the MathDox documents with dynamic content of any nature. Web services are called by inserting appropriate Jelly code into the MathDox document.

The second approach to extension is the possibility for adding media elements such as images, movies, applets and flash. Including images and movies — although not interactive — into a MathDox document is an easy approach to create a more specialized environment. By adding applets and flash applications to MathDox documents, more specialized and interactive content and behavior can be added to the generated web pages. An example of such specialization is the inclusion of an applet editor WExEd [147] to create sentences with the help of defined semantic OpenMath symbols. At this point, communication support for the Geogebra [29] and the WIRIS Math Editor [109] applets have been implemented. This implies that data output from these applets is directly available for usage elsewhere on a MathDox page, by means of JavaScript. Other applets may be used on MathDox documents as well, although such a data link has not been implemented for these other applets as yet.

A third way to extend behavior of MathDox documents is by developing and adding Jelly [47] custom tag libraries. A Jelly custom tag library is written in Java and offers a set of tags which can be used from within the MathDox format. Each tag connects to a Java class that is instantiated and called when the tag is called from the MathDox document.

An example of this is shown in Listing 8, which displays Java code called by a Jelly custom tag. This tag is called `<random>` and needs a variable name and a minimum and maximum boundary. It will return a random integer into the variable whose value will be in between the listed boundaries. The random integer is gotten by executing the Java code also listed in Listing 8. This code is the implementation of the random tag. Of course many other Jelly custom tags can be created and called in the same way.

Listing 7: A call to a Jelly custom tag

```
<mdu:random xmlns:mdu="jelly:org.mathdox.util.UtilLibrary"
            var="a" minimum="1" maximum="6"/>
```

Listing 8: A code example of a Jelly custom tag

```
public void doTag(XMLOutput xml) {
  JellyContext jellyContext = getContext();

  Random random = new Random();
  try {
    int result = random.nextInt(max - min) + min;

    if (var != null) {
      context.setVariable(var, result);
    } else {
      xml.write("" + result);
    }
  } catch(SAXException exception) {
    exception.printStackTrace();
  }
}
```

The fourth approach for extension is by programming alterations in the translation process itself. The MathDox player has been designed and implemented in a very modular way which allows inheritance similar to inheritance in Object Oriented programming languages. Alterations to XSLT style sheets and XML pipelines (as shown in Figure 3) can be created in such a way that there is no need to alter the original code, making it possible to reuse as much as possible.

2.4.3 *The MathDox Formula Editor*

MathDox comes equipped with the MathDox Formula Editor. The MathDox Formula Editor is a specialized JavaScript program that helps users enter semantically rich mathematical expressions in OpenMath format. It does so by allowing users to *point and click* on mathematical symbols as well as to type expressions directly into the input field. The MathDox Formula Editor will translate the entered mathematical expression directly into OpenMath. The formula editor also supports translation into MathML.

An editor like this not only makes entering mathematical expressions easier, it also eliminates the need for users to know about mathematical formats such as LaTeX, OpenMath or MathML. Other programs having the same goal are the WIRIS Input Editor [109] and DragMath [17].

The MathDox Formula Editor is written in JavaScript. This allowed a lot of flexibility in layout. For instance, the editor's input field is always as small as possible when it is started. When expressions are entered, it scales to fit the written expression, using only what it really needs and leaving precious space on the web page available for other content. These can be text, images or even applets. Applets like the WIRIS Input Editor [109] or DragMath [17] do not resize dynamically and therefore either reserve more space inside their displays than they need, or cannot fit the expressions properly. Another feature is the shared floating palette with clickable expression elements used by the editor. Sharing this palette between multiple instances of the MathDox editor reduces the number of palettes required and again saves space on a page.

2.5 MATHDOX AT WORK

In this section a few practical examples of MathDox will be presented. We will pay attention to the creation process of MathDox documents and present a track record of MathDox in various scientific projects.

2.5.1 *MathDox examples*

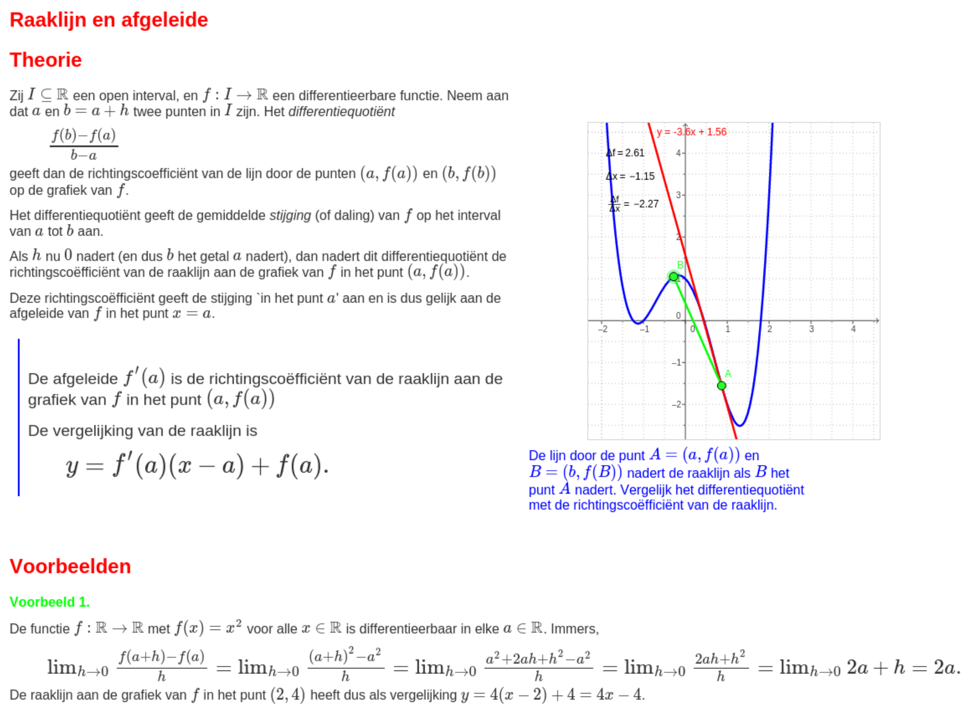
We present some examples that are taken from projects, the complete list of projects MathDox has been involved in is listed in Section 2.5.4. They will give a sample of the wide range of applications in which MathDox can prove its value.

Dynamic explanations.
Interactive mathematical documents are well suited to explain the workings of a mathematical concept. In Figure 4 a MathDox document from the Oncourse project [76] is shown. The example presented here, aimed at future freshmen students, asks for a tangent line to a function f. It demonstrates the use of the derivative to find the correct slope of the tangent line.

On MathDox pages like these, the function f is expressed in OpenMath and the derivative is computed by a computer algebra system and sent back in OpenMath. Ready to be rendered on the screen for students to see, but also to be used for further computations. Note that in this way the results found will still be valid when the original function f is changed, or when another point for the tangent line is chosen. This is supported by offering input fields for both f and the x-coordinate for the point at which one wants to compute the tangent.

Exercises in MathDox.
MathDox documents are also used to create exercises which allow users to test their skills in a mathematical topic, see also [154]. Basically two types of questions are possible; *multiple choice* and *open questions*.

Multiple choice exercises are the easiest way to question users in an electronic environment. The user selects one or more answers from a



Raaklijn en afgeleide

Theorie

Zij $I \subseteq \mathbb{R}$ een open interval, en $f : I \to \mathbb{R}$ een differentieerbare functie. Neem aan dat a en $b = a + h$ twee punten in I zijn. Het *differentiequotiënt*

$$\frac{f(b)-f(a)}{b-a}$$

geeft dan de richtingscoefficiënt van de lijn door de punten $(a, f(a))$ en $(b, f(b))$ op de grafiek van f.

Het differentiequotiënt geeft de gemiddelde *stijging* (of daling) van f op het interval van a tot b aan.

Als h nu 0 nadert (en dus b het getal a nadert), dan nadert dit differentiequotiënt de richtingscoëfficiënt van de raaklijn aan de grafiek van f in het punt $(a, f(a))$.

Deze richtingscoëfficiënt geeft de stijging `in het punt a' aan en is dus gelijk aan de afgeleide van f in het punt $x = a$.

De afgeleide $f'(a)$ is de richtingscoëfficiënt van de raaklijn aan de grafiek van f in het punt $(a, f(a))$

De vergelijking van de raaklijn is

$$y = f'(a)(x-a) + f(a).$$

De lijn door de punt $A = (a, f(a))$ en $B = (b, f(B))$ nadert de raaklijn als B het punt A nadert. Vergelijk het differentiequotiënt met de richtingscoëfficiënt van de raaklijn.

Voorbeelden

Voorbeeld 1.

De functie $f : \mathbb{R} \to \mathbb{R}$ met $f(x) = x^2$ voor alle $x \in \mathbb{R}$ is differentieerbaar in elke $a \in \mathbb{R}$. Immers,

$$\lim_{h\to 0} \frac{f(a+h)-f(a)}{h} = \lim_{h\to 0} \frac{(a+h)^2-a^2}{h} = \lim_{h\to 0} \frac{a^2+2ah+h^2-a^2}{h} = \lim_{h\to 0} \frac{2ah+h^2}{h} = \lim_{h\to 0} 2a+h = 2a.$$

De raaklijn aan de grafiek van f in het punt $(2, 4)$ heeft dus als vergelijking $y = 4(x-2)+4 = 4x-4$.

Figure 4: Dynamic explanation of a tangent line in Dutch

list and the software only needs to know which answers are correct. It does not have to compute mathematical expressions to verify the answer of the user. However, multiple choice questions may not challenge users enough to come up with answers on their own. The small set of answers already warns the user of possible mistakes, since the answer they are looking for may not be in the list. Finding an answer in the list confirms users that they reached the right answer. If this answer is in fact incorrect it will confuse users as they just had confirmation on their beliefs that they did it right.

Checking the answers of open questions is much harder to do in an electronic environment than checking the answers of multiple choice questions. Within MathDox it is possible — with the use of a computer algebra system — to compare the answers given by the user to those of the author of the exercise. The computer algebra system also assists in eliminating the problem that mathematical expressions can be written in many different ways while the semantic meaning remains the same. This allows readers to enter answers that are a variation of the answer as entered by the author. For instance, if the answer given by the reader is $\sqrt{8}$ whereas the author's answer to the question had been $2\sqrt{2}$, the answer given by the reader need not be marked as incorrect. The computer algebra system will evaluate the specified answer of the author and the answer of the reader as equivalent.

The strength of a computer algebra system to cope with variations of answers also brings a disadvantage when an author requires a precisely formulated answer. Sangwin [191] goes into deeper detail regarding computer algebra system related challenges. For example say a student is asked to answer a question with an integer and integer only. Instead of answering 7, the student answers $3+4$. The latter answer is evaluated as being correct when it is evaluated by a computer algebra system. Due to the semantic structure of OpenMath the answer can be analyzed by MathDox Player and the occurrence of the addition symbol will cause the answer to be recognized as not being an integer, and as such will not be accepted as a correct answer to this exercise. Here we use the power of the scripting format Jelly [47]. Indeed, with the use of XPath expressions [145] the XML structure of the OpenMath answer of the student expression can be inspected. The XPath expression displayed in Listing 9 actually tests for the occurrence of the addition symbol in the answer given by a user. It checks for any (recursive) occurrences of `<OMS>` tags that contain `@name='plus'` and `cd='arith1'` attributes. Any that are found are returned representing a positive value, which is then negated by the `not()`, resulting in a `false`. If none are found the expression returns a `true`. Rather severe restrictions can be forced upon OpenMath expressions as they are given as input from users. For example it is possible to test for occurrences of $2\sqrt{2}$ in favor of $\sqrt{8}$ or to test for $(x+1)^2$ instead of x^2+2x+1 and many others.

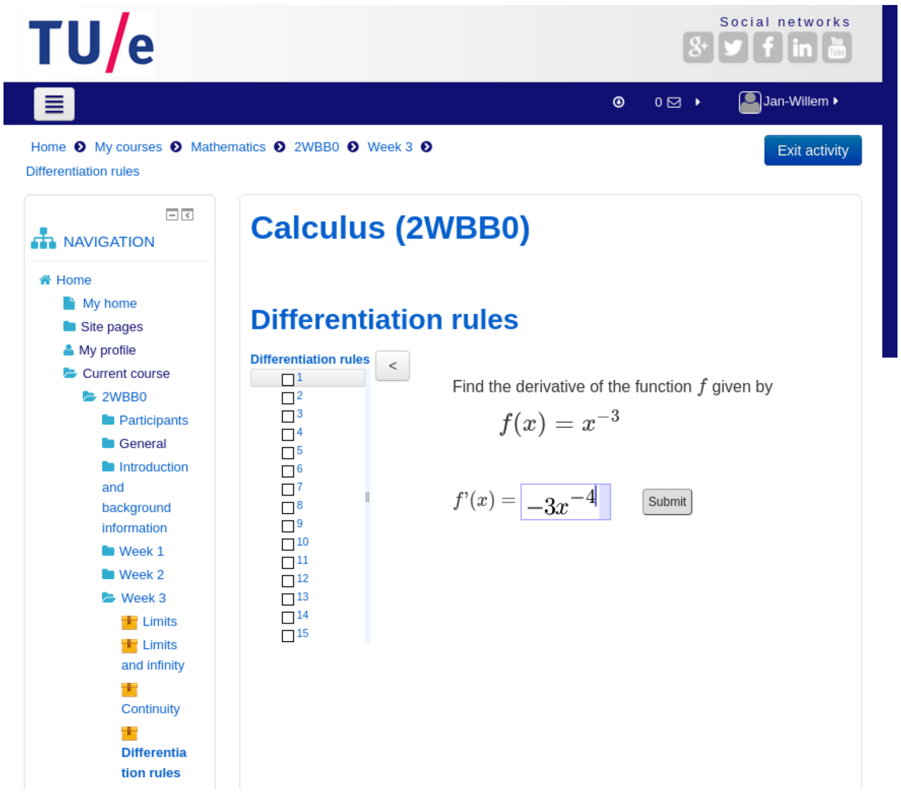
Listing 9: An XPath expression that verifies that no addition is used.

```
not(answer//OMS[@name='plus' cd='arith1'])
```

Exercises written in MathDox also allow the possibility of randomized values to be used in questions. The expected solution to a question is then calculated from these randomized values. This makes it possible to generate a near infinite amount of exercises which users can practice. Because these exercises will always be different, users may re-do the same exercise over and over with different results each time.

Despite the fact that MathDox can be used to create exercises, MathDox was never intended or designed to be a Learning Management System (LMS) like Blackboard [7], Moodle [71], or Sakai [92]. Normal LMSes that automatically process answers and grade them can only work with multiple choice and have trouble with display of mathematics.

A SCORM (Sharable Content Object Reference Model) [95] tool was developed to create SCORM packages that embed MathDox documents into an LMS. The MathDox documents included in a SCORM package do not have to be exercises, it is also possible to embed theory or explanations as in this way, see also Section 2.5.1. In Figure 5 an example is shown of a MathDox exercise within an LMS.



Find the derivative of the function f given by

$$f(x) = x^{-3}$$

$f'(x) = -3x^{-4}$

Figure 5: A MathDox SCORM package in Moodle

2.5.2 *Exercise graphs*

Exercises in MathDox can be created to contain multiple steps. Such exercises are also known as a solution graphs or exercise graphs. An exercise graph is a graph where each node is a step and every edge represents a condition that has to be met by the user in order to navigate to an adjacent node. These conditions usually involve user's answers to questions. As such they are examples of the possibilities of programmable reactions in MathDox. These exercises can be specifically designed to capture mistakes in order to address them and correct the understanding of the user; in such cases they are effectively implementations of the bug model, as described in Section 3.1.5. Reactions to an incorrectly answered exercise may vary from a hint, a reference to the theory, or by simplifying the question.

Of course the other way around is also possible; the exercise starts simple and becomes more complicated as long as the answers are correct. In this way exercises are created that will adapt to the skill of the user. These kinds of exercises are already possible with MathDox in its current state, that is, without the enrichment of a context or any other adaptive features as described in this thesis. What makes this possible is something we call *implicit context*. An *implicit context* is not a *context* or *user model* — we will explain these later in Chapter 3 — in the sense of storing data about a visiting user; in fact it does not store anything. Implicit context is the set of answers that navigates an exercise graph. The mere presence of a user on a certain vertex says something about their answers, because we know what actions they must have taken to get there.

Example 5:

Student Steven visits a MathDox page that presents him with a differentiation exercise, as shown in Figure 6. The exercise Steven has to solve requires him to make use of the *chain rule*, a rule Steven is not that familiar with yet. As a result of his weak understanding of the *chain rule* he answers the question incorrectly. Analyzing his answer, the MathDox page recognizes the problem to be a very basic misunderstanding of the *chain rule*; it therefore selects a new MathDox page that presents Steven with a simplified exercise that deals directly with the principles of the *chain rule*. It even offers Steven a link to yet another page in which the *chain rule* is explained in detail if such is deemed necessary by Steven. However Steven recognizes his error and remembers the basics of the *chain rule* again and answers the second question correctly. The MathDox exercise validates the answer as correct and presents Steven next with a slightly more complex exercise still related to the *chain rule*. Having answered this question correctly also, Steven is finally taken back to the original exercise, being offered a chance to prove he learned what his mistake was by answering the original exercise correctly.

Even though the MathDox pages only have the answers of Steven to work with and do not have anything stored about his past actions, answers or knowledge, the MathDox pages are still able to understand the problem at hand and address it and guide Steven back to the original problem.

In Example 5 the exercise graph could be expanded with a number of other nodes and edges that address unique error situations, but these edges need a valid error recognition too. This brings us to the question of what should and what should not be added to such a graph. A user may make a combination of errors: in addition to struggling with the chain rule they might also be incorrectly differentiating sin or log. We need to be able to recognize the possible and actual mistakes being made before they be addressed.

Exercise graphs can be made extremely complex with a lot of nodes and edges. A big graph with a decent amount of answer analyses linked to edges can really tune in to the abilities and misconceptions of a user. However, currently an exercise graph or a normal Math-

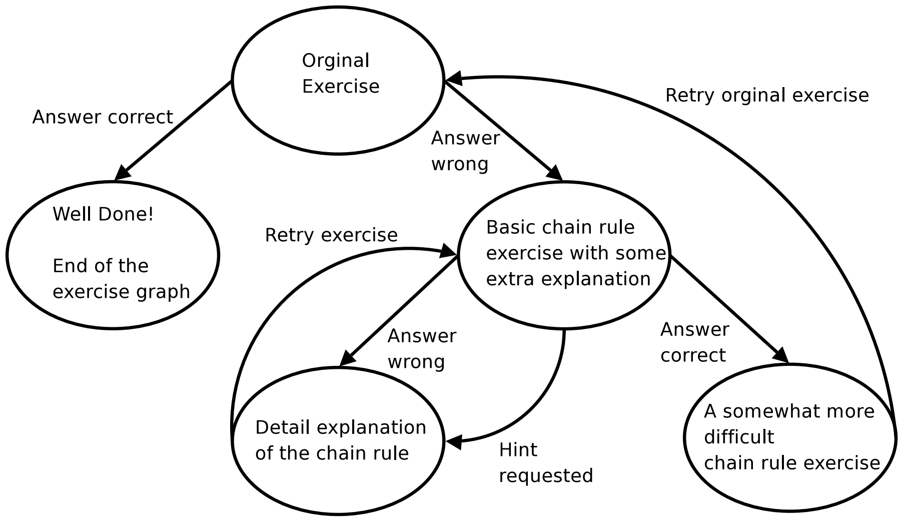

Orginal Exercise
Answer correct
Answer wrong
Retry orginal exercise
Well Done!
End of the exercise graph
Retry exercise
Basic chain rule exercise with some extra explanation
Answer wrong
Answer correct
Detail explanation of the chain rule
Hint requested
A somewhat more difficult chain rule exercise

Figure 6: MathDox exercise graph

Dox page cannot distinguish or remember users from previous sessions, so when a user closes the browser he or she needs to start all over again to continue where they left of the previous time. Although MathDox is interactive it is not personalized, it lacks a context.

2.5.3 *Authoring*

Several tools exist to help with the creation of a MathDox document. Among these are a LaTeX package [169], the Mexico Editor, the LeAMEL Exercise Editor and MathDox macros, see the MathDox Manual [197, 153] for details. Of course there is also the option to write MathDox code directly in your favorite text or XML editor.

Creating a MathDox document is very much like creating an interactive (mathematical) web page. The more complex the MathDox page needs to be, the more programming like skills are needed to create a MathDox document. The various subformats are all open formats and well documented, which assists in the writing of a MathDox document. For those not looking forward to mastering the programming like skills needed to write a MathDox document, a specialized LaTeX package may be of assistance. This LaTeX package allows MathDox content to be written in LaTeX. The MathDox Player will then translate the LaTeX code into MathDox code and from there it will be interpreted as if it were written as normal MathDox code. This approach enables an author to skip the learning process of the MathDox format altogether. MathDox documents written with the LaTeX package have some restricted functionality as not all functionality of the MathDox format can be expressed in the LaTeX package.

Closely related to this is the opportunity to translate MathDox documents back to LaTeX. The resulting LaTeX documents are just for presentation and cannot be translated back to MathDox code in the same way we just described. Also these LaTeX documents are no longer interactive and therefore, for instance, cannot support any real time computations.

Closely related to the macros as defined in the LaTeX package, are the macros as defined in the MathDox format. As can be seen in Figure 3, the translation process of the MathDox format begins with translating any defined macros into proper MathDox code. The use of these macros helps simplify the writing process of a MathDox document to some extent.

A different approach to make authoring mathematical content in MathDox easier, is the Mexico editor. This editor expands upon the MathDox Formula Editor [62] and combines instances of the Formula Editor together with editors for typed text. Together this creates a mathematical document that can then be displayed in any web browser with the help of the MathDox Player. The Mexico editor does not require any knowledge about the MathDox format nor about LaTeX and is therefore ideal for authors unfamiliar with either of them and who want to create a simple page with some mathematics and text.

Authors who want to use MathDox to create interactive exercises and again do not want to have anything to do with the MathDox format may use a special editor for the purpose of creating more complex multi-layered exercises. This editor was created during the course of the LeActiveMath project [51], and produces exercise graphs as discussed in Section 2.5.2. These exercise graphs are written in a language called Le ActiveMath Exercise Language or LeAMEL for short. LeAMEL is automatically translated into MathDox code by the MathDox Player, but is not considered a part of the MathDox format itself.

Finally MathDox documents can also be written with the help of a text editor: preferably one that can assist in validating XML. This is without a doubt more difficult than any of the other options, but also offers the most features and opportunities for interactive mathematical documents.

2.5.4 *MathDox at work in various projects*

MathDox was and is being used in several projects at high school, university, national and European level. We will describe some of them below.

- Wortel TU/e [112], Oncourse [76] and Experience Mathness [23] are projects aimed to detect and repair mathematical deficiencies in the knowledge of (future) first year students at the Technical University of Eindhoven. MathDox exercises are used for testing and to provide feedback.

- Algebra Interactive [148, 2] is an interactive method for teaching algebra to first year undergraduates. Algebra Interactive is not only available in electronic format, but also on paper as a textbook.

- MathAdore [60] is a web based mathematics text book for Dutch high schools, developed by Pragma-ADE [85], Math4all [59] and the DAG group [15] at the TU/e. The projects' aim is to produce a new generation of high quality teaching material incorporating various forms of interactivity. MathDox is used to provide interactivity in both examples and exercises.

- LeActiveMath [51] is a European research project that focused on web-based, multilingual e-learning. The role of MathDox tools within this project was to provide a repository of exercises that can be authored using a web-based editor.

- WebALT [104] is another European project. The WebALT project ran from 2005 to 2006. The project's goal was to create the necessary tools for the creation and maintenance of interactive multilingual mathematical exercises. The multilingual aspect was achieved by representing natural language in a special OpenMath language, which can be translated into several natural languages. The implemented languages were: Italian, French, English, Swedish, Spanish and Catalan. Within the WebALT project an exercise editor was required to create these multilingual mathematical exercises. The editor was written in MathDox and was co-developed for both the LeActiveMath and WebALT projects. The MathDox Player used for the WebALT Editor (WExEd [147]) was adapted with web service calls to the Natural Language Generator [73].

- Intelligent feedback [40] was a Surf-funded project and investigates e-learning tools for linear algebra that give feedback on the level of strategies.

- Wiskunde D [110] is a new subject in Dutch high schools. It is a mathematical course that deals with the more interesting aspects of mathematics at the level of high school students. MathDox was used to create some of the interactive course materials.

- Onbetwist [75] is part of of the "toetsen en toetsgestuurd leren" (test and test-driven learning) program of SURF [98]. It aims to organize national standardized tests for mathematics at higher education level, supported by a database of tests and practice materials. Onbetwist is a cooperation of University of Hasselt, University of Utrecht, Open University, Fontys, Maastricht University, University of Amsterdam and Technical Univeristy of Eindhoven.

- Technology Enhanced Learning of Mathematics for Master Education (TELMME) [99], is a project of the federation of the three technical universities (3TU) of the Netherlands. Like the Wortel TU/e project it aims to mathematically prepare students of –Dutch or international– hoger beroeps onderwijs (higher professional education) schools for a master studies at one of the participating technical universities. MathDox is used to present mathematics via the web to these students. Telmme ran from 2009 till 2011.

SUMMARY

Interactivity in web pages on mathematics is not common. Complexity in calculations, displaying and entering mathematical notations are the main reasons that fully consistent and integrated interactive mathematical documents are scarce. MathDox is a system that consists of an XML based format and a set of tools that transforms said format into dynamic and interactive mathematical web pages.

MathDox supports interactivity by combining interactive web pages with the power of mathematical computations. Since its format is XML based, it is easily transformed into other formats using e.g. an XSLT style sheets. The design of MathDox is highly flexible due to the ability to embed external applications by means of web-service calls. Mathematical symbols are rendered using a JavaScript library that has multiple approaches based on the capabilities of the browser and system being used, guaranteeing an optimal rendering in any environment. Due to this, MathDox documents are easy to access and use without requiring the user to install any software other than a web browser. MathDox is fully based on open standards, which allows it to profit from proven techniques, and also allows others to benefit from the work being done by MathDox.

3 CONTEXT

Many of today's web pages are dynamic and interactive in the sense that they adapt to the needs of the user. Examples of such web pages are web shops (Amazon.com[3] and Bol.com[9] (a well known Dutch web shop) etc.), e-Banking (ING[38] , Rabobank[87], both well known Dutch banks.), auction sites (Ebay[18], Marktplaats[57] (a Dutch site), etc.), and e-learning sites (Moodle[158], Sakai[92], Blackboard[7], etc.).

These pages would be unusable if they would lack the ability of adapting the content to the needs of an individual user. Examples of adaptation in a less direct manner are advertisements. Often embedded in other pages they show the user products he or she might be interested in. Adaptation is not limited to just web pages alone, when you take a look at your mobile (smart) phone it will tell you where you are on the map, where and how far the nearest touristic hot spots are and in what direction. Instead of offering just information (a recommending system) adaptation can also take a more proactive action. For instance a car that monitors its position and speed on the road that not only warns if the car leaves the lane or comes too close to its predecessor, but also automatically hits the brakes when the driver does not react.

Adaption always needs information about the user — the users' model or context — in order to adapt the content in a meaningful way towards the user. The information needed for the users' context can be presented by the user, be readily available within the system, being based on previous visited sites, exchanged between different sites, or any combination of these.

Many adaptive web pages that are aimed at e-learning take information about the user or information that might interest the user into account when the content is generated. By doing this they capture the user's attention, an important requirement if the user is to learn something. Sierra and Bates describe it as follows in [194].

> *"So what does your brain do with all the routine, ordinary, normal things you encounter? Everything it can to stop them from interfering with the brain's real job —recording things that matter. It doesn't bother saving the boring things; they never make it past the 'this is obviously not important' filter."*
> *—Sierra and Bates*

The success of an adaptive system depends on knowing the right data about the user; this data set is called the context. Before an author starts creating an adaptive document, it is important he or she questions him or herself what can and needs to be adapted, and which data is required and how.

An adaptive document will not be able to assist a user if the user is not sincere. Adaptive systems only benefit from context if the data in the context is reliable and for this user cooperation is paramount. Users who try to convince an adaptive system of their knowledge without actually having that knowledge will put any adaptive system on a disadvantage over a normal system. It is therefore important not to see adaptivity as a tool to force the user to follow a predefined path, but to see adaptivity as an opportunity to adapt content towards interests and skills of the user, or spend extra attention to weak spots. An adaptive system has to cooperate with the user, it should not lead but assist. Failure to do so, will lead to situations in which users become annoyed and start to try to circumvent the system. This will trigger a battle between the user and the system. The system will fight to keep the data accurate while the user fights to tweak the system in such a way to get what he or she wants from it. Instead of losing energy in this struggle, better results are gotten by cooperation.

In this chapter we will go deeper into adaptive systems and discuss the different techniques that are used for adaptive systems. Some of these techniques will be used to form a mathematical context model that will enrich MathDox interactive mathematical documents with the notion of context. We propose a generic adaptive mathematical model that will facilitate authors to create context enriched documents. Such a generic adaptive mathematical model is suitable for different kinds of documents, and each such document is free in how much of the model is used. Without a doubt there will be documents among these that are aimed at education. However, non-educational documents should be possible.

We start with a discussion about the context of a user, followed by a closer look at adaptivity in general, including adaptive applications. The chapter is then concluded by an outline of our views for a generic adaptive mathematical system.

3.1 ADAPTIVE SYSTEMS

Adaptive systems find their roots in *personalization* of content. Personalization in the current web 2.0 days of the internet can be found almost everywhere. Examples are web shops such as Amazon [3] and Ebay [18], search engines like Google [33], Electronic Learning Environments (ELE) like Blackboard [7], Moodle [158, 71] and Sakai [92] and the various blog systems such as YouTube [120], Flicker [25], and Blogger [8].

Although the personalization of the internet might be seen as a recent development, in fact the techniques that allow for personalization are not that recent at all. The Common Gateway Interface

(CGI) standard was already developed in 1993, allowing websites to respond to the client web browser's requests with dynamically constructed responses based on information from the query string. This worked by executing a server located program and translating the resulting output to HTML. CGI can be used in combination with languages like Perl [83], Python [86], and Ruby [89], but CGI has some drawbacks[1]. For example it is slow and requires a lot of resources. Its successors like PHP [84], ASP [6], and JSP/Servlets [43, 44] offered the same possibilities with less resources required from the server and are widely used today. PHP in particular is very popular for dynamic websites. MathDox as discussed in Chapter 2 uses Jelly, an open source XML variant of JSP.

Adaptive systems need more.
A dynamic website cannot be considered an adaptive system unless it also has a notion of a *user profile*, *user model* or *context*, which is used to adapt content and store data about the user for the next visit. The definition of adaptive systems that Brusilovsky gives in [138] is the following:

> *"By adaptive hypermedia systems we mean all hypertext and hypermedia systems which reflect some features of the user in the user model and apply this model to adapt various visible aspects of the system to the user."*
> — *Brusilovsky*

Again here it is the system that takes information about the user and adapts the content in a way that fits the user better.

Definitions.
Although the names *user profile*, *user model*, and *context* usually indicate the same thing –the part of the software responsible for storing data about the user– their implementations and definitions might differ in different applications. A user model describes certain aspects of a user in terms of data. Because there are different perspectives, there will be different user models. We consider a *user profile* to be a special case of a *user model* concerned with storing personal data about the user, such as age, location, profession, etc. Due to the nature of this data a *user profile* will not need many updates of its data, if any.

Context, as used in this thesis, will be a set of *user models* containing at least one model but potentially more. Together these user models represent the circumstances that capture the context of a user.

Several user models are described in the literature [186, 139, 155]. Among these there are the scalar model, stereotype model, structural

1 The execution of a command required a separate process including all the overhead that comes with a newly created process, such as memory usage and delays in execution. Overhead costs are out of proportion when compared to the relatively simple commands

model, overlay model, bug model and plan model. We will briefly discuss some of these with their characteristics.

3.1.1 *Scalar model*

The *scalar model* [139, 130] requires users to provide an indication of how advanced they are in the domain that is treated by the adaptive system. If the user indicates that he or she is a beginner in the field, then the system is able to present content in a more basic manner than what would have been served to a user at an intermediate or advanced level. For instance a user at expert level in a certain topic of mathematics might be served with complicated proofs and other backgrounds that would not go well with a beginner, a user at beginner level might be served an intuitive explanation instead. The level indication of the user is applied to all content available in the adaptive system. The scalar model has an averaging effect, meaning that it ignores whether or not the user understands some aspects really well, while the same user has no grasp at all in relation to other aspects.

3.1.2 *Stereotype model*

The *stereotype model* [139] represents a list of characteristics of the user. Based on these characteristics the user is assigned a level much like the *scalar model* which is used to determine what and how to present content to the user. The stereotype model comes either as fixed or flexible. Fixed means that the system determines just once what the level of the user is. Flexible means that the system has been equipped with some rules that can either promote or demote the level and with that the level of content offered to the user.

3.1.3 *Structural model*

Where the stereotype model and the scalar model affect the whole adaptive system with their settings, the *structural model* [138] aims at dividing a system into several parts and storing information about those parts separately. Such an approach would address the averaging effect mentioned of the scalar model in Section 3.1.1. A typical structural model approach would be the overlay model which we will discuss next in Section 3.1.4.

3.1.4 *Overlay model*

The *overlay user model* [190] is the most popular user model used in adaptive systems and especially in *Intelligent Tutoring Systems* [167]; a subclass of adaptive systems specialized in education. The overlay model focuses on the knowledge the user — typically a student — possesses with regard to the knowledge available in the document. Based on this data an adaptive system can adapt its content to the skills of the user.

Adaptive hypermedia systems.
The document knowledge is modeled by a set of vertices in which related items of information or knowledge are connected by edges, forming a graph of knowledge. In this graph the nodes represent content and the edges represent the relations between the various pieces of content. A personalized graph can be obtained by combining both the graph that describes the document knowledge and the knowledge data that is stored in the user model. This personalized graph describes what a user understands, and what new information items may be in reach or are of interest to the user. In the literature [137, 187] there are different names for these items of information. They are called knowledge items, knowledge elements, learning objectives or, most commonly, concepts.

The overlay user model provides two types of adaptations that can take place on content in an adaptive system. *Content-level adaption* focuses on changes in the content directly related to information items, in such a way that the content is easier to understand for the user. These adaptations can include modifications to comply with serving various amount of detail in the content based on the knowledge of the users, or changing the notation to the recorded user preferences.

Navigational-level adaption is meant to make recommendations for further reading. As such it is typically used in hiding, showing, enabling, and disabling links that a reader can use to navigate though the document. Based on the information known about the reader, a system can use this mechanism to guide the reader to appropriate sections that are needed to understand the learning goal of the user. Especially when a user does not subscribe to a predefined sequence in which to visit the concepts and learn, showing the user the next *logical* learning step at the right moment can be quite useful. Note that showing a possible place to continue reading is not the same as forcing the reader to take that path. Users are better motivated if they can choose their own path and adapt their learning paths to what best suits them.

Binary versus incremental.
Traditional overlay models are binary in regard to whether a user understands a specific topic. Either the user does or does not understand. While this makes it easy for an adaptive system to use the user knowledge data, it does not necessary reflect the situation of the user. It does not address how well a user understands the topic at hand. The user might have only a vague idea of the content of a page after reading through it just once, or worse still, the user just clicked on the

link to the page without reading it at all. Yet in both cases the system would assume full knowledge of the topic. The binary knowledge also does not address the situation in which a user forgets what was learned. A user then becomes confused and loses interest in what an adaptive system is trying to teach or tell if the offered content does not match the assumed knowledge the system has about that user. This situation has a high degree of occurrence in any adaptive system that relies on the accuracy of a binary knowledge overlay model.

A approach to circumvent this problem would be a system that can assign a grade to a concept indicating the quality about the gained knowledge of the user in an incremental way. If so needed these classifications could be expanded to include a higher discriminating level. An adaptive system that uses an overlay model with an incremental approach can now judge whether users meet a certain threshold about a specific information element. For instance whether the user is at beginner, intermediate or at expert level. Users who just read a page once are only awarded a beginner mark. When an adaptive system discovers that a user has lost some of his or her knowledge level — or never really gained it in cases of clicking but not reading topics — it can then decide to reduce his or her knowledge grade to reflect a more appropriate level. Note that in this thesis we will limit ourselves to the architecture that enables adaptive mathematical documents. We will not indicate how such an adaptive mathematical system should decide or observe if a user is gaining or losing knowledge. Instead we aim to offer the means for such observations and subsequent decisions.

3.1.5 *Bug model*

Brusilovsky and Millán [139] state that the overlay model is considered too simple to compare it to the real world, where the knowledge possessed by users rarely matches an exact subset of the knowledge as defined in document knowledge. Learning in practice is a gradual process from generalized to refined knowledge, and includes misconceptions. When the knowledge graph is very fine-tuned, it will lead to better matching of the user with subsets of knowledge. However, user misconceptions are near impossible to prevent. It is therefore a wise idea to try and model those misconceptions and act upon those in the adaptive content. These misconceptions are also known as *buggy knowledge*; hence the name of the *bug model*. A good bug model is very hard to develop and, according to Brusilovsky and Millán, very limited in practical use since they are only suitable for very simple domains and in small intelligent tutor system problems. The limited usability makes sense if one considers the trouble it takes to envision as many error conditions beforehand as possible, particular for larger domains or more complex problems. A known adaptive system that makes use of the bug model is WITS [200]. The exercise graphs (see Section 2.5.2) as occurring in MathDox and ActiveMath [181, 1] and as described in Section 3.2.2.6 are also based on the bug model. As they too try to find (common) mistakes made by users and aim to repair these misconceptions.

3.1.6 *Plan model*

Imagine an adaptive system that stores user actions and compares these with a set of predefined *plans*. Whenever the expected actions of one of these plans matches those of the user, the system will have recognized the assumed goal of the user and from that point can assist the user in achieving the assumed goal. Such a model is called the *plan model* [172, 188]. Just like the *bug model* it is hard to implement due to the wide variety of possible user actions and non plan-related deviations from those user plans. It also neglects the possibility of a user changing his mind, or being without a concrete plan whatsoever. These drawbacks make the *plan model* only suitable for a small group of adaptive systems, mainly those in which user actions are limited and easy to categorize.

3.2 adaptive applications, an overview

Several adaptive applications exist that make use of the user models as described in the previous Section 3.1. These can be categorized on the basis of their purpose. We first discuss these categories in Section 3.2.1 and then present some example adaptive systems in Section 3.2.2.

3.2.1 *Categories of adaptive systems*

Adaptive systems can be divided into a number of different subclasses. We will describe some, starting in general and moving closer to adaptive systems aimed at educational purposes.

adaptive hypermedia systems
: The most commonly known name for adaptive systems must be *AHS*, which stands for Adaptive Hypermedia System. Hypermedia in this sense refers to the fact that –although quite common– adaptive systems are not necessarily always accessed via the web or by web browsers. The class of *Adaptive Hypermedia Systems* encloses almost all adaptive systems available.

adaptive educational (hypermedia) systems
: Abbreviated to AEHS, the word *education* in *Adaptive Educational (Hypermedia) Systems* says it all. This class of adaptive systems aims to teach the user something and is therefore often used

for educational purposes. Adaptive hypermedia systems that reason and adapt to an individual user situation by both changing content and navigation are very suitable in capturing the reader's attention and motivation, and therefore provide additional value as compared to a more traditional textbook.

INTELLIGENT TUTOR SYSTEMS
ITSes, are systems that aim to help a user to master a certain task. As such they are a subclass of Adaptive Educational Hypermedia Systems.

RECOMMENDER AND INFORMATION RETRIEVAL SYSTEMS
Some *adaptive systems* act as recommender systems that try to advise users about other points that might be interesting e.g. the well-known phrase: *"Other customers also bought"* on web shops like Amazon [3] and Ebay [18]. But applications have a wider range than just web shops. For instance there also exist tourist guide applications like [144, 143] that offer information of interest to users whenever they happen to be in the neighborhood of an interesting location in a city. Of course what is considered interesting varies from user to user. Another example would be the well-known search engine Google [33]. Google is known for storing data received from interaction with its services, to fine-tune its advertisements and search results to the interests of its users. The benefit of this fine-tuning is that results of a Google search of the web is more likely to yield content the user was actually looking for, resulting in higher customer satisfaction. Google advertisements work in the same way. The same principle is implemented at various sites such as YouTube [120] which recommends videos to watch based on what you have already seen. Pandora/last.fm is an online radio station that selects music based on your previous selections.

3.2.2 *Examples of adaptive systems*

Having discussed different types of adaptive systems, it is time to take a closer look at a few examples.

3.2.2.1 *ELM-ART*

Weber has created an intelligent tutoring system called ELM-ART [20]. His adaptive system teaches the programming language Lisp to its users. It makes use of a structural stereotype model. As such it asks users about their knowledge in web browsers, programming languages and computer experience. Based on the answers to these questions the system categorizes the user and adapts the content towards that category.

3.2.2.2 *The Aria Photo agent*

A second example of an adaptive system is a recommender system that connects to email messages being typed. When the user writes a message, the Aria system [177] uses keywords from that message to select pictures that might be relevant to that message. For instance, when in a message the writer speaks about a wedding and later on –unrelated– mentions a few names, Aria will use that information to search pictures of a wedding which involves the people mentioned. Having found such pictures it will show them to the user and make them ready for attachment to the email message.

3.2.2.3 *GRAPPLE*

A more generic approach than the previous examples is AHA! [132, 134]. AHA! inspired LAOS [152] and MOT [166], and recently GALE, which is part of the GRAPPLE project [135].

The European Generic Responsive Adaptive Personalized Learning Environment (GRAPPLE) project [135] ran from 2007 till 2011. Its purpose is to create a multipurpose general adaptive system aimed at lifelong learning. GRAPPLE recognizes that people have multiple sources of education throughout their lives, being schools, universities, work or other sources. To facilitate livelong learning GRAPPLE offers a Generic User Model Framework (GUMF) that allows distributed storage for sharing of user information and offers adaptive documents information about the user and his or her experiences from the past. This reduces the need of questionnaires asking for the same information again. Another feature is the ability to retrieve information about the user from the web, for instance Facebook. Data stored in GUMF is in serialized Java objects along with a numerical and string values so as to assist in queries. There are two kinds of data stored, knowledge data as used in an overlay model and more general data about the user. The overlay model not only supports boolean values but also gradual values.

The data stored in GUMF is used by the GRAPPLE Adaptive Learning Environment (GALE), which delivers the adaptive materials to the user. GALE works with content resources written in either XML or HTML (HTML will be internally translated into XHTML). These resources are then transformed by processors into content as offered to the user on a page. The format also allows Java expressions embedded into XML tags.

The design of GALE allows for extensions so as customize the adaptivity to a more specific field. Options for extendability include:

- Writing and adding of new Java classes so as to add alternative reasoning to the Java expressions.
- Usage of new tags and the addition of code for these tags into GALE.

- Addition of new processors into GALE so as to cope with other XML subformats.
- Forwarding of page requests to other services outside of GALE.
- Changes to appearances of pages to allow for different look and feel.

To assist authors in creating an adaptive document GRAPPLE has included the GRAPPLE Authoring Tool (GAT). With GAT teachers have a graphical tool that helps them to create a conceptual relationship and link concepts and resources, forming the domain model. The usage of a predefined set of templates reduces the learning curve of a new author.

The different parts of GRAPPLE are connected to each other by the GRAPPLE Event Bus (GEB). Which functions as a message board to which different modules of GRAPPLE can subscribe.

GRAPPLE works closely with LMSes as Moodle [158, 71], Sakai [92] and Claroline [176]. These systems provide the environment in which adaptive documents are offered to users and are already present at schools and institutes. While GRAPPLE focuses on the content the LMSes focus on the need of communication in relation to the teaching materials, such as grade books, forums, and teacher contact.

3.2.2.4 *Panta Rhei*

Panta Rhei [185, 183, 184] is an extension of the OMDoc format (see Section 2.2.4.1). Content written in OMDoc is used to generate (static) documents by means of a transformation process. These documents are then often used for teaching purposes at the Jacobs University, a university aimed at international students. These days a lot of universities get more and more people with different backgrounds in terms of the educational road they followed. Especially if these educational roads have led people through different countries or cultures, they may have been educated by different *Community of Practice* (CoP) and are therefore used to different notation. When these educational roads converge and the different CoPs meet at the same mathematical document, problems will arise. Forcing readers to adapt to the notation used in the document, will introduce the students to confusion instead of knowledge. Panta Rhei is an addition to OMDoc that allows the OMDoc documents to be translated into different versions. Offering documents in different natural languages, notations, explanations in relation to the background of the user. Panta Rhei also supports a recommendation system for supplementary materials.

3.2.2.5 *Math-Bridge*

Math-Bridge [122] was an European project that ran from 2009 to the end of 2011. The Math-Bridge projects goal is to create a platform that assists students in their efforts to repair weaknesses in mathematical knowledge. Especially students leaving high school and about to start their first year of college experience a gap in their knowledge. This gap is seen as an important cause for dropouts in technical studies. The Math-Bridge project tries to remedy this by offering students content in multiple languages, adjustment of notations to the users cultural background and by refreshing or repairing any missing mathematical knowledge. Students are offered predefined courses but they may also search for other materials of interest. Math-Bridge is build on top of ActiveMath [182, 181].

3.2.2.6 *ActiveMath*

ActiveMath [182, 181] — developed by DFKI and Saarland University — is an intelligent tutor system and as such it is aimed at teaching math to its users. We discussed ActiveMath before in Section 2.2.4.2, where we addressed the mathematical aspects, but did not go into details of the adaptivity that is expected of an intelligent tutor system. Here we focus on the adaptive aspects of ActiveMath. The adaptivity in ActiveMath is found in both the exercises offered and the courses (documents) that are generated.

The presentation system compiles and transforms content into courses. To do this right however it requires the services of the rule engine PAIGOS. This is a web service of ActiveMath that accepts the input of a context and a task to create a course for a user. In its process of selecting the right material it utilizes a set of 300 rules. The context in this sense is a set of applicable rules concerning the topic at hand e.g. arithmetic, differentiation, logic, etc. The tasks that are accepted by PAIGOS will tell what kind of course to generate, again with user data in mind such as learning goals. Tasks can be any of the following;

- Discover, Discover and understand concepts in depth.
- Rehearse, Address weak points.
- TrainSet, Increase mastery of a set of concepts by training.
- GuidedTour, Detailed information, including prerequisites.
- TrainWithSingleExercise, Increase mastery using a single exercise.
- Illustrate, Improve understanding by a sequence of examples.
- IllustrateWithSingleExample, Improve understanding using a single example.

Decisions on what to include or exclude, are also based on the user data. The user data is gotten from the user by questionnaires, but also by storing results from exercises. A correctly answered exercise tells the system not only something about the understanding of the current mathematical concept or notion, but also about the underlying concepts. Vice versa is also true, answering an exercise wrong, makes the user not qualified to proceed to a more advanced topic that depends on the current notion. As it is hard to analyze and determine with certainty why a user did something correct or incorrect, the recording of these events is accompanied by a probability. All relevant events combined result in an aggregated probability on whether the user understands a certain concept or not. The generated courses are also adapted towards the users preferences, i.e. the requested language.

Although the overlay model receives input, this does not change the content in the generated course for the user, unless the users asks for a new version.

Adaptivity in exercises does not just focus on logging probability events. The answers a user supplies to questions are analyzed and if they are incorrect an attempt is made to recognize where the error was made and direct feedback is given to the user to make him or her understand what went wrong. This is an implementation of the bug model as discussed in Section 3.1.5. Also the knowledge of the user is taken into account when the answer of the user is analyzed. For example, the rules stored in the context of the user will tell the diagnosis service if the user, in case of differentiation of $f(x) = (x + 1) \cdot x$, is familiar with the product rule, or whether the user is expected to first expand the product to $f(x) = x^2 + x$ before starting the process of differentiating.

3.3 CONTEXT OF A USER

We discussed adaptive systems and their user models in Sections 3.1 and 3.2. Now it is time to discuss the more specific needs of a context for an all-round adaptive mathematical system. What does such a context need?

Before that question can be answered we first need to ask another question: what is it that we are trying to achieve with adaptive mathematical documents? This second question is not easy to answer; after all, we aim to build an all-round system authors can use to build their own adaptive mathematical system with. Therefore we need to facilitate as many different kinds of adaptive mathematical documents as possible.

At a closer look similarities can be found between adaptive applications. In Figure 7 the flow of information through a (mathematical) adaptive system is shown.

This information flow starts with the author of an adaptive document who wants to put his or her information and knowledge into

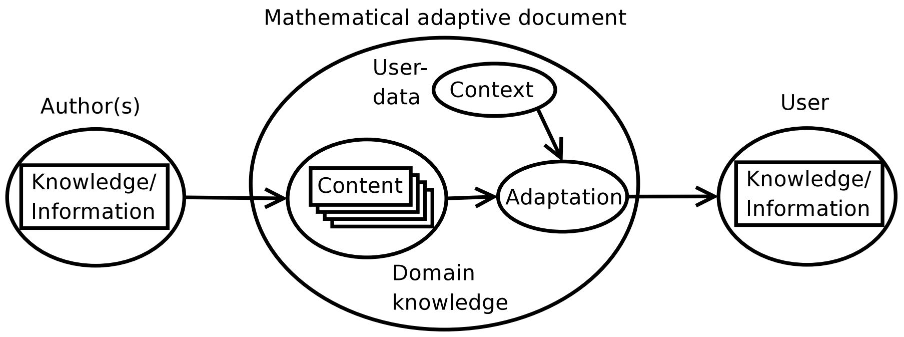


Figure 7: The flow of knowledge/information from an author, through a mathematical adaptive system, to the reader of the document.

a document. This will then be stored as content and such is part of the domain model. We will see later –in Section 4.1– that the domain model will contain more than just content.

Observe that the content is fragmented rather than being one big lump of knowledge or information. This fragmentation is required to be able to feed the user manageable pieces of information. However, in most cases this fragmentation goes further than mere fragmentation into chapters or sections, because smaller fragmentation allows alternative composition of pages that are adapted to the user. The adaptation process happens with the use of the context, which basically is user data of relevance stored within the adaptive system. It is this context in a mathematical setting we will now examine further.

Context data can be roughly divided into three different kinds:

LOGISTICAL CONTEXT

Logistical context contains all non mathematical context data.

This sub-context can be even further divided into a static and a dynamical part. The static part only has to be filled in once and can be assumed only to change incidental afterwards. It would typically store data such as a user's name, date of birth, profession, etc. A more dynamical part will store data like last visit, exercise scores, etc. *Logistical data* is not necessarily mathematically oriented; it is however quite important for a lot of small personalization aspects that make a user feel at home and comfortable.

KNOWLEDGE CONTEXT

The primary task of knowledge context is tracking the knowledge level of a user.

Almost all educational adaptive systems aim to transmit knowledge to their users. Adaptive educational systems, such as intelligent tutoring systems, are especially focused on knowledge gain of a user. Typically such a knowledge representation would be modeled with the use of an overlay user model, as discussed in Section 3.1.4. The overlay model will keep track of each concept and the associated values which indicate the degree of mastery by the user. An adaptive mathematical document does not differ from this principle; it too needs to keep track of progress made by users.

MATHEMATICAL CONTEXT
Variables that occur in a document and are important enough to be maintained between sessions belong to the mathematical context. An interactive mathematically flavored document (which may also include related subjects as electronics and physics, etc.) differs from a normal document by having mathematical variables occurring in examples and exercises used to illustrate notions and principles. Where in a traditional textbook the values of these variables cannot be altered, in the interactive approach they can. Effectively turning a variable from just a name into content. Not only will these variables be remembered in between sessions they are also available for use elsewhere in the document. As stated before, we assume knowledge is fragmented and will be rebuilt later in possible different configurations. This can easily lead towards a mosaic look and feel for users since this content has to be rather independent in order not to conflict in different settings with other pieces of content. With the use of variables that reoccur throughout the document this mosaic effect will be less evident. But more importantly, the use of reoccurring variables will let users feel how different values affect each other as different mathematical notions come together and are combined into new concepts.

When we take another look at Figure 7 it is important to understand that this information flow has the characteristics of a chain. As a direct result the information flow only functions as required if each process is performed with attention to quality. The final result will only be as good as the worst performed process in the chain. When a process is performed badly, it will cause damage to the information flow that can not be repaired later. If the quality in the flow is properly maintained then a user is able to recompose the information offered by the adaptive mathematical system back into knowledge. Quality depends on content written and fragmented by authors but also on the use of context when the content is presented to the user. This process requires more than just reading the adapted content, it requires the user to understand it, only then will information turn into knowledge. Knowledge management is also addressed in [173, 192]. Adaptivity in this regard is both an advantage and a disadvantage. Obviously fragmenting knowledge, reordering and turning it into adaptive content has a high risk of confusing readers when it is not done correctly. Doing it in the correct way is a goal that is hard to achieve since what works with one reader, might fail and confuse another. It is here where the user context really needs to prove its worth. The data known about each user needs to be accurate before it becomes possible to adjust to the exact needs of each user. And here lies the catch. How accurate does the user context data need to be? How complex does the adaptation process need to be to fully utilize the user context data? Obviously, the increase of work related to such an elaborate set of data and the complexity that comes with it threatens the efficiency of an adaptive system. Authors cannot be asked to take into account all possible situations, and users will not appreciate the elaborate set of questionnaires.

Adaptation, performed with care for the reader and aware of the efficiency trap, still has the upper edge as opposed to the traditional unadaptive documents. Well chosen adaptations do not require investment from either author or user, while they can really go a long way to make the reader understand the information offered.

3.4 REQUIREMENTS FOR AN ADAPTIVE MATHEMATICAL SYSTEM

In this chapter we have described adaptive systems and considered the implications for a mathematical adaptive system. In particular we identified some attention points a mathematical adaptive system needs to take into account. Now we translate these into a set of requirements for our design for a mathematical adaptive system. Note that the requirements as defined in Section 2.1 in regard to mathematical formats also apply to a mathematical adaptive system; for this reason they are included into this set of requirements as well.

INTERACTIVITY
: An adaptive mathematical document needs interaction with the users. By observing the user it obtains user data required to adapt to the user's context, making the content easier to understand for the user. In a mathematical document is also is paramount that a document is able to interact to mathematical exercises and examples. That too is content that adapts to the user, and also contains user input to observe and store in context.

USABLE IN MULTIPLE FORMATS
: Adaptive documents require electronic means to display the various pages that are adapted to the needs of the user before they are accessed and read. A format meant for publishing on paper is less suited for this purpose, however the need to display pages in a different format still remains even if that means losing adaptivity.

BEING EXTENDABLE
: The usability of an adaptive mathematical document is greatly

enhanced if it allows for extensions to the format and the adaptive system. Extensions can include new adaptive behavior, or alterations of the adaptive system to a more specific mathematical subfield.

REPRESENTATION OF MATHEMATICS
: Any mathematical adaptive system of course requires the means of working with mathematical expressions in a correct manner — meaning the mathematics are required to be semantic — but it also needs to display mathematics in a way that is easily understood by the reader. The rendering of mathematics should not rely on images, hindering dynamic changes in mathematical expressions, or be limited by what a computer keyboard has to offer.

EASE OF USAGE
: A requirement that remains exactly the same regardless of interactive mathematical documents or adaptive systems, is the *ease of usage*. Any format requires ease of usage. Without it writing content becomes too difficult for authors to write, and improvements and bugs will not be made or solved. The success of a format depends for an important part on the ease in which content is being written and maintained.

USE OF OPEN STANDARDS
: By using open formats a system benefits from tried and proven technologies and the community that backs such an open format. It prevents vendor lock-in, and assists in solving problems by offering the opportunity to fall back upon the community and to use pre-existing documentation. Using software with an open source license allows for verification of correctness.

REUSABILITY OF CONTENT
: Adaptive systems that use fragments, can change page contents by changing which fragments they include. More fragments allow for more adaptability, but also increase the amount of work required to create them. The reusability of fragments by different pages or documents, however, allows for the investment of creation to be earned back.

STABILITY
: Adaptive documents that change content in between visits to their users, run the risk of confusing them because, for example, the users cannot find what they knew was there before. Therefore adaptive systems need to pay attention to what and when they adapt their content and offer their users some stability from adaptations if required.

QUALITY OF ADAPTION
: The goal of adaption is to help the user to understand what he or she is reading or to assist the user in his or her goal. Adaption that does not have the right goal in mind, may therefore confuse the user instead. An adaptive document does not need adaption for the sake of adaption, adaption needs to serve a purpose.

SHARING USER DATA
: Sharing information about a user between different adaptive documents prevents asking the same questions to the user more than once. An adaptive document therefore should be able to accept data from external sources, provided the data is of use.

MANAGEMENT OF USER DATA
: The goal of an adaptive system is to assist the user. However when misconceptions do occur in the user data, the user must be able to correct them.

PRIVACY
: Collecting user data naturally also brings the responsibility to act wisely with the information gathered about users. Nobody other than the adaptive system, the administrator or the user needs access to the data. In case of an e-learning application this list could also include teachers, but only on a need to know basis. An adaptive system needs to guard this privacy.

ROBUST
: If any errors during information gathering, or adaption do occur, the adaptive document is never allowed to enter a situation in which it cannot recover anymore. If, despite all efforts, this still happens, it may not affect any other users, or other adaptive documents being served by the same adaptive system.

3.5 THE NEED FOR CONTEXT IN MATHDOX

The MathDox system as described in Section 2.3 is well suited for presenting highly interactive mathematical pages over the web. However, when serving a document that consists of various related MathDox pages, it is desirable to add a common *context* to unite the different parts of the document, capturing the user's preferences, performance, browsing history, etc. Such a context makes it possible to guide users through the document, present them with related examples in the various parts of the document, and to offer them exercises that meet their knowledge level. We speak of uniting these pages with the means of a context, because with the help of a context all of these pages can now access and benefit from data collected by any of them.

Current state of MathDox.
Currently the MathDox Player does not possess any context. It is un-

able to maintain any knowledge about a page request as soon as the request is completed and the page has been sent to the user's browser. However, MathDox pages can react to specific situations of users. As we have seen in Section 2.5.2 a MathDox page can analyze user's answers and use the combination of the current page and the user's results to direct the user to a specific next page. Even though no information is stored about the user one can consider this a state and transition system and view it as a local context. As soon as the user decides to visit a different MathDox page as presented to him or her this local context is lost and as such it has limited usefulness.

Some context can be added to MathDox by the use of Learning Management Systems (LMS). By creating and placing a SCORM [95] package with MathDox content in an LMS such as Blackboard [7], Moodle [71] or Sakai [92], the task of keeping track of a particular user can be outsourced to these LMSes. However the data stored by these LMSes tends to be too general and too limited. They will keep track of things like login times, exercise completion and scores, they also might allow some customization. But they are not meant to adapt content, and therefore lack the opportunity to interpret the data they collect or to use it for adaptation of content. The possibilities of a context and an adaptive document go beyond those of an LMS.

By adding context to the MathDox Player we aim to develop a system that extends interactive mathematical documents with adaptive qualities. An adaptive mathematical document is considered to be a collection of mathematical pages containing theorems, proofs, examples, exercises, and the like, which undergo adaptations in content and presentation to better suit the user. As is common in adaptive systems, these pages do not need a strict hierarchical structure of chapters, sections, subsections, and paragraphs. Indeed, users may take their own (guided) paths through the pages. Depending on their interest, skills or tasks, these paths may vary. However not only the path that a user is following may depend on his or her personal context, also the contents of each page may dynamically be adapted. Users can choose a favorite running example throughout the different pages of a document, or while the students of a single class are studying the same pages of a course, each can be offered personalized exercises, matching their individual levels.

What is needed is a means to remember users, remember their skills and knowledge, and remember their set variables as used in the document, so that this information can be reused on other pages as well as in later sessions. This is the task the context is going to perform for adaptive mathematical documents to be written by authors.

It is good to realize that the amount of context that is required by an adaptive mathematical document varies between different documents. It also varies which data needs to be monitored and reacted upon.

Adaptive mathematical systems.
Most systems we mentioned in Section 3.2 are systems that are not suitable for mathematical contents. They lack the necessities to support such content. These necessities include an unambiguous format to reason with mathematical expressions, a connection to a computer algebra system or other means to perform exact calculations, and a way to interactively present the mathematics to a user and receive a mathematical response from this user. Panta Rhei [184], described in Section 3.2.2.4 is a mathematical system specialized in adapting notations in mathematical texts. ActiveMath [182], mentioned in Section 2.2.4.2 utilizes a reasoning engine to compose documents towards users on a stereotype basis.

User models and context.
In adaptive systems the user model takes care of maintaining data about the individual user. Some user models were discussed in Section 3.1. These user models are supported in these tasks by domain data. Domain data and user models together enable adaptive systems to adapt the content and to personalize it for each user. The term user model is somewhat ambiguous since adaptive systems can combine different user models and have them cooperate with each other in their job.

All-round adaptive model for mathematics.
We want to enable MathDox to maintain a context and to be able to react on the data stored in that context. By this approach we aim to give authors the means to create MathDox documents that can adapt towards the needs of their readers. As we have seen there already exist a multitude of adaptive systems with a wide variety of purposes and user models. Our goal is to turn MathDox into a mathematical adaptive system that allows authors to create different kinds of interactive mathematical documents with a context that made to fit each document. This means that an author is allowed to omit parts if he or she feels that it is not needed for the interactive mathematical document the author is creating.

By taking an all-round approach to bringing adaptivity to MathDox we prevent MathDox from over-specializing in any user model in particular. Over-specialization might interfere with other types of (non standard) user models and should therefore be avoided. However, this means that if an author wants to implement an adaptive interactive mathematical document, he or she also might need to implement the user model that is required for that particular interactive mathematical document. While the nature of the MathDox format already requires authors to be acquainted with light programming tasks, we do aim to reduce the amount of work required. For this purpose we will equip MathDox with a standard but optional user model. The work required to implement a new user model can be further reduced by supplying the author with a set of useful tools. These tools will enable the author to achieve his or her goal without too much effort; see Chapter 5 for more information about this tool set.

The standard user model we implement is the overlay user model (see Sections 3.1.4 and 4.1). The overlay user model is by far the most popular model used in existing adaptive applications and we predict that will also be true for most of the future adaptive applications in MathDox as well. Also the existence of the aforementioned toolset requires a basic architecture that can be obtained by the implementation of the overlay model. This default implementation of the overlay model will not interfere with other implementations and can be ignored if desired by an author.

Our proposed context enrichment of MathDox should enable an author of an interactive mathematical document to cope with differences between users. These differences can include the following:

DIFFERENCES IN NOTATION
: Mathematicians all over the world have adopted different ways to write down mathematical formulae. For example an open interval can be written in various ways. The notation that is widely used internationally is (a, b), while this is often written as $]a, b[$ in Dutch and French speaking countries. Also $\langle a, b\rangle$ is not uncommon. Despite the differences in notation all three versions mean the same. The preference of the user to which notation should be used is again stored in the user model. The presentation layer will use this data to adapt the content accordingly. Another example is the use of the letter i to indicate complex numbers where electrical engineers and physicists use the letter j instead in order not to confuse the complex numbers with amperes that are also represented by the letter i.

DIFFERENCES IN KNOWLEDGE
: Each user has a skill level that is determined by education, profession, and the sequence in which he or she is reading through study material. With help of the user model presentation choices can be made that result in the creation of a personalized page for each user. The information shown on such a page is therefore based on the knowledge of the user and his or her needs. For instance a user needs to understand the principles of a permutation group and even a symmetric group before reading the notion of orbit or stabilizer. If the user does not yet possess this knowledge a page about the notion of orbits will have to include the basics of a permutation and symmetric group also. We will revisit this particular example later on in this thesis and expand up on it.

DIFFERENCE IN SYMBOLS

A user will be unable to understand a new notion or symbol which is based on other symbols which are not yet understood. In such a case it is required to start with this missing knowledge before a system can continue with the knowledge at hand.

THE NEED FOR MORE ELABORATE EXPLANATION

Texts as they appear in mathematical documents are not always equally understandable for everybody, even if users do possess the required knowledge and therefore should be able to understand. This difference is explained by differences in reading or learning techniques, or in how well somebody is capable of understanding mathematics in general. Some users are capable of recognizing the used formulae in computations and proofs without the need for much explanation while others have more trouble. This is typically something that does not only appear in mathematical textbooks but could also happen in articles. By adapting the presentation of these computations, those users who are in need for more explanation can be satisfied. The inclusion of more examples demonstrating the mathematics at work also assists in a better understanding.

SUMMARY

There is a gray area between a website with dynamic content and an adaptive system; not every dynamic website is an adaptive system. An adaptive system needs means to adapt content to and store data about the user. An adaptive system takes information about a user and adapts content in a way that fits that user better. The kept information about the user is called *context*, and is a set of *user models*, which represent the circumstances that capture the context of users in relation to — in our case — interactive mathematical documents. Several types of user models exist, ranging from simple beginner-novice-expert gradation to the modeling of knowledge of a user based on previous actions.

An interactive mathematical document is a collection of mathematical pages which do not need a strict hierarchical structure of chapters, sections, etc. The path through the document depends on the interests, skills or tasks as stored in the context of the user. Also the difference in notation, knowledge, symbols and formulae depends on the user and is stored in the context.

MathDox needs a context that allows documents to be adapted to the situation of its users. Authors are enabled to write documents that benefit from the requirements listed earlier (see Section 3.4), or a subset thereof depending on the needs of the document and targeted audience.

4 A MATHEMATICAL CONTEXT MODEL

In this chapter we discuss in detail the mathematical context model that was mentioned in Chapter 3, which will be utilized in our context implementation in the MathDox system. The technical implementation of the model will be discussed in the next chapter, Chapter 5.

Generally speaking, an adaptive system that uses the overlay user model — as discussed in Section 3.1.4 — can be divided into three parts. The first is a static part that contains the overlay model graph and the unadapted content. This part is called the *domain model*. The second, the *user model*, keeps track of all known data of importance about the user. Finally, through the *presentation model*, all information is combined and the adapted page is presented to the user.

The names of these models are not always the same in the literature. The *domain model* is also called the *document model* or *doc model* for short, while the *presentation model* is sometimes also called the *adaptive model*. The *user model* is consistently called the *user model*. Here we will use the names *domain model*, *user model*, and *presentation model*.

Adaptive systems need an extra part to function. While its functionality is quite clear, it is not always considered to be a separate part of an adaptive system, but rather integrated with other parts, it is referred to as the *observer model*. The *observer model* is responsible for making assumptions about the user's state. It *observes* the user and feeds the conclusions of these observations into the *user model*.

Based on these components Dolog, Henze, Nejdl and Sintek [157] use the quadruple *(DOCS, UM, OBS, AC)* to define an adaptive hypermedia system. *DOCS* is short for the *document model* or, as we call it, the *domain model*. *UM* stands for *user model*, *OBS* for the *observer model* and finally *AC* is for the adaption component, which is called the *presentation model* in this thesis. Those familiar with computer science and design patterns [163, 162] might notice the similarity of the setup of adaptive systems and the *Model, View, Controller (MVC)* pattern [1] [141].

A mathematical context model for interactive mathematical documents.
We have adopted the *domain, user* and *presentation models* for the realization of adaptive mathematical documents in our MathDox system.

Our *domain model* contains information on the knowledge domain of an adaptive mathematical document. It is an abstract model for

1 The *Model* in the MVC design pattern can be compared with the *user model*, where the *View* matches with the *Presentation model* and the *Controller* is the counterpart of the *Observer model*. In this comparison, the *Domain model* is not present, however in the *MVC* pattern the content is considered to be implicitly available.

the main concepts of an adaptive mathematical document, such as symbols, definitions, theorems, proofs, etc., and their relations within the document. The items in this model refer to collections of MathDox sources.

The *user model* is a model for the data pertaining to a user of the system. The model captures, for example, the logistic information of the user, such as identity, profession, knowledge level, but also information on the history of the user, such as acquired knowledge, navigation history, and scores on exercises.

Finally, the *presentation model* takes responsibility for displaying the content to the users. With input from the user model it selects and transforms content from the domain model into a page rendered for the user's needs.

In our setup the *observer model* is encoded in the content itself and as such it is a part of the presentation model. This makes sense since the content is also responsible for monitoring and analyzing the user. The content is best suited for analyzing the user reactions and drawing conclusions about user knowledge and data.

Mathematical Context Model.
Mathematical content has different characteristics from non-mathematical content normally used in adaptive systems. As mentioned in Chapter 3, mathematical content makes use of variables and computations as opposed to non-mathematical content. Such content therefore needs a specialized user model. This model has to take into account that mathematics is very hierarchical by nature. The overlay model (see Section 3.1.4) is therefore very appropriate as a basis for the mathematical context model, especially since its dependency graph is well suited to represent the hierarchical structure of mathematics.

There exists more than just one hierarchical structure in mathematical content. For this reason we have chosen to include not one dependency graph, but three graphs. One graph will represent knowledge of theories (called the *theory graph*, see Section 4.1.1); another will represent symbols occurring in the document (called the *symbol graph*, see Section 4.1.2) and a third will represent the variables in the document (called the *variable graph*, see Section 4.1.3). In our model we select the theory graph to be our main dependency graph, in compliance with the overlay model. These graphs, together with the relations between them, form the domain model of the mathematical context model.

The graphs we have just described also have an impact on the user model. As we will see in Sections 4.1 and 4.2 these graphs are static and the user model will store data concerning to the user and these graphs. However, the user model can also record other data that might affect how content is presented to the user.

The presentation model is then responsible for combining the content and knowledge, as stored inside the graphs, with the data stored in the user model to form a presentation of the content that fits the user best.

In the following sections we give a more precise description of the domain model, user model, and presentation model. Together these will form the mathematical context model which is required for providing context to adaptive mathematical documents. Some of this work has also been discussed in [149].

4.1 DOMAIN MODEL

A document contains a domain model, which is static in nature. The domain model is concerned with the topic of the document, as it contains the content and structure of the document. At the core of the domain model is the *theory graph*. Alongside the theory graph are the *symbol graph* and the *variable graph*. The main ingredients of these three graphs are the mathematical objects, notions, statements, symbols and mathematical expressions as used within the interactive mathematical document, as well as their interdependence. The purpose of these graphs is to structure the mathematical knowledge on which an adaptive mathematical document is based.

We discuss each of these three graphs and conclude this section with an analysis of their mutual relations and soundness conditions that validates the correctness of an adaptive mathematical document. While the content is also a part of the domain model, we will not address the specifics of the content of an adaptive mathematical document until Chapter 5.

Hasse diagram.
The theory, symbol and variable graphs will be defined as graphs that are Hasse diagrams of a partial ordering. We first introduce the necessary mathematics. A *partially ordered set*, also known as a *poset*, is a pair $(X, \leqslant)$ consisting of a set X and a relation $\leqslant$ on X (written infix) satisfying the following three properties.

- *reflexive*: for all $x \in X$, $x \leqslant x$;
- *anti-symmetric*: for all $x, y \in X$, $x \leqslant y$ and $y \leqslant x$ implies $x = y$;
- *transitive*: for all $x, y, z \in X$, $x \leqslant y$ and $y \leqslant z$ implies $x \leqslant z$.

In general we write $x < y$ to denote $x \leqslant y$ and $x \neq y$.

For any poset $(X, \leqslant)$ one can define the *Hasse diagram* to be the directed graph with vertex set X, in which two vertices (also called nodes) x and y are connected by an edge from x to y if and only if $x < y$ and there is no element $z \in X$ with $x < z$ and $z < y$.

The Hasse diagram is by definition an acyclic graph from which the poset can be fully reconstructed, the partial ordering is the transitive and reflexive closure of the Hasse diagram.

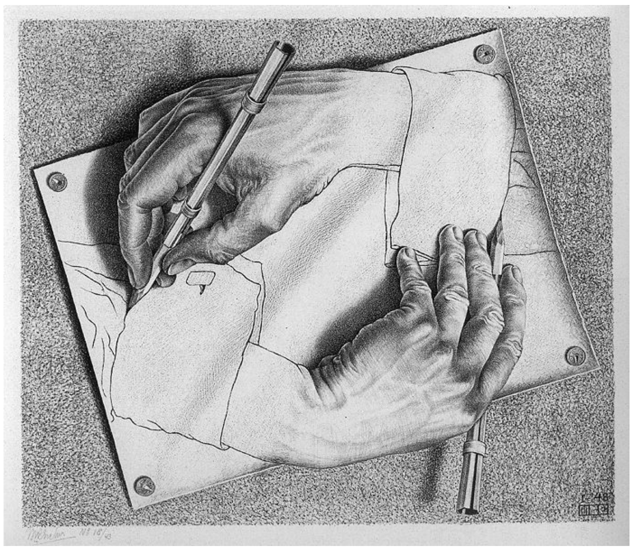


Figure 8: A drawing by Escher showing mutual dependency.

If a cycle is encountered in a dependency graph it would mean that the vertices in that cycle depend indirectly on themselves. The famous Escher picture *Drawing hands* as presented in Figure 8 shows clearly why it is impossible to have a mutual dependency between two nodes. After all, if one hand draws the other then which line was the first to be drawn? And by which hand?

Omitting the redundant edges — as shown in Figure 9 — is a very useful property of Hasse diagrams, since it makes these graphs much easier to read and understand for a human, such as an author or a reader of a document. Leaving out the redundancies also makes sense from the perspective of a user; skipping dependencies is not allowed since that would create a lack in required knowledge. Therefore a user always has to take the long path to reach a desired node. It also makes sense from a technical perspective; a lean dependency graph is easier to reason with and eliminates a lot of non-valid routes (shortcuts).

Definition of domain model.
A *domain model* as used in the mathematical context model is the quadruple

$$\left(\mathcal{T}, \mathcal{S}, \mathcal{V}, \to\right) \tag{1}$$

where $\mathcal{T}$, $\mathcal{S}$, $\mathcal{V}$ are posets defined on disjoint sets T, S, V, respectively, $\to$ is a subset of $(V \cup S) \times V \cup T$. Here, the members of T, S, and V are called *theories*, *symbols* and *variables*, and are nodes of the *theory graph*, *symbol graph*, and *variable graph*, respectively. The dependencies between the elements within each of these sets are given by the partial orderings $\sqsubseteq_T$, $\sqsubseteq_S$, $\sqsubseteq_V$ on the respective sets T, S, V. So $\mathcal{X} = (X, \sqsubseteq_X)$

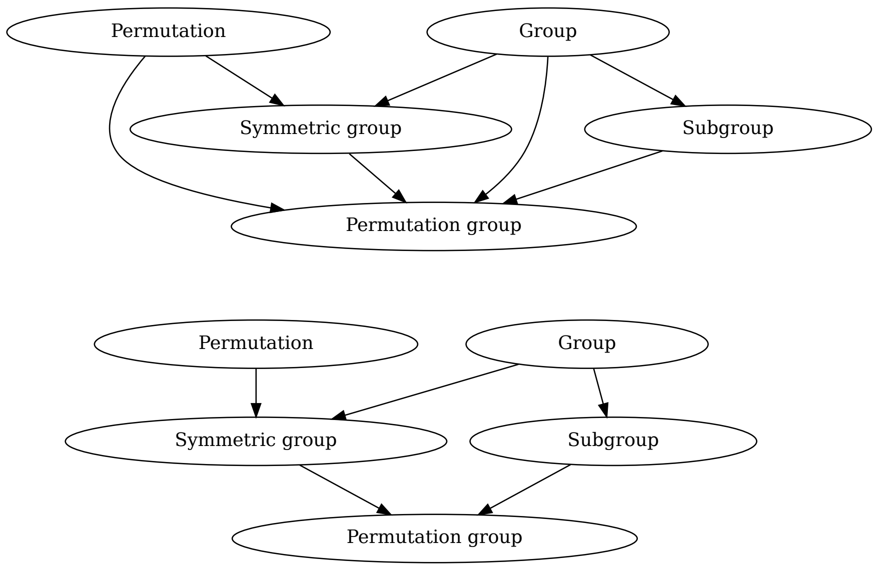


Figure 9: Omitting redundant edges leads to a Hasse diagram.

for X equal to T, S, V. Each will be explained in the following sections. The relation $\rightarrow$ indicates in which theories the variables and symbols occur, and in which variables the symbols occur. We therefore refer to it as the occurs in relation.

4.1.1 *The theory graph*

The main notion for an adaptive mathematical document is the *theory graph*. A mathematical document is a collection of statements involving mathematical symbols and variables together with a logical structure on these statements. The two main types of statements are definitions and assertions. These statements are then often enlightened by examples, exercises or remarks, or accompanied by proofs. A union of such statements hinging on one definition or assertion (possibly a compound one) comprises a *theory*, representing a node of the theory graph. For example, one may view each statement as an individual theory or, at another extreme, clumping together an entire chapter into a single theory.

The dependencies within the theory graph can be understood formally, from either the user's or the mathematical point of view.

From a user perspective, an adaptive mathematical document can adapt itself to the level of skill of the user by keeping track which theories have been visited or mastered by the user. But such a document can do even more for a user when it knows which theories are required prerequisites of the requested theory, or which theories might follow from the theory just read. By taking a look at the prerequisite theories — and determining whether these are understood by the reader — an adaptive document adapts the requested page ac-

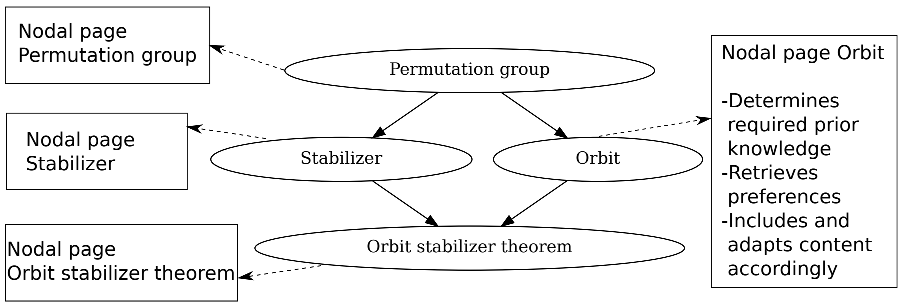


Figure 10: A theory graph with nodal pages

cording to the reader's knowledge. An adaptive document can give recommendations as to what to read next by looking at which theories follow the theory just read. Thus from the user perspective, a theory graph is a graph that represents the relation between theories.

In Figure 10, a theory graph with pages associated to each node is shown. Each of these pages handles the content indicated by the name of the node. The page about *orbit* shows how content is affected by context both in preferences and knowledge as derived from the theory graph. These pages associated with the theories are called *nodal pages*.

A mathematical perspective on the theory graph is that one piece of theory is needed to be able to formulate the next theory. For example, the notion of a permutation group invokes that of a group and a permutation. The edges of the theory graph can be described as a presentation of this relation.

Let T denote the set of all theories — or the set of nodal pages linked to these theories — of a document, and let $\sqsubseteq_T$ be the relation defined by $x \sqsubseteq_T y$ if and only if y depends on x. This defines a partial ordering on the theories and associated nodal pages in the set T.

We are now in a position to give a formal definition of our theory graph; namely, it is the Hasse diagram corresponding to the poset $(T, \sqsubseteq_T)$.

To elucidate our notions, we will work out the example of a document on permutation groups. This particular example will also reappear in later examples as we pay attention to other aspects of the mathematical context model. This running example will allow us to illustrate various aspects concerning the mathematical context model; among these aspects, the most notable are the theories, symbols and variables. By using the same context-enriched document it will become clear how these different aspects are joined together to form a powerful yet flexible document about permutations.

Example 6:

The mathematical starting point of this example document is a finite set X. A *permutation* on X is a bijective map $X \to X$. Usually, if X has size n, we take X to be $\{1, \ldots, n\}$.

A bijective map f is then defined by a list $[f_{(1)}, \dots, f_{(n)}]$ of elements f_j from X that are all distinct. This indicates that f maps the element j to f_j. The set of all permutations on X is denoted by $\mathrm{Sym}(X)$ and contains the *identity element* e, which maps x to x for each $x \in X$. It is closed under *composition* of maps: if $f, g \in \mathrm{Sym}(X)$, then $fg \in \mathrm{Sym}(X)$, where fg is the map sending $x \in X$ to $f(g(x)) \in X$. Finally, $\mathrm{Sym}(X)$ is closed under inverses: if $f \in \mathrm{Sym}(X)$, then the *inverse* map f^{-1} characterized by $ff^{-1} = f^{-1}f = e$ is also in $\mathrm{Sym}(X)$. These three properties make $\mathrm{Sym}(X)$ into a *group*. A subset G of $\mathrm{Sym}(X)$ containing e and closed under the operations of composition and taking inverse is called a *permutation group* on X. If g and h are permutations, the intersection of all permutation groups on X containing g and h is again a permutation group on X. This is called the subgroup of $\mathrm{Sym}(X)$ *generated by* g and h.

Let G be a permutation group on X. If $x \in X$, then the *orbit* of x under G is the subset

$$\{g(x) \mid g \in G\} \tag{2}$$

of X. This orbit is usually denoted Gx. If H is a subset of a permutation group G which is closed under composition, then H is called a *subgroup* of G. The *stabilizer* in G of an element $x \in X$ is a subgroup of G, defined as

$$\{g \in G \mid g(x) = x\}. \tag{3}$$

This is actually a permutation group on X and is usually denoted by G_x.

The Orbit Stabilizer Theorem states that, if X is finite, the following relation between the various cardinalities holds, for any $x \in X$:

$$|G| = |G_x| \cdot |Gx|. \tag{4}$$

Instance of the theory.
By way of example, take $X = \{1, 2, 3, 4, 5, 6\}$ and choose the permutations g and h as follows.

$$\begin{aligned} g &= (2,4)(3,5) \\ h &= (1,4,6,3)(2,5) \end{aligned}$$

Let G be the permutation group on X generated by g and h. Then G consists of 24 elements. Now, if $x = 1$, we find

$$\begin{aligned} Gx &= X \\ G_x &= \{(), (2,3)(4,5), (2,4)(3,5), (2,5)(3,4)\} \end{aligned}$$

and so the Orbit Stabilizer Theorem is confirmed, as

$$|G| = 24 = 4 \cdot 6 = |G_x| \cdot |Gx|.$$

The mathematics described so far can now be distributed into the nodes of the theory graph displayed in Figure 10.

The dependencies are dictated by the facts that the notions of *Orbit* and *Stabilizer* make no sense without the notion of *Permutation group*, and, likewise, the *Orbit Stabilizer Theorem* needs the notions of *Orbit and Stabilizer* to be formulated. It is the choice of the author to chop this into four nodes. For example another author might have created a node for the notion of the symmetric group (that is, Sym(X)).

Also visible in Figure 10 are the nodal pages linked to the theories in the theory graph. Each nodal page contains all that is needed for the proper construction of content related to that theory node. It needs to determine what content to include and how to present it. This may, for instance, lead to a page that refuses access until all previous theories have been mastered. Or to a page that includes useful definitions and examples of previous theories — not yet mastered — before presenting the theory at hand. These nodal pages decide on questions like: *How should this theory be presented? Does it need a lot of examples? Is a proof required? Is it necessary to link to an example or application that the reader is familiar with due to his/her affiliation or profession?* The decision process is part of the *presentation model* as described in section 4.3.

4.1.2 *The symbol graph*

The next graph we discuss is the *symbol graph*. The *symbol graph* is a graph whose nodes are the mathematical symbols found in mathematical objects of an adaptive document.

Mathematical objects in the MathDox sources are represented by OpenMath expressions, providing semantic encoding of notions. In our set-up, a mathematical symbol is an OpenMath symbol. These OpenMath symbols occur in OpenMath expressions as they are used

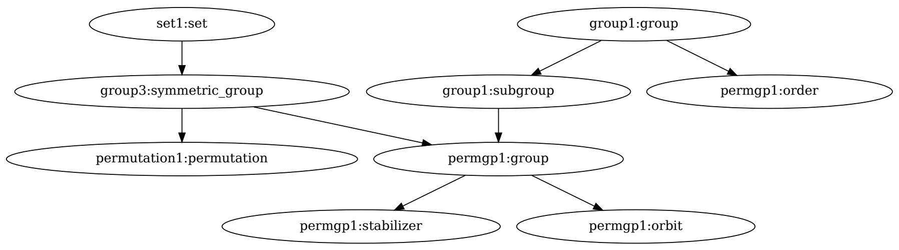


Figure 11: A symbol graph on group theory.

in the document, variables, expressions or computations. OpenMath symbols are defined in *content dictionaries*, see [77].

The content dictionaries provide some information on the relations between the various mathematical symbols. However, in many cases, this information is too limited; there is no official (partial) ordering that specifies which symbols need to be understood before another can be comprehended. Obviously an adaptive mathematical document runs into problems if it is not able to determine which mathematical symbols the user does or does not know. In such cases it is no longer capable of determining a sequence of content to present or teach to the user. Authors may have different opinions about the proper (partial) ordering of symbols. We therefore offer authors the flexibility to capture the dependencies between these symbols and store them in a dependency graph called the *symbol graph*.

Let S be a set of OpenMath symbols, and suppose $\sqsubseteq_S$ is the relation defined by $x \sqsubseteq_S y$ if and only if the mathematical notion of y depends on that of x.

As before, the relation $\sqsubseteq_S$ on the set of symbols S in a sound mathematical document is assumed to be a partial ordering. So, we can define the symbol graph to be the Hasse diagram of the partial ordering $\sqsubseteq_S$ on the set S of OpenMath symbols in the document.

The ordering $\sqsubseteq_S$ on the set of symbols S inside a MathDox document takes into account the OpenMath symbols needed for a formal definition of each element of S. Figure 11 shows a small symbol graph example related to the same interactive mathematical document as in Figure 10 from Example 1.

The symbol graph in Figure 11 supplies the presentation model (see Section 4.3) with information concerning dependencies of the symbols, and therefore the order in which symbols need to be introduced to the user to be understandable. The presentation model then adapts the content accordingly so that each user will be able to understand the presented knowledge.

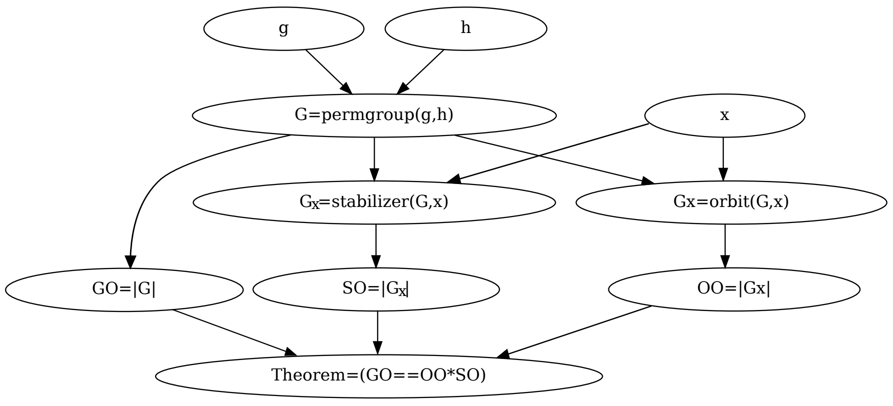


Figure 12: A variable graph related to orbits.

4.1.3 *The variable graph*

The third and last graph that we consider in our domain is a graph whose vertices are the mathematical variables inside our document. Let V denote the set of variables occurring in a given document.

A variable inside a document can have a value assigned to it by the author, the user, the document, or it can have a value that depends on other variables. For example, the values could be randomly chosen for generating different exercises, or they could be the answers of a previous exercise for a running example.

The variables in V are ordered by a relation $\sqsubseteq_V$, where $x \sqsubseteq_V y$ for $x, y \in V$ means that variable y depends on variable x. Here dependence means that the value of x is used to determine the value of y. Examples are mathematical expressions, such as $y = \cos(x)$, or instances of mathematical notions introduced, as $y = \text{VectorSpace}(3, z)$, where z is a variable whose value should be a field, for instance, $z = \text{GaloisField}(9)$. As in the previous cases, the *variable graph* is a dependency graph represented by a Hasse diagram.

Example 7:

In Figure 12 a partial variable graph is shown. In this variable graph, dependency relations between variables are visualized as they occur in an adaptive mathematical document. We will now show how different variables and their values from different parts of the document cooperate and can be used in examples concerning the topic of the document: permutation groups (see Example 6). Since the variables are not bound to any page and depend on their predecessors for their values, it becomes possible to create running examples throughout the document in which each occurrence of a specific variable contains the same value as anywhere else in the document. If values on which these variables depend upon change, then the values of these variables change accordingly.

Looking at Figure 12, one can see that the variables x, g, and h have no predecessors. Their values are therefore independent of any other variables and need to be set by either the reader of the document or be given a default value by the author of the document.

In Figure 10, it was shown that each theory had its own nodal page. Now we will expand the document shown in Example 6 by adding examples to the nodal pages shown in Figure 10. These examples make use of variables for their — on screen — computations and also for storage of their values. This allows examples on other pages to build on top of an example either on the same or on a previous nodal page.

At a first example, on the nodal page entitled *permutation groups*, the reader is shown how permutation groups work. For this purpose the reader is asked to supply two permutations g and h. The variable G is then used to store the permutation group generated by the permutations g and h. Of course the user is allowed to change the permutations g and h so as to see the effect this has on the group G.

Next on the same nodal page about *permutation groups* a different example shows the order of the group G resulting from the previous example. The value of the order is computed by an expression that is stored in the properties of variable graph variable GO; this variable also stores the value.

For future examples the student is also asked to select an element from X. The chosen element is then assigned to the variable x.

On the nodal page about *orbits* an example about the construction of orbits is shown. For this goal the permutation group G is used again in conjunction with the user picked element x. Together they provide the needed input to construct the G-orbit of x.The formula to do so is assigned to the variable Gx; this variable also stores the actual computed orbit. Note that this orbit is indirectly based on the user selected permutations g, h, and x on the nodal page about *permutation groups*.

On the nodal page about *stabilizers*, an example demonstrates the stabilizer theorem, by using G and x again to create the stabilizer subgroup in G of the element x. This subgroup is then assigned to the variable G_x, and just like the orbit Gx this subgroup indirectly depends on g, h and x.

Finally, at the nodal page about the *Orbit stabilizer Theorem* it is shown how the data from the variable graph comes together by accessing the value of GO, which holds the order of the group G, the variable OO which holds the order of the orbit Gx, and the value of SO which holds the order of the stabilizer subgroup G_x. It is shown that these variables are related by GO is the product of OO and SO. This result is assigned to the variable $\mathrm{Theorem}$.

Note that all variables depend upon the variables g and h and x. If any of these variables change, the variables that (indirectly) depend on them will to adapt to reflect these modified values.

No user data stored in the variable graph.
The variable graph as part of the domain model is, just like the other dependency graphs, immutable for users and is shared throughout the document. The variable graph is used solely to define the dependency relations and mathematical definitions of the variables and in keeping these sound. As such, the variable graph does not contain any values other than default settings made by the author. The values assigned to variables by a user, as described in Example 7, are user dependent and stored in the *user model* as will be discussed in Section 4.2.

4.1.4 *Synthesis*

Until now, the theory, symbol, and variable graphs have been considered as if they are independent of each other. However, given a mathematical document, a number of mathematical symbols and variables are found in content such as assertions, examples, proofs, exercises or remarks of the document. Moreover, each symbol or variable can occur in the various nodal pages of the document, linking symbols and variables with theories. Note that symbols also occur within variables from the variable graph.

This is formalized by the *occurs in* relation denoted by $\rightarrow$. The *occurs in* relation can be used between the union of the sets of symbols and variables, with the set of theories. Let $W = S \cup V$, then

$$W \rightarrow T \tag{5}$$

Symbols can also *occur in* variables; this is represented as $S \rightarrow V$.

The reverse relation, from the set of theories T to the *containing* elements from W, or from the set of variables V to the *containing* symbols from S, is called the *contains* relation, denoted by $\leftarrow$.

For $w \in W$, $s \in S$, $v \in V$ and $t \in T$ let us write $w \rightarrow t$, if w *occurs in* t, and also $s \rightarrow v$ if s *occurs in* v. And for the reverse relation, we write $t \leftarrow w$ to mean t *contains* the variable or symbol w. Similarly, we write $v \leftarrow s$ to mean v *contains* the symbol s.

We are now ready to define some maps based upon the *occurs in* and *contains* relations.

Let Σ_S denote the map which assigns to each theory $t \in T$ the set of symbols of S, that occur in t.

$$\Sigma_S(t) = \{s \in S \mid s \rightarrow t\}. \tag{6}$$

Similarly, let Σ_V denote the map which assigns to each theory $t \in T$ the set of variables from V that occur in t.

$$\Sigma_V(t) = \{v \in V \mid v \rightarrow t\}. \tag{7}$$

Combining $\Sigma_S(t)$ and $\Sigma_V(t)$,

$$\Sigma(t) = \{w \in W \mid w \rightarrow t\}. \tag{8}$$

Let Θ denote a map which assigns to each symbol and variable w the set of theories $t \in T$ *containing* w. So for $w \in W$, we have

$$\Theta(w) = \{t \in T \mid t \leftarrow w\}. \tag{9}$$

This set $\Theta(w)$ is called the *scope* of the element w.

Let w be a symbol or variable from W and consider the set $\Theta(w)$. If this set contains a unique minimal element $\theta(w)$ with respect to the poset of the theory graph, then $\theta(w)$ is called the *introducing theory* for w. As we will see later, all symbols and variables have either introducing theories or are considered prior knowledge to an interactive mathematical document. Where Θ maps symbols and variables to the theories in which they *occur*, the map Φ maps symbols to variables in which they *occur*; for $s \in S$ define

$$\Phi(s) = \{v \in V \mid v \leftarrow s\}. \tag{10}$$

The relation $\Gamma(v)$ give us the symbols that occur in a given variable $v \in V$.

$$\Gamma(v) = \{s \in S \mid s \rightarrow v\}. \tag{11}$$

Relations between graphs.

The maps Σ_S, Σ_V, Θ, Φ and Γ all connect the theory, variable and symbol graph with each other. This is made visible in Figure 13.

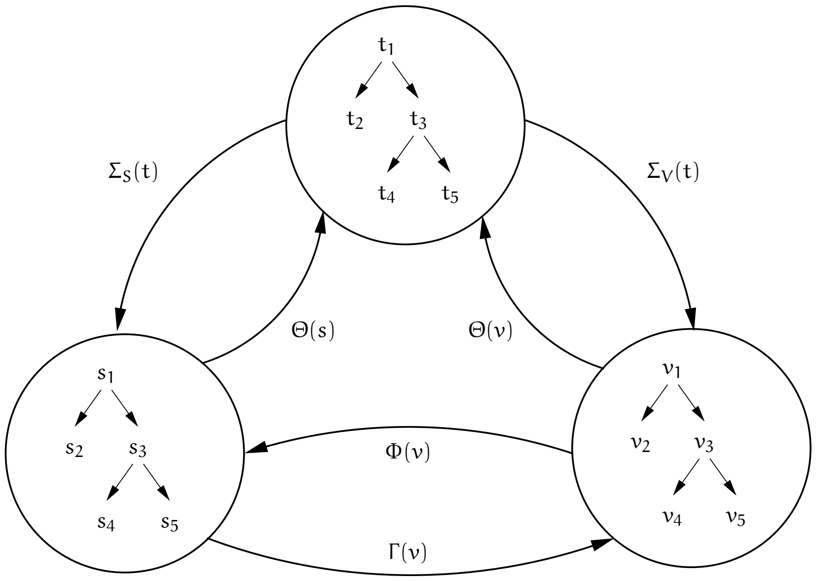


Figure 13: Inter graph relations.

Recall the partial ordering of $(T, \sqsubseteq_T)$ defining the dependency relations on the elements of T. This relation now helps us define the *dependency set* $\Delta_T(t)$ of an element t. The dependency set contains all $t' \in T$ on which the theory t depends, including t.

$$\Delta_T(t) = \{t' \in T \mid t' \sqsubseteq_T t\}. \tag{12}$$

Dependency sets can also be found in the symbol graph; for $s \in S$,

$$\Delta_S(s) = \{s' \in S \mid s' \sqsubseteq_S s\}. \tag{13}$$

Variable graphs also have dependencies, which in turn are described by

$$\Delta_V(v) = \{v' \in V \mid v' \sqsubseteq_V v\}. \tag{14}$$

The dependency set $\Delta_T(t)$ can be used to determine all symbols available to a theory $t \in T$. The set of these available symbols is then defined by

$$\Omega_S(t) = \{s \in S \mid \theta(s) \in \Delta_T(t)\}. \tag{15}$$

Similarly for variables we define

$$\Omega_V(t) = \{v \in V \mid \theta(v) \in \Delta_T(t)\}, \tag{16}$$

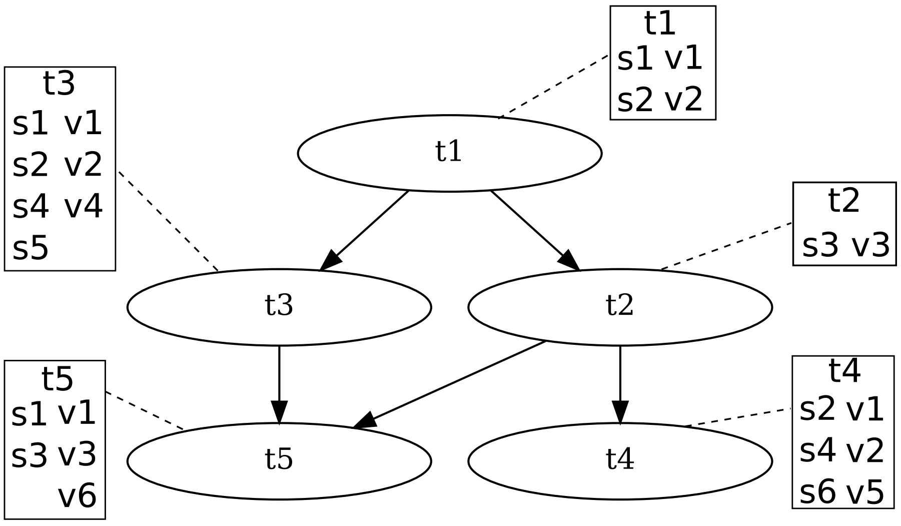


Figure 14: Used symbols and variables on nodal pages

and for both variables and symbols, define

$$\Omega_W(t) = \Omega_S(t) \cup \Omega_V(t) = \{w \in W \mid \theta(w) \in \Delta_T(t)\}. \tag{17}$$

The difference between Σ and Ω.
The Σ_S and Σ_V mappings must be seen as indicating which symbols and variables are being used by a theory, whereas the Ω relation should be considered indicating as what is available for a theory to use.

Example 8:

In Figure 14 an example theory graph is given. For each theory, the symbols and variables used are indicated. With the $\Theta(s)$ relation it is determined where a specific symbol or variable is used, for example:
$\Theta(s1) = \{t1, t3, t5\}$,
$\Theta(s5) = \{t3\}$,
$\Theta(v1) = \{t1, t3, t4, t5\}$.
The maps Σ_S and Σ_V yield the following values for the theories t1 and t5.
$\Sigma_S(t1) = \{s1, s2\}$,
$\Sigma_V(t1) = \{v1, v2\}$,
$\Sigma_S(t4) = \{s2, s4, s6\}$,
$\Sigma_V(t5) = \{v1, v2, v5\}$.

Observe that Θ and Σ are not each other's inverses. This becomes clear when one considers $\Sigma(t1) = \{s1, s2, v1, v2\}$, but $\Theta(s1) = \{t1, t3, t5\}, \Theta(s2) = \{t3, t4\}, \Theta(v1) = \{t1, t3, t4, t5\}$, and $\Theta(v2) = \{t1, t3, t4\}$. It is quite clear that $\Theta \circ \Sigma(t1)$ is not equal to just t1 as it would have to be for the maps to be each other's inverses. The same holds for Φ and Γ.

The set of dependencies found in the theories $t1$ and $t5$ are given by:
$\Delta_T(t1) = \{t1\}$,
$\Delta_T(t5) = \{t1, t2, t3, t5\}$.
The set of all symbols available to theories $t1$ ant $t5$ are:
$\Omega_S(t1) = \{s1, s2\}$
$\Omega_S(t5) = \{s1, s2, s3, s4, s5\}$
Similarly the variables available from the same theories $t1$, $t5$ are given by:
$\Omega_V(t1) = \{v1, v2\}$
$\Omega_V(t5) = \{v1, v2, v3, v4, v6\}$

4.1.4.1 *Soundness*

We call a context-enriched adaptive mathematical document *sound* if

- For all symbols or variables $w \in W$ there exists an introducing theory $\theta(w)$, unless w has been marked as *prior-knowledge*. Prior knowledge does not require introducing theories. All symbols and variables that are used and are not considered prior knowledge need to be introduced somewhere in the document. If there is more than one introducing theory then that is an indication of a possible error. Multiple introducing theories are not necessarily incorrect, although it is considered bad practice and should be avoided.

- For all symbols $s, s' \in S$ for which $s \sqsubseteq_S s'$ holds, $\theta(s) \sqsubseteq_T \theta(s')$ must also hold. When a symbol s is introduced in a theory t, all other symbols s' that depends upon s are introduced in either the same theory t or in a theory t' that contains theory t in the dependence set $\Delta(t')$.

- For all variables v, v' the validity of $v \sqsubseteq_V v'$ implies $\theta(v) \sqsubseteq_T \theta(v')$; as in the case of symbols, all variables v' that are used and depend on variable v should be introduced in the same theory as the variable v or in a later theory that is (in)directly connected in the partial ordering.

Note that when all afore-mentioned criteria for soundness have been met, theories can make use of only those symbols and variables that were introduced in a theory where the current theory depends upon. All symbols or variables have to be introduced in a theory that is in the dependence set $\Delta(t)$ of the current theory. Thus the following will also hold.

Lemma. *For all* $t \in T$*:*

- $\Sigma_S(t) \subseteq \Omega_S(t)$;
- $\Sigma_V(t) \subseteq \Omega_V(t)$.

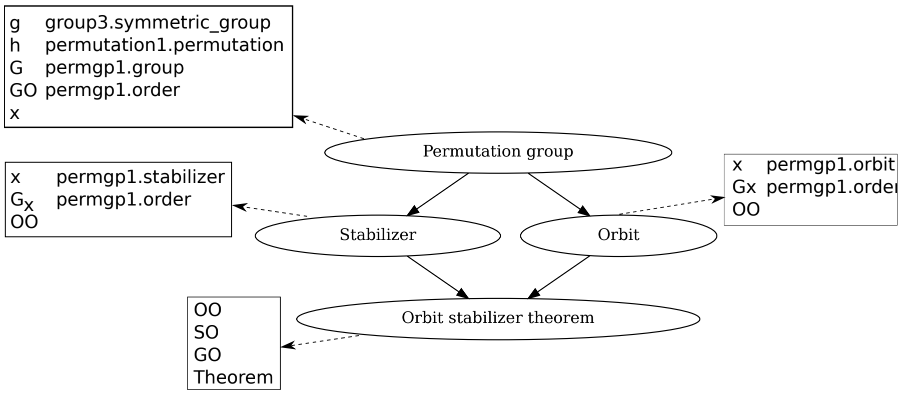


Figure 15: Running example usage

Proof. Fix $t \in T$. Let $s \in \Sigma_S(t)$. Then $s \to t$, so $t \in \Theta(s)$. But $\theta(s)$ is by definition the smallest of all theories with this property, so $\theta(s) \sqsubseteq_T t$. This establishes $\theta(s) \in \Delta_T(t)$, which is tantamount to $s \in \Omega_S(t)$. Hence the first assertion holds. The second is proved analogously.

If the theory symbols and variables are indeed all sound, then the following also holds for all $s \in \Phi(v)$;

$$\theta(s) \sqsubseteq_T \theta(v). \tag{18}$$

This notion can then be expanded upon, by expecting the same for each variable v depends upon;

$$\forall v \in \Delta_V(v) : \forall s \in \Phi(v) : \theta(s) \sqsubseteq_T \theta(v) \tag{19}$$

Soundness is a necessity in a context-enriched interactive mathematical document, as it prevents a reader from going astray because of obvious inconsistencies in the set-up and use of variables, symbols, and theories in relation to each other. Our notion of soundness is primarily a kind of well-formedness of the mathematics served and has little bearing on formal mathematics.

The mathematical context emerges as the structured sets of symbols and variables, as well the values of the variables, pertaining to the theory visited by the user. As this is dependent on the user, further treatment of it belongs to the next subsection.

4.1.5 *An example*

In this section we take another look at the previous examples. We will see that these examples can be taken together into a combined example which includes the theory, symbol and variable graphs.

In Example 6 we introduced a theory graph about *permutation group*, *orbit*, *stabilizer* and the *orbit stabilizer theorem*. This theory graph is shown in Figure 10, with nodal pages that are associated to the theories in the graph. Figure 15 shows the same theory graph, but instead of the nodal pages, the variables and the symbols that occur within the nodal pages are listed. These symbols are also in the symbol graph of Figure 11. Both these graphs and the variable graph from Figure 12 are from the same domain model and as such the relations mentioned earlier apply.

In Figure 16 the theory graph and symbol graph are shown. It also shows the usage of the symbols by the various theories; these are the maps Σ_S and Θ. Note that the theory *Orbit Stabilizer Theorem* is not referred to at all. This is explained by the fact that *Orbit Stabilizer Theorem* only uses the symbols *arith1:equal* and *arith1:times* which are both considered prior knowledge and are therefore not part of the symbol graph. There are also symbols in the symbol graph which are not referred to; this is due to this particular example with the dressed-down theory graph and nodal pages which uses far fewer symbols than would the case in a real document. The symbol graph only requires symbols that will be introduced in the document and therefore all symbols will be used by nodal pages. Exception to the rule would be symbols that have been added for prior knowledge reasons. Allowing the document and the user to refer to these prior knowledge symbols.

In Figure 16 the sets $\Sigma_S(t)$, where $t \in T$, and $\Theta(s)$, where $s \in S$, are visualized. By following all dotted lines from a theory t to all connected symbols the set $\Sigma_S(t)$ for theory t is obtained. Similarly, in the reverse direction: by following all dotted lines from a symbol s to all connected theories, the set $\Theta(s)$ is obtained.

Figure 16 can be simplified if we only consider $\theta(s)$. The relation $\theta(s)$ indicates where symbols are introduced. The simplified graph representing $\theta(s)$, where $s \in S$, is shown in Figure 17. The figure basically divides the symbol graph into four parts. With the exception of the first part, which is considered prior knowledge, the parts are associated with the nodal page that introduces the symbol in the document. Because theses nodal pages are linked to the theory graphs, the different parts in the Figure 17 are also linked to the theory graph. Indeed the parts have the same ordering as can be found in the theory graph in Figure 16. Some variations are possible, when a nodal page does not introduce symbols; in such cases the theory nodes are skipped. Other deviations where the ordering is altered in comparison to the theory graph, indicate problems with soundness.

Also note that by taking the union of all introduced symbols from the theory pages that exist in $\Delta_T(t)$ (all predecessors of t and t itself), we obtain $\Omega_S(t)$.

Combining the theory and variable graph gives similar results. Indeed, it represents the sets $\Sigma_V(t)$ where $t \in T$ and $\Theta(v)$ where $v \in V$. Again, it is possible that not all theories have references to variables. This happens when nodal pages do not use any variables. However, all variables should have a connection to at least one theory; if there

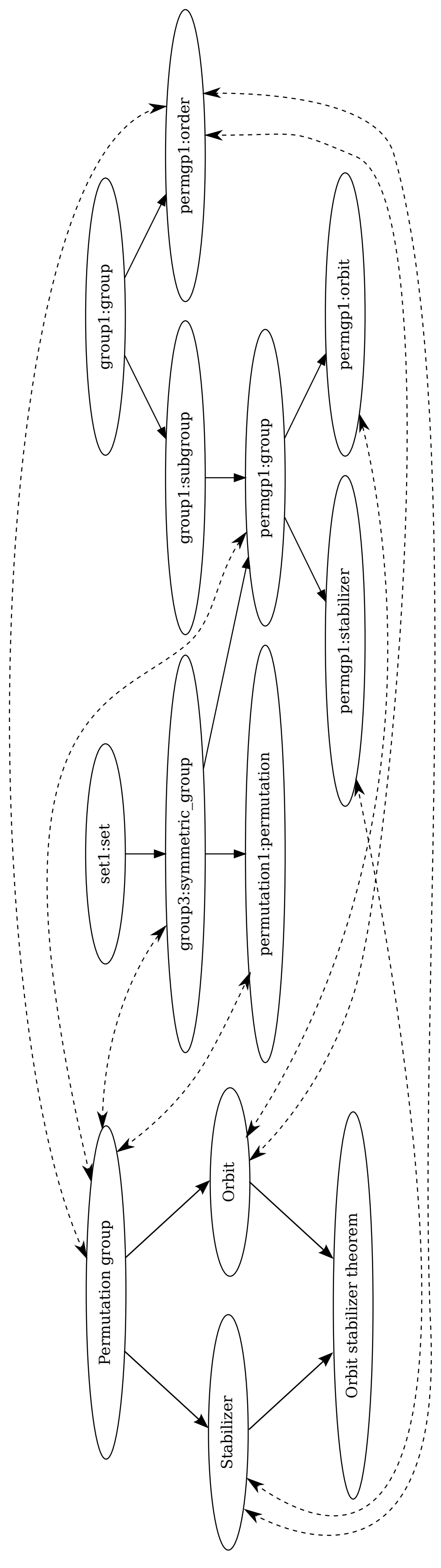


Figure 16: Theory graph (on the left) and symbol graph (on the right) relations Σ_S and Θ

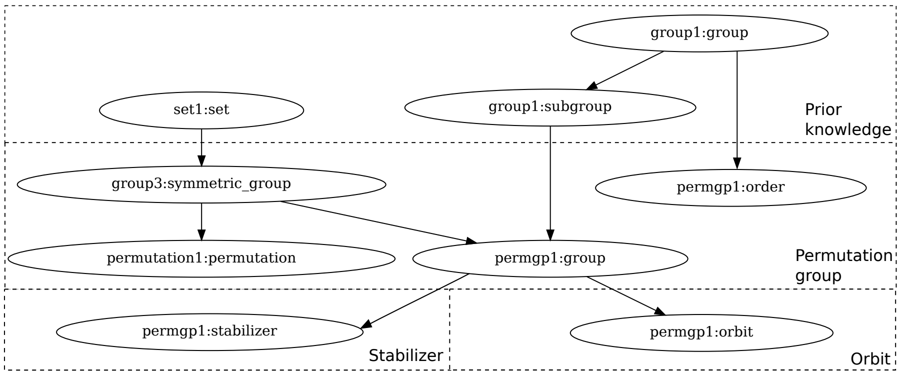


Figure 17: Symbols introduced at theories ($\theta(s)$)

is none it means the variable is not used in a theory at all, and is a redundant step in the dependency set $\Delta_S(s)$ of the variable graph. Redundant does not mean illegal, but one can wonder if the variable is needed or desired.

Examining the $\theta(v)$ diagram it will produce a similar graph to the one in Figure 17. The same remarks also apply in this case.

4.2 USER MODEL

A user model's responsibility is to keep track of any user-related data so that the presentation model can adapt content from the domain model to the needs of the user. The domain model is static and is created just once for a document. It is therefore immutable during the use of the adaptive mathematical document. In contrast to the domain model, a user model is dynamic and subject to constant change to stay accurate and appropriate as to reflect the ever changing context of the user. The user model assists the presentation model and needs therefore to store the data required by the presentation model and to keep this data up to date.

Roughly the data required is divided into three parts; *mathematical context*, *logistic context* and the *knowledge context*. These include values of variables connected to the user and a document, user-specific logistic information, and the knowledge information such as acquired skills, read topics and passed exercises.

Needless to say, every user has their own instance of a user model, making all data stored in each instance of the user model private for each user.

4.2.1 *Mathematical context*

Adaptive mathematical documents rely heavily on variables. Variables can store different kinds of values but in this section we will limit ourselves to mathematical content only. These mathematical expressions range from basic values to complex OpenMath expressions. The use of variables allows existing content to recur in different forms due to the different values of the used variables, without the need to alter any of the content. This kind of reuse of content typically occurs with exercises and examples allowing an author to just write it once but present it multiple times with different values.

Note that to the user the content changes when the variables are changed. This demonstrates that the values of variables define a page state. Now imagine extending the scope of these variables from a single page to the whole document in such a way that other pages can use the same values as well. This introduces a user state within a document. Moreover, an important part of the notion of context: it represents the *mathematical context* of a user.

As variable values are data belonging to the user and to the document, it is the responsibility of the user model to keep track of variable data and to allow for changes in their values.

The variable graph, as described in Section 4.1, contains definitions of variables. As such, the variable graph only contains static immutable data and does not contain any variable values set by the user. The values belonging to these variables are stored in the user model instead — to be more precise in the mathematical context — completing the variable definitions of the variable graph.

Here, the difference between the symbol graph and the variable graph becomes quite clear: the knowledge information (described in Section 4.2.3) tells us which nodes t the user has visited, and so $\Sigma_S(t)$ can be determined from it. In contrast, the knowledge of $\Sigma_V(t)$ is insufficient and needs to be complemented with the values from the user model of the variables that it lists. As these values may vary as the user progresses and can be called at each instance, these are contained and maintained in the mathematical context.

The mathematical context is stored in the *user model* as a list of context variables, where all context variables are represented by at least one tuple of the form *(Variable ID, Value, Timestamp)*. Each change to a variable will lead to a new tuple being added to the *user model*, creating a list of tuples representing the mathematical context of any variable at any given time.

4.2.2 *Logistic context*

Logistic context is the set of user data that contains logistic information about the user. It consists of data such as name, date of birth, address, education, affiliation, profession, etc. The exact composition of data stored as logistic context is up to the author to determine. Although this data has no mathematical meaning, it is required to be able to adapt a document in a less mathematical way to the needs of the reader. By incorporating the name of the user, his or her profession, study, culture or notation, a better connection between the user and the content is created.

Logistic user information is not interconnected by nature. The data is considered an independent set of properties stored by variables. The logistic context is captured by a list of *(Property ID, Value, Timestamp)* tuples. This list is stored in the user model. Each tuple contains a *Property ID* containing the name of the user property, a *Value* of that user property and the time when this data was set. Organizing the list of logistic information in this way allows an author to add those properties he or she deems necessary for a document without being restricted by the possibilities offered by a pre-defined set of variables. The presence of timestamps allows retrieval of previous values of properties in case they have been altered and need to be reverted to their previous values to reflect the status of the context at an earlier time. Note the similarity between the *mathematical context* and *logistic user information* tuples. This similarity will allow us to regard logistic context in the same way as mathematical context.

4.2.3 *Knowledge context*

Knowledge context concerns all data relating to the theory graph and symbol graph as discussed in Sections 4.1.1 and 4.1.2. Visited or mastered nodes from the theory or symbol graphs are contained in the knowledge context. Both the theory and symbol graphs are situated in the *domain model* instead of the *user model*, and as such the knowledge information refers to the nodes in the theory and symbol graph. In Section 3.1, we discussed different types of user models. The most convenient user model for an adaptive document to keep track of knowledge is the overlay model. In the case of an adaptive mathematical document this is even more valid due to the hierarchal nature of both this model and mathematics in general.

Keeping track of user progress is done by storing event tuples containing references to specific nodes from either the theory or the symbol graph and actions. These tuples take the form *(Node ID, Action, Value, Timestamp)*. Typical values for *Action* may be *Visited* or *Mastered*. *Visited* indicates that the user has visited that particular theory or symbol explanation. *Mastered* means that the adaptive document considers the theory or symbol understood by the user. If so desired, an author is free to add extra actions allowing the adaptive document to fine-tune the document even more to the user.

The *value* field is rather straightforward as it stores the value associated to the action. The type of value is not specified beforehand. It could be boolean, as would be suitable for *mastered* actions, but it could also be a number value to indicate the times a nodal page has been visited. Note that the *Timestamp* records the moment the tuple was added to the *user model*. As before, the timestamp is required to let the document re-adjust itself to a specific point in the past for replay purposes.

The knowledge events that are stored enables different strategies for how to work with the overlay model. It can be utilized as a binary system, in which it only answers whether or not a user understands a certain concept. It can also serve as an incremental system, in which the overlay model predicts to what degree a user understands a concept. In both approaches an analysis of the stored events is required.

4.3 PRESENTATION MODEL

The third and last model we discuss in this chapter is the *presentation model*. After discussing the domain model and the user model, the presentation model is the place where it all comes together. The presentation model is responsible for the adaptation and presentation of the content. Choices are made on what content is shown and how to present it to the user. The presentation model is responsible for the inclusion of content that deals with required knowledge or symbols for a selected page. It is also responsible for fetching the values belonging to variables from the variable graph and using these in an example that may or may not be included, depending on the decisions made in the presentation model.

The adaptation process on a document is intended to make content more accessible and easier to understand for the user. When content is more easily understood it makes the knowledge transfer more efficient and emphasizes the importance of a proper functioning presentation model.

To achieve adaptation of content, information and decisions based on information are required. The information originates either from the domain or user model or is gathered by the interactive document itself. The process of information gathering is done by queries. The decisions in the adaptation and presentation process are then made by feeding the results of the queries to a set of rules. Rules effectively represent the author's interpretation of a situation as described by the information available. Based on this interpretation the rules draw conclusions and perform the required actions in adaptation and presentation of the content. The final step is then to present the content to the user; in our case, this is the MathDox Player translating the content to HTML pages viewable by the user in a web browser; see Chapter 2 for more details.

The process of gathering information, having this information interpreted by the set of author-defined rules and delivering the results to the user is what we define to be the presentation model, see Figure 18.

Flexibility in the presentation model is achieved by the possibility of adding new and very specific rules for each interactive mathemat-

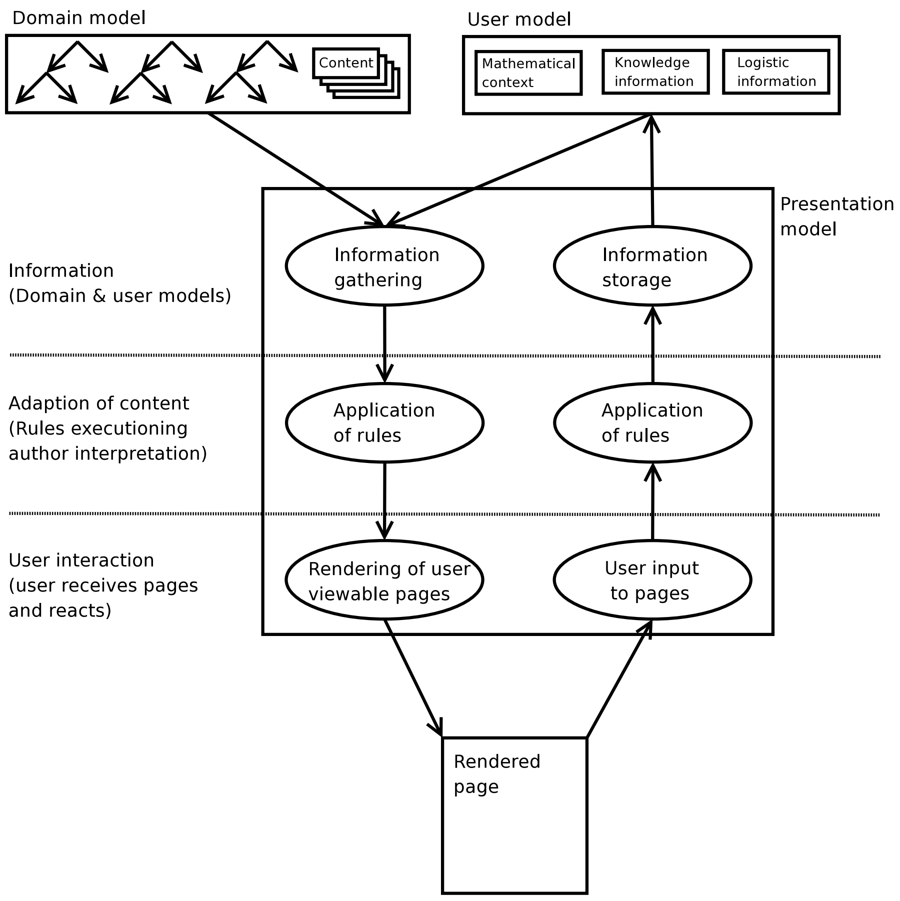


Figure 18: The schema of the presentation model.

ical document. There is, however, no need to require an author to supply each and every single rule that is used by the presentation model. There are rules that are recurring and are applicable to any interactive mathematical document. By creating a standardized set of frequently used rules, documents will be easier to create and more consistent.

The main responsibilities of the presentation model are: communication with the user and domain model, selecting and structuring of content and the presentation of content. In the following sections we discuss these three architecture aspects of the presentation model in detail.

4.3.1 *Communication with the user and domain models*

The domain model as presented in Section 4.1 provides an adaptive document with all the required building blocks such as the theory, symbol, and variable graphs, as well as (unadapted) content. The user model (see Section 4.2) stores data related to the user in three separate categories: mathematical, logistic, and knowledge context. Note that the mathematical and knowledge context get their meaning from the graphs in the domain model and that the logistic context is the place to store preferences to take into account when content is presented to the user. Unlocking the information offered by both the domain and user models and making it available to the presentation model is an essential step and is done by queries.

The purpose of queries is to obtain or store information. Upon a user entering a page, the presentation model acts by retrieving the relevant information. The information retrieved by queries is to be used by an adaptation rule to adapt (parts of) a requested page to fit the needs of a user.

When the user reacts to the page presented to him or her, the user's actions are analyzed by a rule that implements the conditions that define if and what data should be updated in the user model. The rules that update the user model implement what is sometimes in literature referred to as the observer model. Indeed, their job is to observe the behavior of the user, draw conclusions from these actions and store these in the user model.

4.3.2 *Selecting and structuring of content*

The most straightforward way to offer an adapted page to the user is to create a number of unadaptable pages beforehand, each written for specific situations. This is an approach that would do well with the scalar and stereotype models, see Sections 3.1.1 and 3.1.2. The adaptation process is then limited to selecting the most appropriate prefabricated page. This approach offers pages of potentially good quality as problems ill fitting pieces of content will have been dealt with by the author. However, creating pages for all possible situations becomes more and more complex and harder to achieve as the number of different situations and content in general, increases.

For this reason, adaptive systems often fragment their content and offer these fragments as building blocks to create pages specifically for each user. Prefabrication of fragments as opposed to prefabricated pages require less effort due to the possibility of reusing the same fragments on multiple pages in different combinations. By making a correct selection and rearrangement of finely fragmented content, all the knowledge a user needs is included into the offered page. Such a page includes fragments with content that is required and omits those fragments that are not needed by or not suited for the user. Each page created in this way is therefore tailor-made for each individual user.

The selection of which fragments to include, and with that the content selection, is done by rules. These rules decide what definitions need to be included into the presentation content, decide on examples that demonstrate certain principles which are directly connected to the topic at hand, or the selection of examples that demonstrate applications of the topic in a field the user is more familiar with. These rules first query the domain and user model — for instance the theory and symbol graph from the domain model and the set of current knowledge from the user model — to get the data about the user that is of influence on the decisions made by the rules and then decide on what content to include or exclude. The actual decision process itself is implemented by the author, so the selection of content to be shown to the user is the result of the implementation decisions made by the author. We therefore call these rules *author rules*.

As the name *author rules* implies, the decisions made by the rules are not influenced in a direct way by users. Rather they are the results of the author's automated view on the user data stored in the user model.

Decomposing and recomposing knowledge.
Fragmenting content and recomposing it on a page comes quite close to the process of *decomposition of knowledge* and *recomposition of knowledge* as explained in [173]. The difference here is that Kohlhase and Kohlhase talk about the human side where knowledge from the author is fragmented when it is put into a document. The fragmented knowledge is then composed back into an understanding by a reader of that document. Our approach uses the knowledge decomposition as done by the author to compose a page tailor-made for the reader, improving the understanding of the reader in the matter and increasing the process of recomposing knowledge residing with the user.

Content Structuring.
As described in [140], creating a page from fragments is subject to the processes *content adaptation* and *content presentation*. The process of *content adaptation* is performed by two sub-processes: *content selection* and *content structuring*. The *content selection* deals with the selection of the fragments that are important to the requested page and user. The sub-process of *content structuring* deals with finding the right way to fit these fragments together and adapting them so that they fit with each other on a page in such a fashion that the user is not bothered by the fact that the page has been composed of fragments rather than being written as a single page. Within the mathematical context model we attached a page to each node from the theory graph: the nodal page (see Section 4.1). From these nodal pages, fragments that deal with the theory at hand or other fragments of interest to the user are included. In our mathematical context model, the fragments that are included do not necessarily contain only static text, but also more adaptation rules. These rules allow fragments to adapt their contents to their neighboring fragments and ease the transition of the content between these fragments.

4.3.3 *Presentation of content*

After selecting and structuring the fragments there is still the job of adapting the content to the needs of the user. Adaptation of the presentation of content is achieved by again making use of rules.

Techniques used by presentation models for adaptation among others include:

- Providing and coloring extra links that make additional content -within the interactive mathematical document or by means of external links- easily accessible.
- Adaptation of notation as used in the context so as to match the standards familiar to the user, an important design goal of Panta Rhei [184].
- Offering of stretch texts that can further explain concepts by making hidden texts visible that explain concepts in greater detail.

The set of rules that are involved with the presentation of content do — as opposed to the author rules — take into account user preferences stored in the logistic context of the user model. These sets of rules are called *user rules*, they allow user input to affect their behavior. For example, one rule that takes into account the preferences set by a user is a rule that displays a link to an external source for more information on a specific topic in a color specified by the user. Another example is a rule that consults the logistic information to let the user replace the symbol i for the imaginary number $\sqrt{-1}$ for an j.

Example 9:
The student Steven visits the adaptive mathematical document about permutation groups. He has not yet visited any page before and wants to learn straightaway about the *orbit stabilizer theorem*, without first reading anything about *permutation groups*, *orbit*, and *stabilizers*.

As a consequence Steven has no registered knowledge in the knowledge context about either of these theories or the symbols involved; what Steven is missing exactly can be deduced from the theory graph and the symbol graph. Steven needs knowledge about all predecessors of the *orbit stabilizer theorem* and he is also missing the knowledge concerning the symbols that were introduced in the corresponding nodal pages.

This information is retrieved by queries to the knowledge information in the user model, as opposed to the theory and symbol graphs in the domain model. Next, the information is forwarded to an author rule that decides to include those content fragments that deals with this knowledge. These rules can take into account Steven's background when they decide how elaborately the content needs to be shown. A student of mathematics would require less elaborate content about the missing knowledge than a student in computer science or engineering. Another rule might employ a query to find out if Steven likes examples on his pages; this would be information stored in the logistic context section of the user model.

A user rule will adapt all included examples to work with the values Steven has supplied for the permutation variables g and h or any derivatives of these variables in the variable graph (see section 4.1.3 for the used variable graph).

The next step is translating all content to HTML, and presenting the page to the user. The mathematical context model then awaits a reaction from Steven.

If the page included an exercise that was answered correctly by Steven, the interactive mathematical document might conclude with a rule that Steven has mastered the presented knowledge and will fire a query that will record this in the knowledge section of the user model. A wrong answer is another reason for updating the knowledge information in the user model. If the page has no exercises, the system might still update the user model.

SUMMARY

In this chapter we discussed the domain, user and presentation models. The domain model encapsulates the content and the structure of an adaptive document by means of the theorems, proofs, definitions, symbols, etc. It uses the theory and symbol graphs to store this structure.

The user model holds user-related data to facilitate adaptation to the needs of the user. This data includes personal information like profession and identity, but also the experience, skill level and history within the system.

Lastly, the presentation model combines both the domain and user model by means of queries and rules, to present the user with an adaptive mathematical document. The model adapts and transforms the content to be displayed from the domain model according to the needs specified by the user model.

5 IMPLEMENTATION OF CONTEXT IN MATHDOX

The mathematical context model has been described so far in an abstract manner without much attention to implementation details. The reason for this is that the mathematical context is a model in its own right and not necessary linked to the MathDox Player. Our implementation however was done on top of the MathDox Player. In this chapter we discuss the details of the context implementation.

Enriching MathDox with the mathematical context model.
The MathDox format and the MathDox Player are already a powerful team, capable of producing and delivering interactive mathematical content to the user. Therefore the implementation of the mathematical context model is not allowed to interfere with the original functionalities of the MathDox Player or format. To this end we kept the implementation of the mathematical context model out of the MathDox Player altogether. This will guarantee that any MathDox documents written without the mathematical context model will keep working as intended. The mathematical context is implemented as an add-on to the MathDox Player, in much the same fashion as web browsers these days expand their functionality by allowing add-ons and plugins. As such, the implementation also serves as a demonstration of the MathDox Player abilities to be expanded upon.

Chapter overview.
In Figure 19 a schematic overview of the context implementation is given. The MathDox Player is shown in the middle, the lower part depicts the core of the MathDox Player, while the upper part shows the addition of the mathematical context model. Especially the context object, the Jelly interpretation and the change in interpretation sequence — the only change to the MathDox Player necessary — are important aspects and are discussed in more detail in Section 5.1. The document file, graphs, conditions, content and fragments, in short all that matters for the domain model are discussed in Section 5.2. In Section 5.3 we talk about the database design. This includes among other topics knowledge context, mathematical context and variables, logistic context and logging. Finally, the presentation model including the query and rules, as well as some examples are discussed in Section 5.4.

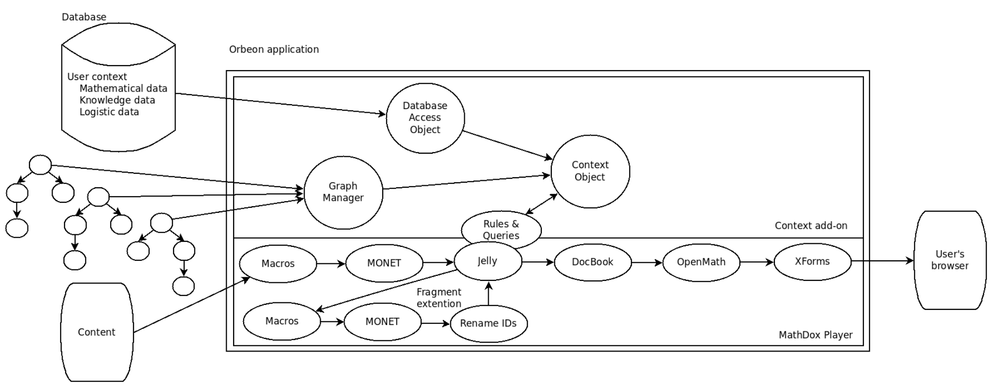


Figure 19: The MathDox Player and the context add-on.

5.1 IMPLEMENTATION DESIGN

In MathDox a page determines what is being shown to the user. This is a direct consequence of the stateless character of the MathDox Player. As stated, to remain in the spirit of the MathDox Player, the MathDox Context implementation is not centrally driven either; there is no center of operations responsible for making decisions on what content is being offered or adapted. Instead, the requested MathDox page itself determines what content to offer and decides what needs to be done to adapt content to the needs of the user.

This is in contradiction to for example ActiveMath [1] where a centralized rule engine is responsible for creating courses based on data known about the user. Mixing content and presentation also goes against the principles of modern day computer science. Indeed mixture of content and rules reduces reusability of content on other pages, where quite possibly other rules apply causing conflict.

However a decentralized approach with — to a limited extend — mixture of content and rules also has advantages. A centralized approach tends to create a mosaic feel in adaptive documents, the separate pieces of content remain present as the seams between these pieces can be quite obviously identified. Also in our opinion the mixing of rules and content offers more freedom for an author to create adaptive mathematical documents. This may include different kinds of documents than intelligent tutor systems, or not even related to education. Later in this chapter we will introduce an approach based on fragments that allows reuse of content and content and rules to be less mixed while still maintaining the decentralized approach.

5.1.1 *The context object*

For a MathDox page to adapt content to the needs of the user, methods are needed to consult the static data as stored in the domain model and the dynamic user data as stored in the user model. This data is processed and conclusions are drawn by the presentation model's rules as coded into the MathDox code on the page. Armed with these data sets and interpretations as performed by the presentation model's rules, the MathDox code is able to make the right adaptation during its interpretation path from source code towards HTML code, which forms a web page for the user.

The means offered to a MathDox page for accessing data from the domain model and user model, comes in the form of the *context* object. The context object can be seen as the center of the mathematical context model implementation. It embodies the domain and user model of the mathematical context model by keeping track of the theory, symbol and variable graphs of the domain model and by providing access to the database with user related data from the user model.

The context object as a session.
In our implementation of the mathematical context model for the MathDox Player we designed the context object to be a Java class which is implemented following the singleton [163, 164] design pattern. As such the context object is only created when it is needed and is reused afterwards. There will always be at most one object of this class at any given time, hence the name singleton. Indeed, there is always just one context object available within the MathDox Player implementation.

The benefits of using the singleton approach are two-fold. Each page request is seen as an independent request by the MathDox Player, and there is only a very minimal session environment available for storage of user related details within the MathDox Player. In fact, the only items the MathDox session does store are not even required for most MathDox pages. They are the name of the user and his or her password for possible database access. To keep in line with the statement of not altering the code of the MathDox Player, we kept the minimal MathDox session and added our own more elaborate session within the context object. These sessions are responsible for caching user data and storing this data after the session expires. A reference to the context object from rules, typically implemented by Java or Jelly code is obtained by using the singleton's static *getInstance()* method of the context object, making it as easy to use as the session interface of a typical web application.

Caching of Domain model graphs by the context object.
Another direct consequence of the singleton design of the context object is that all documents and all users of a MathDox Player share the same context object, making it possible to maintain just one copy of each theory, symbol or variable graph from the domain model of an adaptive document. Since these graphs are immutable it does not matter how many different users actually use them, nor how many different documents reuse the same graphs. Each graph will only be

loaded once when it is actually needed and is reused by any user or document in need for it. For this purpose the graphs are identified by the URLs that where used to locate them.

Caching of user model data.
The context object also serves as a database access point. It provides access to the data stored in the user model. If not yet available the data will be retrieved from the database and cached. For this purpose the context object contains a separate database access object (DAO design pattern) [121, 156], responsible for all database interaction. As there is just one context object that is reused by all users, the object also has the responsibility to only grant access to data belonging to that particular user, and hence handle privacy issues.

Caching user data gives the context enriched MathDox Player a performance gain as the number of queries to the database is reduced. An activity table was added to the context object to keep track of expiring sessions. When a session expires, cached data needs to be stored if changed and purged from the system.

5.1.2 *Jelly implementation*

The key to the implementation of the mathematical context model as an add-on to the MathDox Player is to make use of the extensibility and flexibility of the MathDox format. The architecture of the MathDox format allows for a combination of different execution steps, see Figure 20 which was previously shown as Figure 3 in Chapter 2. Especially the Jelly phase is very suitable for building such extensions. Extensions in Jelly may take the form of web calls for extra information or custom tags for new and different behavior of the MathDox code. With custom tags also comes the opportunity to add Java classes for more advanced implementation aspects, such as interaction between the MathDox format and the context object. This Jelly phase and the suitability for extensions will be used by queries to access the context object and allows for the implementation of the mathematical context model.

In Figure 22, we take a closer look at the Jelly execution step as seen in Figure 20.

Example 10:
For the purpose of understanding fully what happens at the Jelly execution step, we will demonstrate MathDox code being interpreted according to the schema in Figure 20. The code shown in Listing 10 is in progress of being interpreted and is just before the Jelly execution step, and consists of Jelly core and custom tags. These tags start with the *c:*, and *mdu:* prefixes. Specialized XML Jelly tags are also available, but not used in this example. The code

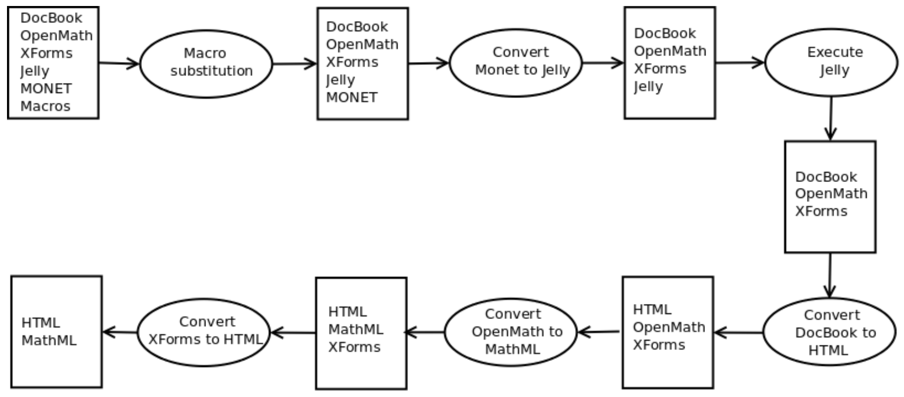


Figure 20: The MathDox interpretation schema.

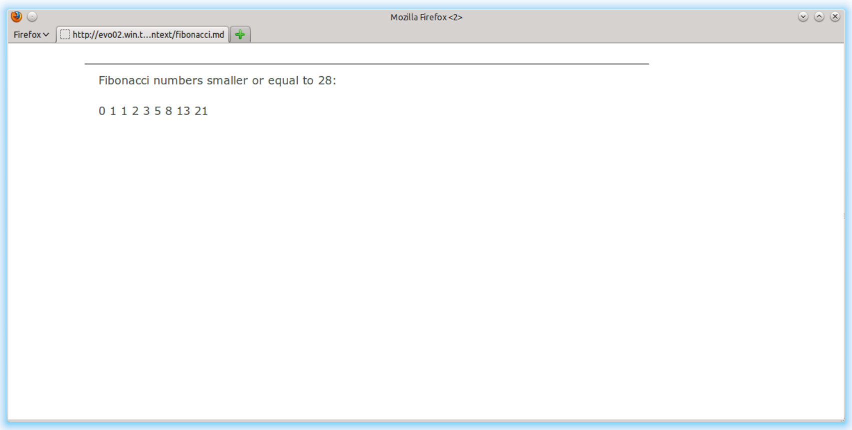
shown in Listing 11 is code in which all Jelly tags have been executed and replaced by new expressions in either the DocBook, OpenMath, or XForms format. Note that code in between conditional tags such as `<c:while>` or `<c:if>` only is translated when these test conditions evaluate to `true`. The generated page is shown in Figure 21

Listing 10: A MathDox code example containing Jelly

```
<article xmlns:c="jelly:core" xmlns:mdu="jelly:org.mathdox.util.
    UtilLibrary">
  <para>
    <c:set var="a" value="1"/>
    <c:set var="b" value="0"/>
    <c:set var="c" value="0"/>
    <mdu:random var="d" minimum="10" maximum="50"/>

    Fibonacci numbers smaller or equal to ${d}:
  </para>

  <para>
    <c:while test="${c le d}" >
      ${c}
      <c:set var="c" value="${a+b}"/>
      <c:set var="a" value="${b}"/>
      <c:set var="b" value="${c}"/>
    </c:while>
  </para>
</article>
```

Listing 11: A MathDox code example with executed Jelly

```
<article xmlns:mdu="jelly:org.mathdox.util.UtilLibrary">
    <para>Fibonacci numbers smaller or equal to 28:</para>
    <para>0 1 1 2 3 5 8 13 21</para>
</article>
```



```
Fibonacci numbers smaller or equal to 28:

0 1 1 2 3 5 8 13 21
```

Figure 21: Output of a MathDox page with Jelly statements

Creation of custom tags.
Jelly tags are extremely useful for conditional logic and for gathering (external) information on a MathDox page. These are properties that are required to access the domain and user models via the context object.

Listing 12: A library definition for custom tags

```
public class TheoryBlocksTagLibrary extends TagLibrary{
 /**
   registration of Jelly custom tags
 */
 public TheoryBlocksTagLibrary() {

  // returns all variable names in a given symboltable.
  registerTag("get–variablenames", GetVariablesNamesTag.class);

  // test conditions of variables
  registerTag("test–conditions", TestConditionsTag.class);

  // Returns current doc id as given in the document.xml
  registerTag("get–docid", DocumentIDTag.class);
```

The creation of custom tags is quite straightforward. It requires a Java class that extends the *TagLibrary* class from Jelly, in which the names of the custom tags are linked to classes that implement the tags, see Listing 12. Adding the new custom tags to the MathDox Player only requires a jar file being added to the classpath of the MathDox Player. This jar file must contain the classes implementing these new tags and the extended *TagLibrary* class. A Java class implementing a custom tag needs to implement the *TagSupport* interface and supply an implementation for the *public void doTag(XMLOutput)* method. This method takes an XMLOutput object to which output needs to be written. The XMLOutput object is provided by the Jelly execution engine. Of course the output written needs to be valid XML as the output is placed into the XML of the MathDox code that is being processed. For convenience an XMLWriter object can be used to assist in generating valid XML. As it turns out it often pays off to implement the reasoning for these custom tags in yet another separate class. In this way that specific reasoning can also be used by other tags and multiple reasoning objects can be combined to form new reasoning.

Implementation of rules and queries.
Tags typically are the tools used for information retrieval, the previously named queries. Their Java nature however also allows to make decisions based on this information and therefore they can also be used as implementation of rules from the presentation model. But as we will see in Section 5.4, tags are not the only way to implement rules.

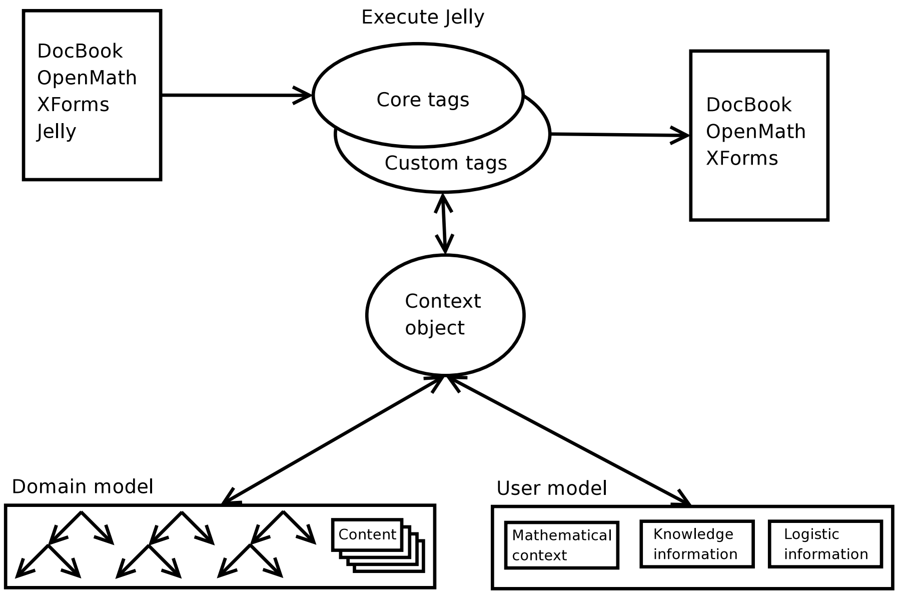


Figure 22: A closer look at the Jelly phase and context influence on a page

5.1.3 *Alterations in the MathDox Player*

For the context add-on to function properly some extra code needed to be added to the MathDox Player. This was limited to some extensions of the translation steps as depicted in Figure 20. These code extensions in the MathDox Player were kept to a minimum so as not to disturb the original behavior.

These changes included the addition of an identification process, necessary to identify users and to retrieve their data from the database. Until now, identification was either not necessary or left to a learning management system.

Another change included the execution process of MathDox fragments (see Section 5.2.4 for more information about fragments in MathDox). Until the introduction of fragments MathDox only used static inclusions performed by XInclude [117] at the very beginning of the translation process. Fragments as discussed in Section 3.3, required also conditional inclusion of fragments and therefore the need to be processed and be included at the Jelly phase of the MathDox page interpretation. At the same time fragments require a complete interpretation of its code as not to lose any macros and MONET expressions within a fragment. Fragments therefore require a separate partial translation to keep the translation process in sync with the translation process of the main document.

With the exception of the partial fragment translation process, all these changes also directly benefited the MathDox Player itself, and where not just solely implemented for the enrichment of context. The implementation of partial MathDox fragment translation could not be resolved in any other way. However, the implementation is only used when required and does not hinder the normal behavior of the MathDox Player.

5.2 DOMAIN MODEL

The domain model contains all sources that belong to a document. As such it contains the theory, symbol and variable graphs and unadapted content. With the notions of the context object and the custom tags we will explain in this section how the domain model is made available for adaptive mathematical documents.

The Document file.
A MathDox page of a context enriched document requires access to the theory, symbol and variable graphs and content. Remember that MathDox code is stateless by nature and is therefore oblivious as to where to find any additional resources. This includes the unique identifiers that tell us which graphs are required and where these are to be found. A *document file* containing document specific information offers a solution. By placing the document file at the root of a document, custom tags know where to find it and are able to look up what they require. The document file shown in Listing 13 holds the URLs to the theory, symbol and variable graphs and also contains other useful information, such as the document identifier and prerequisite knowledge that the documents assume to be already understood.

Dependency graph format.
All graph URLs and the URL of the document file are used as identifiers in the context object. These identifiers are linked to Java structures representing the cached graphs, document settings and prior knowledge. The object structures are then used to execute queries to

gather information from a graph. The JGraphT Application Programming Interface (API) [48] is used for the Java object structures that represent the graphs.

Listing 13: An example of a document file

```
<document docid="idacontext-cp1">
  <url>http://evo02.win.tue.nl/rikkomathdoxplayer/</url>
  <documenthome>/experimental/idacontext</documenthome>
  <theorygraph>http://evo02.win.tue.nl/rikkomathdoxplayer/
      experimental/idacontext/graphs/theorygraph/theorygraph.xml<
      /theorygraph>
  <symbolgraph>http://evo02.win.tue.nl/rikkomathdoxplayer/
      experimental/idacontext/graphs/symbolgraph/symbolgraph.xml<
      /symbolgraph>
  <variablegraph>http://evo02.win.tue.nl/rikkomathdoxplayer/
      experimental/idacontext/graphs/variablegraph/variablegraph.
      xml</variablegraph>

  <prior-knowledge>
    <required-symbols>
      <OMS cd="arith1" name="abs"/>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="arith1" name="minus"/>
      <OMS cd="arith1" name="plus"/>

      <!-- some symbols were deleted for clarity-->

      <OMS cd="setname1" name="Q"/>
      <OMS cd="setname1" name="R"/>
      <OMS cd="setname1" name="Z"/>

      <OMS cd="sequence1" name="sequence"/>

      <OMS cd="transc1" name="ln"/>
    </required-symbols>

    <required-CDs>
      <required-CD cd="prog1"/>
    </required-CDs>
  </prior-knowledge>

  <extraDefinedSymbols>
    <OMS cd="setname1" name="Z"/>
  </extraDefinedSymbols>
</document>
```

In order to translate the GraphML [34] description of a graph to a JGraphT object structure a graph import library was written. The dependency graphs are written in the GraphML format as a collection of nodes with edges to connect them. A feature of the GraphML format and the import library to include optional properties — to be defined for the nodes or edges — may be utilized by authors to add specific extra information into their graphs. These properties then become available in the graph object structure and by means of queries are available for adaptation purposes on a page. Additional properties are not mandatory and author defined. This fits into our philosophy of offering a generic and powerful system for creation of adaptive mathematical documents.

Listing 14: A theory graph description file

```
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">

  <!-- properties -->
  <key id="oms" for="node" attr.name="oms" attr.type="string"/>

  <key id="threshold" for="node" attr.name="threshold" attr.type=
      "string"/>

  <key id="indexdata" for="node" attr.name="indexdata" attr.type=
      "string"/>

  <key id="title" for="node" attr.name="title" attr.type="string"
      />

  <key id="knowledgesummerize" for="node" attr.name="
      knowledgesummerize" attr.type="string"/>

  <graph edgedefault="directed">
    <!-- nodes -->

    <!-- s1  -->
    <node id="s1p1">
      <data key="title">Division</data>
      <data key="threshold">1</data>
      <data key="oms">
        <symbols>
          <OMS cd="integer1" name="quotient"/>
          <OMS cd="integer1" name="factorof"/>
        </symbols>
      </data>
      <data key="indexdata">a,b,c,d</data>
      <data key="knowledgesummerize">fragments/s1p1/
          theorysummerize.mdf</data>
    </node>

    <node id="s1p2">
      <data key="title">Quotient</data>
      <data key="indexdata">b,d</data>
      <data key="threshold">3</data>
      <data key="knowledgesummerize">fragments/s1p2/
          theorysummerize.mdf</data>
    </node>

    <node id="s1p3">
      <data key="title">remainder</data>
      <data key="indexdata">c,d</data>
      <data key="threshold">3</data>
      <data key="oms">
        <OMS cd="integer1" name="remainder"/>
      </data>
      <data key="knowledgesummerize">fragments/s1p3/
          theorysummerize.mdf</data>
    </node>


    <!-- for clarity reasons other nodes have been omitted from
        this example -->


    <!-- edges -->
    <edge id="s1p1—s1p3" source="s1p1" target="s1p3"/>
    <edge id="s1p1—s1p2" source="s1p1" target="s1p2"/>
    <edge id="s1p1—s1p5" source="s1p1" target="s1p5"/>
    <edge id="s1p1—s4p1" source="s1p1" target="s4p1"/>
    <edge id="s1p2—s1p4" source="s1p2" target="s1p4"/>
    <!-- edges removed from example for clarity reasons -->
  </graph>
</graphml>
```

Example 11:

A theory graph is listed in Listing 14, in Figure 23 the same graph is visualized. This theory graph describes the partial ordering between mathematical notions of the realm of group theory. The format we used to describe the graph is GraphML, a rather common format to describe graphs in XML. The way GraphML is structured is first to declare properties that might exist for edges or nodes. These are named by the *id* attribute. The *for* attribute tells if the property is associated with either nodes or edges. The type of value the property holds can be specified with the *attr.type* attribute. Each property can be fitted with a default value. The default value will be used if a node or edge does not specify a value for that property.

The nodes and edges themselves need to be declared within the *graph* section. This *graph* tag takes an attribute *edgedefault* in which we specify that the graph is directed. The nodes and edges each take an *id* that uniquely identifies the node or edge. If there are no properties that apply to a node then nothing more than just the tag with the id is required. An edge obviously also requires a source

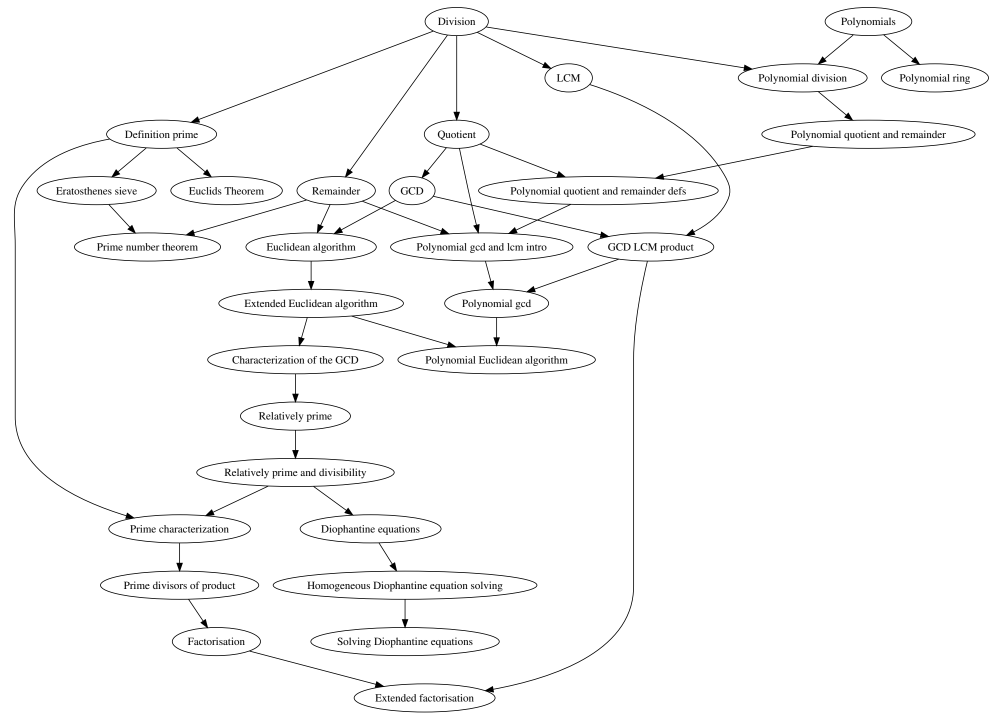

Division
Polynomials
LCM
Polynomial division
Polynomial ring
Definition prime
Quotient
Polynomial quotient and remainder
Eratosthenes sieve
Euclids Theorem
Remainder
GCD
Polynomial quotient and remainder defs
Prime number theorem
Euclidean algorithm
Polynomial gcd and lcm intro
GCD LCM product
Extended Euclidean algorithm
Polynomial gcd
Characterization of the GCD
Polynomial Euclidean algorithm
Relatively prime
Relatively prime and divisibility
Prime characterization
Diophantine equations
Prime divisors of product
Homogeneous Diophantine equation solving
Factorisation
Solving Diophantine equations
Extended factorisation

Figure 23: A visual representation of the theory graph.

to start from and a target to go to. These are listed as attributes with node ids as value.

5.2.1 *Theory graph*

A theory graph, as previously discussed in Section 4.1.1, is a dependency graph that defines the partial ordering between the different mathematical notions, called theories. The relation $\sqsubseteq_T$ between theories is captured in the theory graph described in an XML file, and loaded into the context object. An example of a theory graph was given in Example 6.

5.2.2 *Symbol graph*

The symbol graph has similarities with the theory graph for instance, both graphs are used for overlay purposes. Additionally, the theory graph also refers to the symbol graph as it keeps track on which nodal page a symbol is introduced.

This relation with the theory graph could be made more explicit by applying the partial ordering of the theory graph to the symbols as well, effectively merging the symbol graph into the theory graph. However, it is preferable to keep the theory and symbol graph separate due to separation of responsibility. This helps to keep the graphs less complicated and reduces the chance of error. Reuse of graphs is encouraged, when little changes have to be made, it is easier to replace just one graph and keep the other one unchanged, than replacing a combined graph for a new one. Finally, finding the required predecessors is easier and far more intuitive with the partial ordering of the symbols in a separate graph.

Listing 15: Listing of a symbol graph

```
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph edgedefault="directed">
    <!-- nodes -->
    <node id="arith1.gcd">
      <data key="title">GCD</data>
      <data key="representation">title</data>
      <data key="page">document</data>
    </node>

    <node id="arith1.lcm">
      <data key="title">LCM</data>
      <data key="representation">title</data>
      <data key="page">document</data>
    </node>

    <!-- edges -->
    <edge id="factorof-lcm" source="integer1.factorof" target="
        arith1.lcm"/>
    <edge id="factorof-gcd" source="integer1.factorof" target="
        arith1.gcd"/>
    <edge id="factorof-P" source="integer1.factorof" target="
        setname1.P"/>

  </graph>
</graphml>
```

Variations in symbol graphs.
Just like the theory graph, the symbol graph is created by the author of a document. It is likely that other authors take different approaches to the same topic. This is because their documents may be aimed at different audiences or simply because they take a different view on the matter at hand and explain things differently. This is a situation very similar to the one we already discussed in Section 4.1.1 about theory graphs. The distinction here however is that, whereas the theory graph uses author defined nodes for the graph, the symbol graph uses OpenMath symbols. OpenMath has a large collection of predefined and standardized symbols and as such these are suitable to be shared between different symbol graphs, as opposed to the more freely defined theory nodes. If an author instead opts for symbols that are not (yet) part of this standardized set, full cooperation between different symbol graphs becomes hard. Sharing knowledge of

symbols between documents reduces the need for explicit data harvesting from the user, but requires privacy permission.

Another similarity with the theory graph is the way extra properties can be added to the nodes and edges of the symbol graph. Just like we discussed at the start of Section 5.2.

In Listing 15, the listing of a partial symbol graph XML file is shown. Note that this file is structured in the same way as a theory graph, as discussed in Section 5.2.1. This includes the way properties are added to either the nodes or edges. The location of the symbol graph file is mentioned in the document file.

5.2.3 *Variable graph*

The variable graph indicates the dependency relations between the various variables, again this is done by a GraphML file of the graph, which is — just like the theory and symbol graph — loaded into the context object.

The variable graph holds the declaration of the variables, meaning all relevant data for a variable except for its value. The variable graph does not record the values chosen by the users or the calculated derivatives, that is the job of the mathematical context, a sub-model of the user model, see Section 5.3

Properties in the variable graph.
Just as in the theory and symbol graph, the information stored in the variable graph can be extended. If an author feels a variable needs extra information, he or she can add this as extra attributes to the variable vertices, or the connecting edges. This extra information can later be used to fine tune content adaptation to the user, or for analysis by the author. Some predefined attributes that have a meaning to variable nodes are *om*, *input*, *type*, *cas*, *xpathcondition* and *omcondition*. We discuss these properties as used in the variable graph.

Listing 16: A partial variable graph

```
<graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi
    ="http://www.w3.org/2001/XMLSchema-instance" xmlns:mdcv="http
    ://mathdox.org/mathdoxcontext/variables" xsi:schemaLocation="
    http://graphml.graphdrawing.org/xmlns http://graphml.
    graphdrawing.org/xmlns/1.0/graphml.xsd">

  <key id="OMExpression" for="node" attr.name="OMExpression" attr
      .type="xml"/>
  <key id="type" for="node" attr.name="type" attr.type="xml"/>
  <key id="conditions" for="node" attr.name="conditions" attr.
      type="xml"/>
  <key id="inputNode" for="node" attr.name="inputNode" attr.type
      ="boolean">
    <default>false</default>
  </key>
  <key id="cas" for="node" attr.name="cas" attr.type="boolean">
    <default>gap</default>
  </key>
  <key id="edgetype" for="edge" attr.name="edgetype" attr.type="string">
    <default>dependency</default>
  </key>

  <graph edgedefault="directed">
    <node id="s1p1.a">
      <data key="cas">gap</data>
      <data key="inputNode">true</data>
      <data key="conditions">
        <conditions>
          <condition type="xpath">
            <xpath>not(//OMS[@cd="arith"])</xpath>
            <message>no symbols from arith1 dictionary are allowed</message>
          </condition>
          <condition type="cas" cas="gap">
            <omexpr>
              <OMA>
                <OMS cd='logic1' name='and' />
                <OMA>
                  <OMS cd='relation1' name='leq'/>
                  <mdcv:variable/>
                  <OMI>100</OMI>
                </OMA>
                <OMA>
                  <OMS cd='relation1' name='geq'/>
                  <mdcv:variable/>
                  <OMI>25</OMI>
                </OMA>
              </OMA>
            </omexpr>
            <message>A must be inbetween 2 and 10!</message>
          </condition>
        </conditions>
      </data>
    </node>

    <!-- other nodes are not shown -->
    <!-- the variable graph has no edges defined -->
  </graph>
</graphml>
```

In Figure 24, a variable graph is shown associated with the running example of the orbit stabilizer theorem as previously discussed in Examples 7 and 16. Figure 24 was also shown earlier in Chapter 4 as Figure 12, but is repeated here for convenience of the reader.

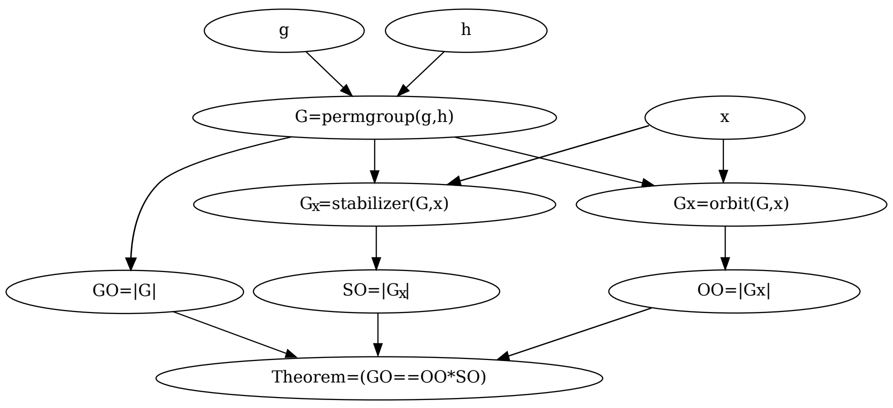


Figure 24: A component of a variable graph related to orbits

5.2.3.1 *The om attribute*

The *om* attribute is a required attribute used for assigning an OpenMath expression to a variable. Such an OpenMath expression can be anything from a value to a formula that depends on other variables before it can be evaluated.

5.2.3.2 *The input attribute*

Because nodes from the variable graph depend on each other, the root nodes have a lot of influence on the values associated with the other nodes in the graph. For instance, consider the group G that is defined as the group that is generated by the permutations g and h. Because G is defined by these permutations, the two permutations will always be elements of G. As such both the permutations and G, and other successor nodes in the variable graph share a dependency relation with the first picked permutations g and h. Note that if the values of one or more of the non-root variables (like G, Gx, GO, OO and others) are changed, instead of the values of g, h and x, then the relation between the variables in the variable graph is lost. As a result the *Orbit stabilizer theorem* does not necessary hold any longer. These situations must not occur.

Two kinds of variables.
This demonstrates that there are two kinds of variables in the variable graph. The first kind are those of which a user is allowed to change the value without breaking the defined relations between the variables. These variables are called *input variables* and are the roots of the variable graph. In Example 7 the variables g, h, and x are input variables. All other variables are called *function variables*; these should not, and therefore cannot, be directly changed by a user. The variable graph does not keep track of values assigned to variables by the users but it can indicate which variables are suitable for value assignment by the user and which are not.

Definition of functional variables.
Only variables f with $x \sqsubset_V f$ are considered to be *functional variables*. Furthermore there is a mathematical expression determining the value of f uniquely, once all variables occurring in it have been assigned a value.

Note, when all variables used in a functional variable f have a value, it necessarily follows that the same must recursively be true for all variables that are used in f.

Definition input variables.
An input variable x is characterized by the absence of any dependencies of the form $y \sqsubseteq_V x$. In particular it does not have a definition. Therefore input variables need to obtain their values either as default values from the author or have them to be assigned by the user. These assigned values are then stored in the *user model* (see Section 5.3).

The input attribute.
The input attribute as given to a variable node from the variable graph, tells the adaptive mathematical document which variables are considered input variables. All variables that do not have the *input* attribute set to true, are automatically considered functional variables.

Input variables without values.
If an input variable has no value, then no value can be assigned to functional variables depending on it. This is a situation that is prevented by the author by supplying default values for all input variables and demand valid new values when these are changed. See Section 5.2.3.4 about conditions.

5.2.3.3 *The cas attribute*

The *cas* attribute stands for which computer algebra system needs to be used to evaluate the variable. While it is usually a computer algebra system that performs computations, any mathematical service can be used as long as it is able to understand the MONET [70] protocol, typically provided by phrasebooks [142]. This attribute only applies to functional variables, as input variables do not require evaluation by computer algebra systems.

Using different computer algebra systems allows one to use the best computer algebra system available for the kind of computation required. Any computer algebra system used in this way needs to be defined in the properties file of the MathDox Player along with a URL where its phrasebook can be found. If the *cas* attribute is omitted the default of MathDox Player is used as specified in the properties file of the MathDox Player. As can be seen in Listing 16 a default value for a cas can be given in the variable graph. In Listing 16 this is the gap phrasebook.

5.2.3.4 *Conditions*

Conditions enable the checking of values of the variables as some values might be considered illegal or undesired. For example a division by zero, a square root from a negative value, or any other value that might lead to unwanted or illegal situations. Illegal situations do not have to appear in the same variable, if the value is used to determine the value of a functional variable problems may occur there or even in other functional variables later.

In the MathDox context implementation, we have created two different types of conditions. An XPath condition, that needs to be applied upon the OpenMath expression of the variable, and a computer algebra system condition (cas condition). As the name suggests the later is a condition that includes the OpenMath expression from the variable and is sent to a computer algebra system for evaluation. XPath conditions are discussed later, first cas conditions are explored a bit further. In Figure 25 a variable graph is shown that also makes use of XPath and OpenMath conditions.

Cas conditions.
Looking at the OpenMath XML is not always enough to know whether the evaluated value it represents is within allowed value ranges. The most practical way to verify whether value conditions are met, is to send the OpenMath expression together with the conditions of to a computer algebra system, and verify that the returning answer is satisfactory.

If a cas condition is used, the expression needs to be constructed in such a way that the computer algebra system returns an OpenMath true or false value. This is rather easy to manage with the use of OpenMath symbols such as *lesser than*, *equal* to or *greater than*. It is not hard to expand conditions to suit the required needs with logical operators as *and*, *or* and *not*. The value of the current variable is included in a cas condition by the use of the *this* tag.

Note that although it would make sense to create cas conditions only for the *input variables* in such a way that functional variables can no longer produce illegal values in a variable graph, conditions can also be attached to functional variables. This gives an author the freedom to choose whether he or she decides to check at the gate (at the input variable level) or wants to warn a reader for an illegal situation at the moment it occurs and not before. Conditions at functional variables also encourage reuse of parts of the variable graph. XPath conditions are less useful for functional variables as their values are computed and not given as input by the user. An exception occurs when a functional variable is being reused as an input variable.

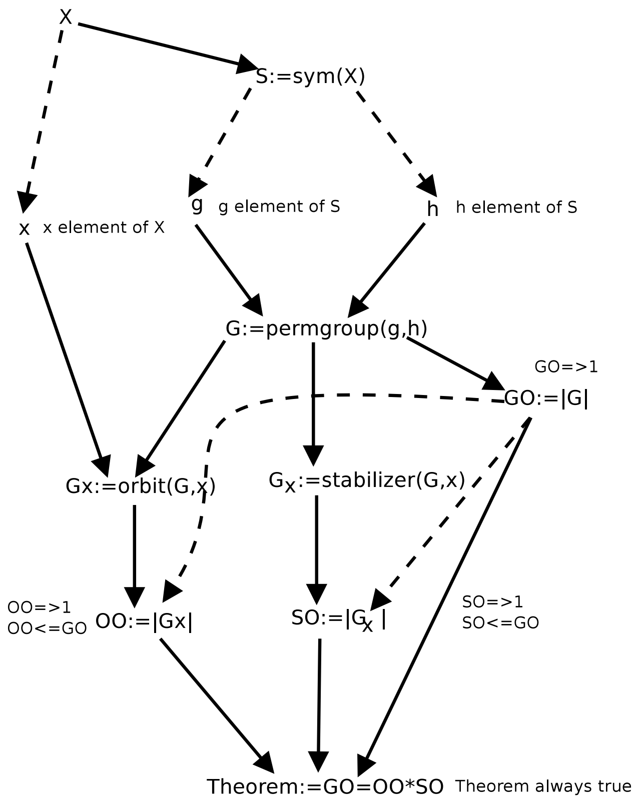


Figure 25: A variable graph related to orbits with conditions

Example 12:

In Figure 24 a variable graph is shown that is used to connect some variables regarding permutation groups, orbits and stabilizers. It did however not show any conditions. In Figure 25 the same variable graph is shown again, now with added conditions.

As was already clear from the previous example g and h both need to be permutations. The element that functions as starting point for the orbit and stabilizer algorithm however also needs to occur in either g or h or both. A good way to guarantee proper values for all input variables, is to apply conditions that bind them all to the same set X. Making sure that g and h are indeed permutations containing only elements from the set X we insist that each of them is an element from the symmetric group Sym(X) as this group will hold all permutations possible for the set X. This reflects in the variable graph as two extra variables X and S as compared to Figure 24. The added variables X and S also have a dependency as S depends on X.

Condition dependencies.
It would be logical to expect dependency relations also to exist between S and X on one side and g, h and x on the other side. However, g, h, and x are *input variables* and as such they have no formula that determines their value based on dependencies on either X or S. Still these input variables are subject to conditions. In this case the condition that these values have to be elements from either X for x or S for g and h. Condition dependencies are represented by a dotted edge in the variable graph. The conditions themselves are properties of the variables nodes and as such linked to the variables to which they apply. Input variables that are connected to a condition dependency edge are still available for input, however this input is subject to conditions that need the variable at the other side of the edge for verification of the condition. In the current example this would result in a situation that a user is able to specify any value for these input variables as long as it is an element of the aforementioned sets.

Among other conditions that apply to the variable graph in Figure 25, there are restrictions on the order of the values of variables GO, OO and SO. These cannot be negative and OO and SO cannot be any larger than GO. Also, it will be clear that the graph is invalid if $\mathsf{Theorem}$ ever holds a false value instead of true. Note however that these conditions will always be true provided correct values were chosen for the input variables.

Conditions contaminate content.
With restrictions put on the variables, one could say that the variable graph is crossing the boundary from being a graph that plays a supporting role for content to becoming content itself. After all with severe restrictions on the conditions, the users' freedom to choose those values is heavily restricted. However, if the author makes these decisions there are probably some good reasons for it. That being said, we consider it a best practice, that an author should limit the use of conditions to the level that is needed to warrant correctness within the document. Using more conditions than whats needed for document correctness would also limit the freedom of users to explore and discover how expressions and values relate to each other.

XPath conditions.
An XPath condition does not try to compute a condition like a cas condition, instead it utilizes an XPath expression to query the OpenMath value. In both the XPath and cas condition types, the result needs to be a boolean true value for the condition to pass. In case of XPath this means either a direct *true* value is required as result, or a non-empty data set as selected by the XPath expression, which defaults to a true value.

An XPath condition only checks for the syntax of OpenMath expressions, as all functional variables have a set syntax it is of little use to apply XPath conditions on functional variables to verify user input. However XPath conditions may be used to verify the type of a value, i.e.. whether a value is an integer or a real.

XPath conditions on input variables can be used to check for the type of an expression, or for the use of certain (forbidden) operators. An XPath condition might check if an OpenMath expression does not contain any elements the author would have disapproved of. For instance if the author requires an integer, he or she may have reasons to really want an integer, and not just an expression that results in an integer.

Multiple cas and XPath conditions can be used on one variable. Providing a mixture of conditions to verify the correctness of the value.

5.2.4 *Content*

Besides the theory, symbol and variable graphs, the domain model also harbors content. The presentation model will make a selection from the content based upon the needs of a user. Since the needs of different users vary, the content they get served needs to change accordingly. The content stored in the domain model is fragmented. The nodal pages that represent the theories from the theory graph are without any content themselves. The content that they present to the user is formed by the included fragments.

In this section we discuss fragments and the way they are implemented in detail. The MathDox format that is used to write the content was discussed earlier in Section 2.3.

Fragments in other formats.
The principle of distributing content over multiple files and taking them together by including them into one file is already quite well known. Indeed content can be reused at multiple locations while it only needs to be written once. This technique is used in LaTeX, but also in programming environments like XInclude [117], PHP [84], JSP [43], and many others. They all focus on content inclusion and convert the original file enriched with all inclusions to a virtual file with the complete contents. This is in contrast to object oriented languages, where the use of objects leads to inclusion of functionality instead of a verbatim copy of the code. The verbatim inclusion of fragment code means that an author needs to know how things are done in a fragment. Similar to LaTeXwhere authors really do need to know what kind of environments are used in the included parts, or which variables are defined in included fragments in JSP, in order to prevent collisions with these declarations. In our mathematical model implementation in MathDox, we aim for more than just inclusion of the literal content, we want to include the functionality offered by the fragments. Wherever a fragment is included a separate entity with its own scope will be used. This will prevent undesired data collisions

and warrant its functionality regardless of the contents of the environment it is included into, be it in the file itself or in other fragments already included.

Characteristic fragment properties.
A MathDox fragment is a piece of MathDox code that is stored in a separate file and has the properties of being reusable, easy to include and to exchange operational data. There is no set limit to the depth of (recursive) fragment inclusion.

Reusable.
Fragments are designed to be included by other MathDox code. This mechanism of including fragments, makes reuse of the same content on different pages easy. Whenever rules of the presentation model tell a page to include a definition about group theory, all a page has to do is include the fragment that contains this definition. This allows content to be written once and reoccur on each and every single page that is in need of it.

The responsibility to preserve the look and feel of a page, especially at the seams of used fragments lies with the caller of these fragments. This caller may be another fragment or the page itself. It is up to the code that performs the inclusion to make the content seams of included content as smooth as possible and prevent a mosaic look and feel. As the including code is aware which fragment it is including, while the included fragment is not aware which code it is included from. In this regard it is important that the content in a fragment does not assume that the user already possess any information or knowledge other than what is directly needed for the topic of the included fragment. Failure to adhere to this, quickly reduces the reusability of a fragment both in content of the fragment itself but also in relation to fitting it in with the other content on the page.

Also, in order to encourage reusability a fragment must be easy to call and include, and should not force the author into learning any of its inner workings. Keeping the content of a fragment small and to the point, and allowing configuration parameters will benefit the reusability of fragments greatly. Some documentation explaining what the fragment and the configuration parameters do, should be enough.

Easy to include and to exchange operational data with.
Inclusion of a fragment is as easy as a function call in a programming language. To form one continuous MathDox page it is important for a page to be able to exchange operational data with the content/code of the fragment. The separation of scopes prevents data collisions but also direct data access. The solution comes in the form of having a fragment accept arguments and to letting it be capable of returning any useful results from its execution.

Fragments can include other fragments and create compound fragments, combining the content and behavior of smaller fragments into one larger fragment. A larger compound fragment is less suitable for reuse, as it will contain more rules that might conflict upon reuse. The view that is offered by a compound fragment can be considered one of many. This means that there is no objection to create another compound fragment that offers a different view on the same topic. These other compound fragments can still include the same fragments as the original compound fragment did, but can also deviate by including extra content or fragments, reconfiguring fragments by adjusting their configuration parameters or by omitting content that is not required. The creation of a new compound fragment allows for a new ordering and reuse of the included fragments without the need to rewrite them.

Scope and encapsulation.
As fragments can be included by one another the fragment itself remains unaware of such embedding and remains responsible for its correct functioning. Fragments therefore need a mechanism to shield their properties — such as Jelly variables and XForms identifiers — for outside use. This phenomenon is called *encapsulation* in object oriented programming. Encapsulation for fragments is achieved by adding a Jelly scope to protect variables, and a renaming mechanism for XForms identifiers.

Rule implementing fragments.
In our design a fragment is a piece of MathDox code. MathDox code already has the possibility to make choices based on user interaction. The possibility to construct fragments and include these on pages, results in having the possibility of including decision making algorithms. It is this programmable behavior that allows cyclic or recursive inclusions of fragments. As long as the end condition is well defined there is no risk for an endless recursion. Programmable behavior as described here, lets us use fragments as rules as described in the presentation model in Section 4.3. In Section 5.4 we go into deeper detail.

Fragment implementation.
Implementation of fragments is done with the help of custom Jelly tags which can be called from MathDox code. This means that by giving the right parameters to a Jelly fragment tag, the MathDox fragment file is located and executed. However, consider the MathDox translation pipeline as previously shown in Figure 3 and Figure 20 and also in Figure 26 where it is expanded by the translation pipeline as needed for fragments. As can be seen from these figures, when any Jelly tag is executed, all macros and MONET code present anywhere in code is already translated. As a matter of fact MONET code is translated to Jelly code and macros can be used by authors as substitution

for both MONET and Jelly code. Having a Jelly tag that acts as a hook for fragment code therefore presents a problem as the code within the fragment may still contain MONET code and macros. Furthermore it may even contain other fragment calls that contain MONET code and macros as well. Clearly any fragment code that is encountered at the Jelly phase needs to have their MONET code and macros translated first before the fragment can be executed. To this purpose the MathDox Player — on a rare occasion — was slightly altered. Whenever the MathDox player receives a request for a file in XML format that contains the *<fragment>* root element, it only executes the first two steps of the translation process and renames all XForms identifiers to prevent name clashes later on. The intermediate result of the fragment is then included into the calling code at the execution Jelly step. From this point all content exists in the same (virtual) file and is ready to be processed by the remaining translation steps that transforms DocBook, OpenMath and XForms to HTML. Note that this process also copes with recursive fragment calls. This process is shown in Figure 26.

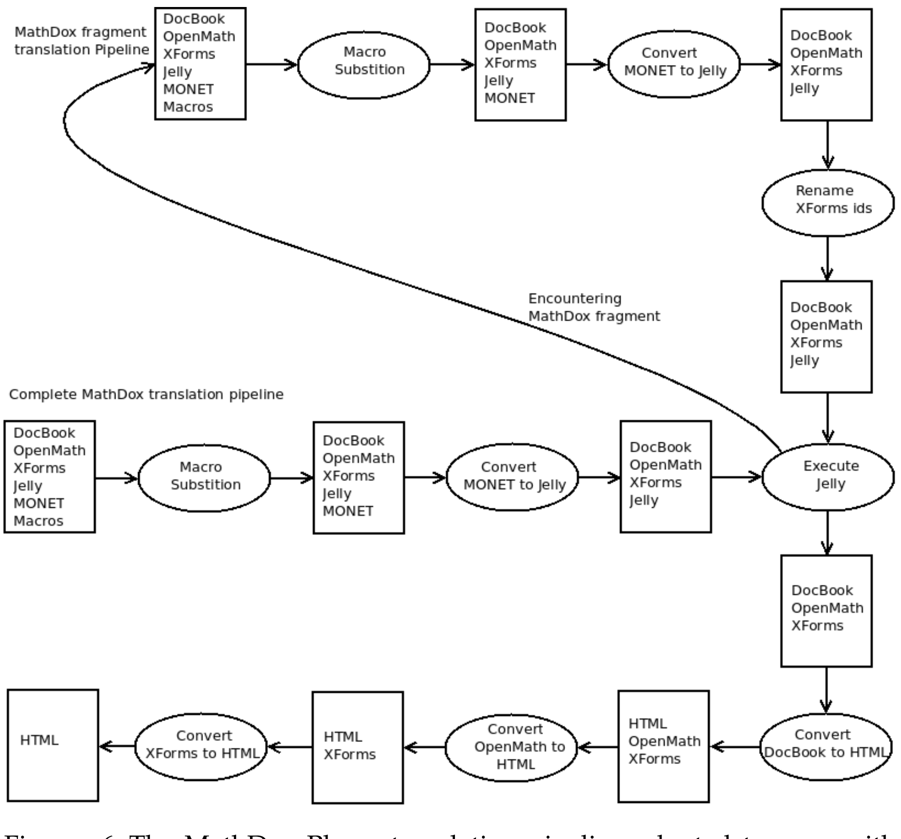

MathDox fragment translation Pipeline
DocBook OpenMath XForms Jelly MONET Macros
Macro Substition
DocBook OpenMath XForms Jelly MONET
Convert MONET to Jelly
DocBook OpenMath XForms Jelly
Rename XForms ids
DocBook OpenMath XForms Jelly
Encountering MathDox fragment
Complete MathDox translation pipeline
DocBook OpenMath XForms Jelly MONET Macros
Macro Substition
DocBook OpenMath XForms Jelly MONET
Convert MONET to Jelly
DocBook OpenMath XForms Jelly
Execute Jelly
DocBook OpenMath XForms
Convert DocBook to HTML
HTML OpenMath XForms
Convert OpenMath to HTML
HTML XForms
Convert XForms to HTML
HTML

Figure 26: The MathDox Player translation pipeline adapted to cope with fragments.

Data within fragments.

Among the responsibilities of the Jelly fragment tag is encapsulation of data within the fragment. The protection of Jelly variables as they are declared inside fragments is rather easily solved by Jelly. The Jelly format offers the creation of an extra scope [160] in which the Jelly code of the fragment is executed. As a result of the extra scope, Jelly code can access all Jelly variables from the calling MathDox code, but the calling MathDox code cannot access any Jelly variables from inside the fragment. In case of variables with the same name, the variable within the current scope takes precedence avoiding the use of variables that where declared elsewhere. This very strongly resembles to how program languages handle variables declared inside a block i.e.. a while or for block as opposed to their availability outside that block.

Encapsulation has it that the calling code should not be required to know about any variables of the fragment and vice versa. For this reason the fragment tag as used in MathDox code accepts a list of name value pairs acting as (configuration) parameters for the fragment, rather than having a fragment accessing variables outside its current scope to obtain this configuration data. This list of name value pairs is converted to a set of Jelly variables within the scope of the fragment called. Alternatively new variables can also be declared and set in the body of the fragment tag. Again these variables are set in the scope of the fragment and therefore unreachable from the MathDox code that is calling the fragment.

As a fragment needs parameters from the calling MathDox code to adapt itself to its environment, the calling MathDox code may need a result variable as parameter from the fragment. This works by a list of name value pairs that is assigned to a variable identifier given to the fragment as an attribute setting in the fragment tag.

Listing 17: MathDox code calling a fragment

```
<c:set c="jelly:core" var="a">16</c:set>

<mdc:include-fragment xmlns:mdc="jelly:org.mathdox.context.
    TheoryBlocksTagLibrary" fragment="fragments/example.mdf"/>

${a}  <!-- will print the value of the variable-->
```

Listing 18: The fragment that is called

```
<mdc:fragment xmlns:mdc="jelly:org.mathdox.context.
    TheoryBlocksTagLibrary">

  <c:set var="a">32</c:set>

</mdc:fragment>
```

Example 13:

Consider the code examples in Listing 17 and Listing 18. In Listing 17 is an example of a fragment that is being called, while in Listing 18 the content of the called fragment in shown.

As both snippets of code have a declaration of a Jelly variable called *a*, one has to wonder what exactly will happen if these statements are joined into one (virtual) file. Indeed in a similar format as JSP [43] both declared variables are considered to be the very same. We want an inclusion of a fragment to be as easy as a call to a procedure in a modern programming language. In order to achieve this goal we need the two different declarations of the variable *a* to remain different variables. The extra scope in Jelly will make all variables as declared inside the fragment invalid to the code outside of the fragment. Variables declared outside the fragment can be accessed in the fragment code, but only if their name is not being overwritten by new declarations. The Jelly statements within the fragment will immediately be executed during this inclusion. The print statement in Listing 17 will print the number 16 even though the variable *a* has a different value inside the fragment.

Listing 19: The include fragment tag

```
<c:new xmlns:c="jelly:core" var="argumentsList"
       className="java.util.HashMap" />
<c:set xmlns:c="jelly:core" target="${argumentsList}"
       property="argument1" value="value1" />
<c:set xmlns:c="jelly:core" target="${argumentsList}"
       property="argument2" value="value2" />

<mdc:include-fragment
  xmlns:mdc="jelly:org.mathdox.context.TheoryBlocksTagLibrary"
  args="${argumentsList}" var="resultList"
  fragment="fragments/example.mdf" />
```

Example 14:

In Listings 19 and 20, MathDox code is shown that demonstrates a fragment inclusion. In Listing 19 a list *argumentsList* is constructed, which is then filled with the name value pairs *argument1/value1* and *argument2/value2*. This list *argumentsList* is then passed to the fragment by means of the attribute *args*. The list with return values is stored in the Jelly variable *resultList* as passed along in the attribute *var*. The fragment that is to be included is being identified with the URL in the attribute *fragment*. Note that the fragment url points to a file that is served by the MathDox Player; the MathDox Player is going to perform the required translation steps upon the request of this fragment and will return MathDox code without any macros or MONET expressions.

Listing 20: The include fragment tag

```
<mdc:include-fragment
  xmlns:mdc="jelly:org.mathdox.context.TheoryBlocksTagLibrary"
  var="resultList"
  fragment="fragments/example.mdf"
>
  <c:set xmlns:c="jelly:core"
            target="argument1"
            value="value1"
            />
  <c:set xmlns:c="jelly:core"
             target="argument1"
             value="value1"
             />
</mdc:include-fragment>
```

Example 15:
In Figure 20 MathDox code is shown with the same functionality as previously shown in Figure 19. As opposed to the previous example, the parameters that are to be passed to the fragment are no longer inserted into a list of name value pairs. Rather, they are being declared as variables in the body of the *include-fragment* tag. Both approaches are considered identical.

5.3 USER MODEL

A user model is all about the data of the user. It stores raw data that is later used by queries and rules from the presentation model to adapt the content to the needs of the user. To make adaptation by the presentation model work, it is of importance for the user model to have a good data set about each user and to keep this up to date.

Initial data set.
An initial data set about the user may be obtained by presenting the user with a questionnaire. The need and complexity of a questionnaire may vary from different documents. Sometimes it may be best to ask a user for permission to copy relevant data that is already known about the user from other adaptive documents, saving time filling out questionnaires.

Keeping data up to date.
The initial data set needs to be maintained and expanded upon during visits of the user so as to reflect the progress made in the document. The user reads through the document, learns new things, makes changes in values and visits more and more different pages. This is of course expected from the user, and needs to be recorded in the user model so the document takes them into account upon the next visit of the user. Updating the user model for a user can be done implicitly, explicitly, or in a combination of both. Implicit data gathering focuses on retrieving data from the user without him or her being consciously aware of it. As such it may be subject to incorrect assumptions made by the system, introducing a chance on inaccurate user model data. Explicit data gathering is much like the afore mentioned questionnaire, it asks the user to verify an assumption or to enter required data. Explicit data relies on the honesty of the user to provide the correct data. A good design for an adaptive mathematical document, takes a balance between implicit and explicit data gathering and use implicit data to verify the explicit data and vice versa.

History.
As data is being updated it will change, causing the data set of a user over time to migrate from one state to another to yet another. These newer data sets will benefit the user as it leads to a different adapted content, but there is also a danger as changed content might confuse the user. After all if content is adapted it changes in presentation, and the user may not recognize it any more as the same content. Confusion will occur especially in cases where the user wants to look back at things which are no longer there or have been replaced by other fragments of content. The user will need a mechanism to (temporary) roll back any data changes and use the data that has been stored at an earlier point in time, and let the adaptive document reproduce the content exactly as it is was before.

For this purpose each and any change to data in the user model is done by means of information addition. That is, information is not allowed to be replaced as to preserve the original values. Instead, new information is added with a newer time stamp attached to it. In this way, a versioning scheme is employed such that older data stored in the user model is still present. Queries to the user model will need a timestamp as to return the data that was known at that moment.

Values that are randomly picked and influence content need to be stored in the user model, even if they normally would not be. Failure to do so will still lead to different content when the user rolls back to a previous timestamp.

eXist.
The user model is divided into three sub-models; mathematical, logistical and knowledge context. Each has its own responsibility. The

mathematical context keeps track of all mathematical values, logistical context keeps track of non-mathematical values and the knowledge model keeps track of which notions are understood. These are being kept in the XML-database eXist [22]. An XML-database is especially useful for the mathematical context as the mathematical expressions are written in OpenMath, an XML format. Due to the XML nature of the database XQuery [127] and XPath [145] are useful tools in retrieving information from the database.

Mathematical and logistic context.
Variables are used throughout an adaptive mathematical document, and can be categorized into either mathematical or logistic (non-mathematical) variables. Besides the math - non-math difference, there is also a difference in scope. Logistic variables must be accessible from anywhere in the document, while mathematical variables should only be accessible in the nodal pages they are introduced at or in any successor nodes. As a consequence variables are grouped in symbol tables, one for each nodalpage and one for all logistic variables. Variables listed in the same symboltable have the same scope in the document. Also, assigning each nodal page a symbol table, will grant a page the freedom to name its variables without any name clashes.

Listing 21: Mathematical context as stored in the user model

```
<variables>
  <document docid="idacontext—cp1">
    <symboltable name="s1p1">
      <variable name="a" date="20150426154909" type="openMath"
          vargraphvar="true">
        <OMI>12</OMI>
      </variable>
      <variable name="a" type="openmath" date="20150428094655"
          vargraphvar="true">
        <OMI>8</OMI>
      </variable>
      <variable name="b" date="20150426154909" type="openMath"
          vargraphvar="true">
        <OMI>4</OMI>
      </variable>
      <variable name="c" date="20150426154909" type="openMath"
          vargraphvar="true">
        <OMI>14</OMI>
      </variable>
      <variable name="c" type="openmath" date="20150428094657"
          vargraphvar="true">
        <OMI>12</OMI>
      </variable>
      <variable name="d" date="20150426154909" type="openMath"
          vargraphvar="true">
        <OMI>7</OMI>
      </variable>
      <variable name="ab_q" type="openMath" date="20150426201453" vargraphvar="true">
        <OMI>3</OMI>
      </variable>
      <variable name="cd_q" type="openMath" date="20150427161733" vargraphvar="true">
        <OMI>2</OMI>
      </variable>

      <!-- other variables entries have been removed for clarity -->
    </symboltable>

    <!-- removed others symboltables for clarity -->
    <symboltable name="root">
      <variable name="firstVisit" date="20150427112131" type="text" vargraphvar="false">true</variable>
      <variable name="showProof" date="20150426154909" type="text" vargraphvar="false">true</variable>
      <variable name="lastName" date="20150426154909" type="text" vargraphvar="false"/>
      <variable name="goal" date="20150426154909" type="text" vargraphvar="false">nogoal</variable>
      <variable name="dateOfBirth" date="20150426154909" type="text" vargraphvar="false"/>
      <variable name="minimizeQuestions" date="20150426154909" type="text" vargraphvar="false">true</variable>
      <variable name="firstName" date="20150426154909" type="text" vargraphvar="false"/>
      <variable name="date" date="20150426154909" type="text" vargraphvar="false">20150427000000</variable>
    </symboltable>
  </document>
</variables>
```

Consider Figure 21, it displays a part of a user model serialized as an XML file. Each user has its own document containing the variables from both the mathematical as logistic context. In Figure 21, a part of the variables that belong to a user is shown. As can be seen by the document tag all variables from different adaptive mathematical documents are stored in the same file, but in a different branch of the XML code. A document is identified with a document identifier as mentioned in the document file discussed in Section 5.2. Figure 21 also shows that each symboltable is named after its nodalpage and stores the variables that are introduced on that nodal page. The exception to this rule is the *root* symboltable. The root symboltable holds all logistic variables. A third kind of symbol tables are local nodal symbol tables which are used for local variables that should only be used by the nodal page itself.

The symbol table name (prefix) and the variable name (suffix) compose a variable identifier for use outside the symbol table. Because the symboltable name is used as a prefix, it is impossible for two variables from different symbol tables to coexist with the same name. Note that determining all variables available to a nodal page connected to theory t now results in collecting all variables stored in the symbol tables connected to the nodal pages of the preceding theories. This is in line with the definition given in Section 4.1.4, where we described all mastered predecessor theories as $\Delta(t)$ and $\Omega_V(t)$ represents all variables available for use in the nodal page associated with t. Therefore the lemma: $\Sigma_V(t) \subseteq \Omega_V(t)$ as stated in Section 4.1.4 and where $\Sigma_V(t)$ represents the variables actually used on a nodal page will be true, provided that $\Omega_V(t)$ also includes the logistic variables declared in the root symboltable.

The variables stored inside a symboltable contain a number of attributes. First, there is the *name* of the variable, used for identification within the symboltable. Note that variable names in a symboltable are without the symboltable prefix. The variable identifier (the name with the symboltable/nodalpage prefix) is used to look up any special properties tied to this variable in the variable graph. Among these properties may be OpenMath expressions and conditions that apply to the values that a variable can assume. This kind of information may not be changed by the user and is therefore part of the variable graph in the domain model.

The next attribute contains the timestamp as previously discussed in this section. These timestamps take the form of *yyyyMMddhhmmss*. Where *yyyy* stands for the year, *MM* indicates the month and days are given by *dd*. The same goes for *hh* which is the hours in a day, *mm* are the minutes, and finally *ss* seconds. The sequence from large time units to smaller units also helps in sorting, since newer timestamps automatically have a bigger number. Time-stamps of variables are used to select the right value for the variable at some time in the past. This is useful when the user wants to return to a page as it was at an earlier time.

The *openmath* attribute indicates if the variable contains an OpenMath expression. Mathematical variables contain per definition OpenMath expressions, but the same does not necessary hold for another type of variable, like logistical variables as listed in the root symboltable. The *openmath* attribute defaults to true.

Functional variables.
Both input and functional variables are stored in the mathematical context. Variables are assigned values by either the user, the author (in case of default values), or by the document when functional variables are involved since those values need to be calculated on the basis of other variable values. Note that some computations in functional variables can be quite heavy, especially if such a value is used repeatedly. It therefore makes sense to store the values of both input and functional variables, especially as it prevents unnecessary com-

putations for functional variables. Doing so implies the need to keep track when the value of an input variable changes, because it has consequences for the depending functional variables. All functional variables depending on the changed input variable need then to take notice and mark their values as out of date and recompute them. The re-computation only happens when the value of that variable is requested either by the document or by another functional variable that needs it as input.

Logistic context.
Logistic user information typically consist of name-value pairs. As such they are ideally suited to be stored as variables. Instead of being mathematical variables, and therefore containing OpenMath expressions, the values are usually text based. The logistic variables are not bound to a specific nodal page and are therefore stored in the *root* symbol table. Due to the high similarity between logistic information and the mathematical context, it made sense to treat the logistic information in the same way. As a result, logistic information variables have the same attributes as the variables from the mathematical context, with the exception that their names are not identifiers in the variable graph for extra information about the variable. As such, there are no conditions or values to select the right computer algebra system. Note that while logistic information tends to be text, the logistic variables still have the *openmath* attribute and therefore also support OpenMath expressions. Both kinds of variables are accessible within MathDox code by the use of the same *variable* tag.

Local variables.
Looking again at Figure 21, one can find symbol tables named after nodal pages, but also symbol tables with an extra *-local* suffix. These *local* symbol tables store variables that only have the scope of the nodal page they are associated with and are not to be used outside the nodal page. The local symbol table is only accessible to the nodal page it belongs to, and the symbol table will only be part of the collection of symbol tables when its associated nodal page is visited. The use of these local symbol tables gives nodal pages the freedom of using variables that are not used in other expressions outside their own control, on other nodal pages.

Copy of a variable graph.
Local variables stored in symbol tables with a *-local* suffix, are quite suitable for creating copies of variable graphs. Creating a copy of a variable graph comes in handy when new concepts are being demonstrated and different values are needed, and the original relations between the variables have to be preserved. It is required to find those variables within the copy of the variable graph that can serve as new input variables, without losing the relations between the variables of interest. Suitable new input variables are those that can be

altered without creating conflicts elsewhere in the partial variable graph copy.

These new input variables are found by an algorithm that works as follows:

- Make a complete copy of the variable graph, and mark all variables that are used by the nodal page.
- Eliminate all variables that are successors of the required variables, as these are not needed.
- All variables that do not have any outgoing edges, but are still in the same branch with another marked variable will mark their predecessor variables, and remove the connecting edges. This step is repeated until all marked variables are no longer in the same branch of any other marked variable.
- The variables that have been marked, are those that are required in the partial copy of the variable graph. A new copy of the variable graph containing only the marked nodes and their connecting edges can be used as the partial copy that was required.

In Figure 28 one can see the algorithm in action. The new input variables in a fresh (partial) copy of the variable graph are those nodes where the tracing stopped. Note that it is quite possible that some or even all new input variables happen to be the same as in the original variable graph. In such cases the tracing process continued until there were no more predecessors left and a partial copy is not possible.

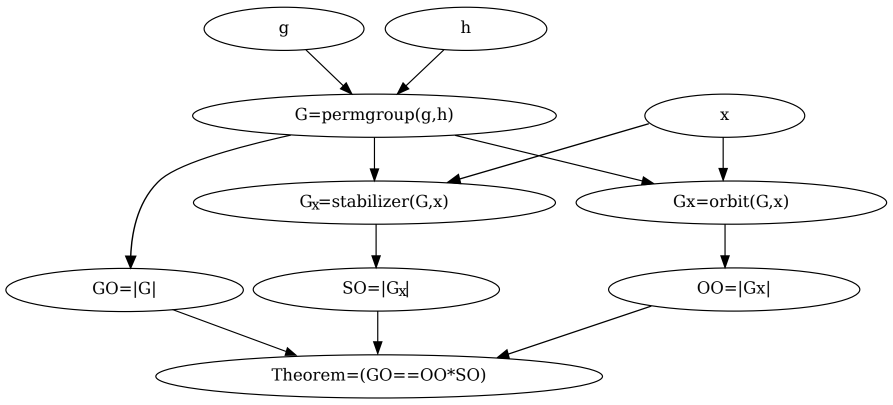


Figure 27: A component of a variable graph related to orbits

Example 16:

Take a look at Figure 27. Here we can see that if a nodal page for the stabilizer theorem wants to use some other values for the variables OO, SO and GO that we need to maintain the relation between them. If three new random values are chosen, there will be no guarantee that the final theorem $GO = OO * SO$ will still be true. The algorithm just described should be applied to preserve this relation. The result of the algorithm is that the variable graph should be copied except for the input nodes g and h. The group G will take their place as the new input node. This means that in the copied variable graph the variables x and G are the ones that can be freely adapted, provided associated conditions are met.

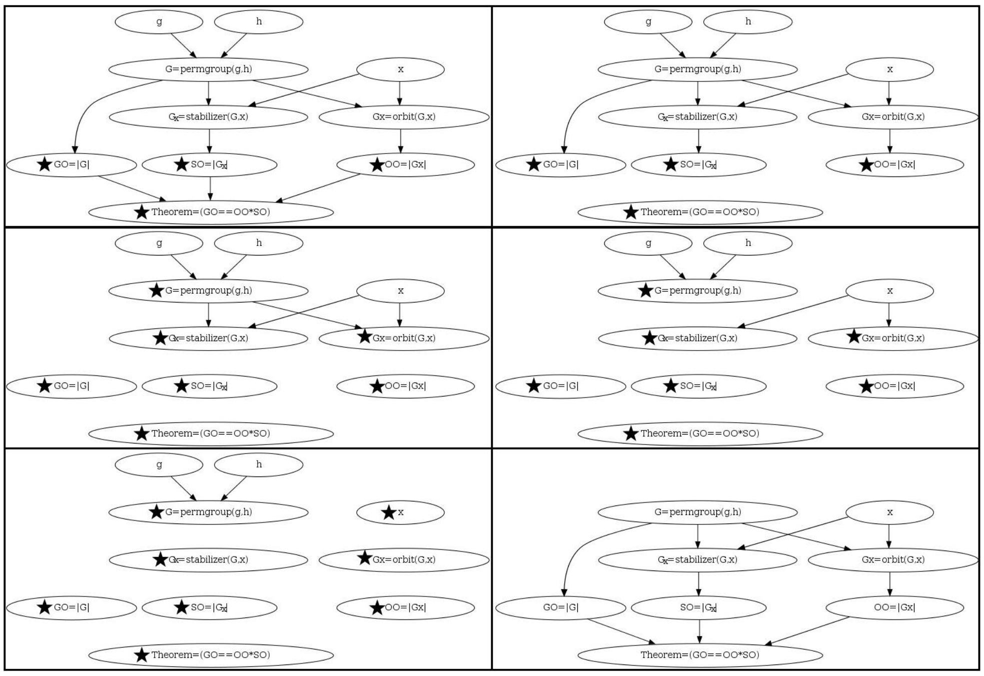


Figure 28: The steps of the algorithm for a partial copy of the variable graphs

Knowledge context.
The overlay model, on which a large part of the adaption relies, requires that the user model keeps track of the mastered knowledge. In our case that means that the knowledge context needs to keep track of nodes of both the theory graph and symbol graph that are considered understood. This can be done in a binary way, that is, either the user has mastered the knowledge or the user has not. Another approach is to award points to represent the degree on which the user has mastered the knowledge.

Both approaches are possible with the same implementation, and it is up to the author to decide which is preferred. In the case of the binary system, any points regarding an item of knowledge indicate that the user has mastered that knowledge, while in the other case the awarded points need to be summed first to see if the user passes a threshold.

An implementation that uses probabilities instead of points was not pursued, but is not hard to implement. Instead of awarding points, the system needs to log probabilities and a rule to interpret these.

Listing 22: A piece of the knowledge information

```
<knowledge>
  <theorygraph>
    <documents>
      <document docid="idacontext-cp1">
        <action actionid="read">
          <theory name="s1p1" date="20150407200429"
            resource="/experimental/idacontext/nodalpages/s1p1/
                index.md" graphURL="http://evo02.win.tue.nl/
                rikkomathdoxplayer/experimental/idacontext/graphs
                /theorygraph/theorygraph.xml" points="1"/>
          <theory name="s5p2" date="20150413091601"
            resource="/experimental/idacontext/nodalpages/s5p2/
                index.md" graphURL="http://evo02.win.tue.nl/
                rikkomathdoxplayer/experimental/idacontext/graphs
                /theorygraph/theorygraph.xml" points="1"/>
          <theory name="s4p3" date="20150413091609"
            resource="/experimental/idacontext/nodalpages/s4p3/
                index.md" graphURL="http://evo02.win.tue.nl/
                rikkomathdoxplayer/experimental/idacontext/graphs
                /theorygraph/theorygraph.xml" points="1"/>
          <!-- other entries have been removed for clarity -->
        </action>

        <action actionid="mastered">
          <theory name="s4p1" date="20150404110455"
            resource="/experimental/idacontext/nodalpages/s4p1/
                index.md" graphURL="http://evo02.win.tue.nl/
                rikkomathdoxplayer/experimental/idacontext/graphs
                /theorygraph/theorygraph.xml"/>
          <theory name="s4p1" date="20150404111315"
            resource="/experimental/idacontext/nodalpages/s4p1/
                index.md" graphURL="http://evo02.win.tue.nl/
                rikkomathdoxplayer/experimental/idacontext/graphs
                /theorygraph/theorygraph.xml"/>
          <!-- other entries have been removed for clarity -->
        </action>

        <action actionid="MyOwnActionIdentifier">
          <theory name="s1p4" date="20150429135515"
            resource="/idacontext/nodalpages/s1p4/knowledgetest.
                md" graphURL="http://evo02.win.tue.nl/
                rikkomathdoxplayer/experimental/idacontext/graphs
                /theorygraph/theorygraph.xml"/>
        </action>
      </document>
    </documents>
  </theorygraph>

  <symbolgraph>
    <documents>
      <document docid="idacontext-cp1">
        <action actionid="mastered">
          <symbol name="integer1.quotient" date="20150427113938"
              resource="/idacontext/nodalpages/s1p3/index.md"
              graphURL="http://dam04.win.tue.nl/
              rikkomathdoxplayer/idacontext/graphs/symbolgraph/
              symbolgraph.xml"/>
        </action>
      </document>
    </documents>
  </symbolgraph>
</knowledge>
```

In Listing 22, an example is given of how the knowledge context is stored. Just like with the logistical and mathematical context, the knowledge context is also stored as an XML document in the XML-database eXist [22]. The knowledge information is captured in the root tag *knowledge*, which contains a *theorygraph* and *symbolgraph* elements. Within these elements there is space for document elements, each with an *docid* attribute referring to the identifier of the document file, so as to prevent confusion between documents. Each document element can hold a multitude of action elements, each identified with an *actionid* attribute. The most common would be the *mastered* and *read* actions. The first being for nodes from the theory or symbol graph that are considered mastered, the second for nodes that have been read, but require rules to determine how much is understood. The author can also define extra actions, like the *MyOwnActionIdentifier*, these do not interfere with the system but can store extra information that can be made available to a document. Each action contains entries for nodes to which the said action applies, these entries are either *symbol* or *theory* elements. An entry will contain the node *name*, a *date*, a *resource*, a *graphURL* and optionally *point*s. Extra parameters are allowed to enable further customization. The name and graphURL together uniquely identify which node from which graph this entry is about. Which nodalpage is responsible for the entry is listed in the resource attribute. The points attribute holds the points awarded by this action. This may be a positive but also be a negative number, or be omitted if it does not make sense for the action.

There is some redundant information in these entries as the graph identifiers are being listed in the document file and are referred to in the stored nodes in the database. However, experience teaches us that graphs and documents may change overtime — especially during initial development — and by keeping track of the graph and resource information it becomes easier to understand why and what has been recorded. Also, when documents share information about a user, they will have access to the information in the database but not necessary to each others document files.

Knowledge context query tags.
In the context implementation some queries have been made available to check the knowledge of the visiting user. In Listing 23, three examples are shown.

The first example checks if the knowledge information has any record of a node with the identifier s2p1. If it does, the MathDox code in the body of the element will be executed.

The second example takes the threshold instead of the binary approach and adds the attribute *threshold* to its call. The threshold attribute takes a number and the code in the body is executed if the sum of the points associated with the node in the theory graph is greater or equal to the threshold parameter.

The third example shows the same for the symbol graph. Note that this is the same tag as used for the theory graph; it is the *type* and *name* attribute that identify the correct node.

Optional attributes are *jelly-variable* and *timestamp*. The *jelly-variable* attribute takes the name of a jelly variable which is going to hold the boolean outcome of the query. The other optional attribute, the *timestamp* attribute, is used to execute the query on the state of the data at the given date. The timestamp setting as stored in the logistical context is used as default.

Listing 23: An example of knowledge information query tags

```
<!-- 1 -->
<mdc:if-knowledge-available type="theorygraph" name="s2p1" >
  <!-- MathDox code -->

</mdc:if-knowledge-available>


<!-- 2-->
<mdc:if-knowledge-available type="theorygraph" name="s2p1"
    threshold="5" >
  <!-- MathDox code -->

</mdc:if-knowledge-available>


<!-- 3 -->
<mdc:if-knowledge-available type="symbolgraph" name="arith1:gcd"
    >
  <!-- MathDox code -->

</mdc:if-knowledge-available>
```

Logging.
Every now and then an adaptive document needs to store logging data. For instance to record when the user accessed a page, or to record answers of the user to questions. This kind of data is not meant to be stored in the knowledge, mathematical or logistic context. To this end a separate folder was added to the database. Logging events are similar to knowledge events and require a *documentid*, *timestamp* and an *action*. The first two are provided by the system, while the third needs to be specified by the log tag.

5.4 PRESENTATION MODEL

The presentation model is responsible for combining the data available from the domain and user model and use these to produce an individualized page for the user. The process takes into account the knowledge and the background of the user, it knows how the different notions relate to each other, and combines the different pieces of content.

As discussed in Chapter 4 the presentation model has queries and rules at its disposal. The queries retrieve the required data, while the rules use the retrieved data to adapt the contents on the page.

Local reasoning.
The application of queries and rules is done by means of *local reasoning* as opposed to an external rule engine. Local reasoning means that the queries and rules are embedded in the content, allowing the rules to take the implicit context of content into account. Implicit context occurs when content is not written in an independent manner and often involves assumptions of the author in relation to prior knowledge of the users and their goal. Such content is less suited to be fragmented and be presented in a different form and should generally speaking be prevented. However, it is very hard to write any content free of implicit context, as authors tend to work with a bigger picture of what they want to tell and then (subconsciously) make assumptions on behalf of the reader. Besides, content void of any implicit content may feel stiff and when it is combined with other content into an adapted page it might appear somewhat mosaic. So instead of denying implicit context in content, it is more practical to accept that there might be some present. Stuckenschmidt and Klein remark in their paper [196] also that local reasoning helps in making the right decisions when dealing with modular ontologies. Our setup of content divided in fragments has a great similarity with just such modular ontology. Separation of adaption rules from the content and metadata as proposed in [157] works for adaptations with a minimal context. Due to the wide variety of users and the number of required alterations, content adaptations will become finely mazed and in need for a more elaborate context, making it hard for external engines to cope with

the fine mazedness. Furthermore, an external reasoning engine limits the free form math documents we are used to in MathDox and aim to achieve in the MathDox context implementation. Therefore content needs to be aware and be able to adapt itself or its subcomponents by means of local reasoning and we opt for local implemented rules for adaption. However, separation of content and adaptation is and will remain important. For instance it is possible to organize fragments in such a manner that a fragment is without actual content but does contain rules on how to use other (small) content fragments. Ideally a fragment with adaptation rules has only control over a limited set of fragments, and will coexist with many other fragments organizing different parts of the document.

5.4.1 *Queries and rules*

Rules decide what happens in the presentation model but they require data provided by the queries to do so. The data provided can be anything from the name of the user to the fact that the user does not yet understand some required knowledge for the page he or she is reading. When the queries deliver their data to the rules, then the rules interpret the data and take actions accordingly. For instance they decide to display the name of the user on the page, or decide to include extra content so as to compensate for the knowledge of the user.

Queries and rules are responsible for the local reasoning in the presentation model. Queries connect the user and domain model to the presentation model and rules implement the presentation model.

Queries.
Data gathered with queries are made available to MathDox code by Jelly custom tags. They query the context object for data sets, and — if needed — combine and process the required data. The context object (as discussed in Section 5.1) maintains copies of the theory, symbol and variable graphs from the domain model, as well as also user model data of active users. Any data not yet available is retrieved by database queries.

Rules.
As rules require conditional logic to make decisions, they are to be implemented in either a Jelly custom tag or as MathDox code — often in the form of fragments — as those are the only parts of the MathDox format capable of conditional logic.

MathDox rules.
The most straightforward way to implement a reasoning rule is to use Jelly statements directly in the MathDox code. This approach requires the least amount of work compared to other options, but re-

sults in rules that are impossible to reuse in different settings. Rules implemented in this way are best used to take care of specific details, details that do not deserve to take the extra effort to implement the rules in a reusable way.

Listing 24: An example of a rule implemented in MathDox code

```
<mdc:knowledge-mastered name="s1p1" type="theorygraph" var="
    mastered"/>

<c:choose>
  <c:when test="${mastered}">
    The knowledge on this page has been previously understood.
  </c:when>
  <c:otherwise>
    By reading this page, you have now understood this knowledge.
  </c:otherwise>
</c:choose>
```

Example 17:
In the code example in Listing 24, MathDox code is shown that first informs about the knowledge about division[1], a node from the theory graph as depicted in Figure 23. The result of this query about the knowledge of division is then stored into the variable *mastered*. The same mastered variable is used in a Jelly *choose* statement tag, that looks at the result of the query. If the variable holds a boolean *true* value, it responds that the user has already understood the content on this page as the result of a previous visit. Otherwise the user is currently visiting the page for the first time and is informed that the system now assumes that the user has understood the knowledge. We tried not to complicate things too much in this example, but obviously there are many other possibilities, like inclusion or adaptation of content.

This example illustrates how queries can be fired from within MathDox code and how their results are then used to make decisions about content. Furthermore, it makes quite clear that these kind of rules cannot be reused elsewhere in the document and therefore these kind of rules are best suitable for subtle rules that will not be reused and do not deserve a separate implementation in fragment or Jelly custom tag.

Jelly custom tags rules.
We already encountered another way to implement rules in a more

1 Node identifiers and titles of nodal pages can differ. The MathDox context has been set up in such a way that all nodes should be referred to by their identifiers.

reusable format: Jelly custom tags. Previously these where used for queries, but the Java code that goes inside them is also perfectly suitable for more complex conditional logic and it has also far easier access to the various Java APIs compared to the MathDox format. Since Jelly custom tags are separate objects linked to tags that can be used in MathDox code, reuse of rules that are implemented in this way becomes easy.

Listing 25: A Jelly rule being called

```
<mdc:createNewUser userName="${user}" passwd="${passwd}"
                   group="${group}"/>
```

Listing 26: An example of a rule implemented by means of a Jelly custom tag

```
package org.mathdox.context.db;
import org.apache.commons.jelly.JellyTagException;
import org.apache.commons.jelly.MissingAttributeException;
import org.apache.commons.jelly.TagSupport;
import org.apache.commons.jelly.XMLOutput;

public class CreateNewUserTag extends TagSupport {
  protected String userName="";
  protected String passwd="";
  protected String group="";

  public void setUserName(String userName) {
    this.userName=userName;
  }

  public void setPasswd(String passwd) {
    this.passwd=passwd;
  }

  public void setGroup(String group) {
    this.group=group;
  }

  public void doTag(XMLOutput arg0) throws
      MissingAttributeException, JellyTagException
  {
    if ("".equals(userName)) {
      throw new MissingAttributeException("UserName not set");
    } else if {"".equals(passwd)} {
      throw new MissingAttributeException("Password not set");
    }

    XPathAccesObject.addUser(userName, passwd,
                             group, getContext());
  }
}
```

Example 18:
The MathDox example code in Figure 26 is a rule that is meant for the administrator of a MathDox context system. It is part of a MathDox page that allows administrators to add new users. To do so it needs parameters like username, password and user group, these are then passed to this tag as shown in Listing 25. The tag implementation does some verification and then passes on the request to the XPathAccesObject.

Fragment rules.
The third and last way to implement rules is by means of fragments. Fragments are designed for reuse of content, but the content itself can and often will contain rules. This approach allows a set of rules to be written in MathDox code, but opposed to the first example in a reusable manner. The opportunity to also include macros, DocBook and MONET queries sets fragment rules apart from Jelly custom tag rules. Rules implemented by fragments tend to apply for larger parts of content. Where Jelly custom tag rules typically are used for smaller items, while rules implemented by fragments are ideally suited for styling of content or even a complete page.

Listing 27: An example of a fragment implementing a rule

```
<mdc:fragment xmlns:mdc="jelly:org.mathdox.context.
    TheoryBlocksTagLibrary"
              xmlns:x="jelly:xml"
              xmlns:xforms="http://www.w3.org/2002/xforms"
              xmlns:c="jelly:core">

  <c:new var="_result" className="java.util.HashMap"/>
  <c:set var="iterator" value="${from}"/>

  <c:while test="${iterator le till}">
    <x:parse var="result">
      <monet:query xmlns:monet='http://monet.nag.co.uk/monet/ns'>
        <monet:classification>
          <monet:directive-type href="http://mathdox.org/
              phrasebook/mathematica#eval"/>
        </monet:classification>
        <monet:body>
          <monet:output>
            <OMOBJ>
              <OMA>
                <OMS cd="set1" name="in"/>
                <OMI>${iterator}</OMI>
                <OMS cd="setname1" name="P"/>
              </OMA>
            </OMOBJ>
          </monet:output>
```

```
25         </monet:body>
26       </monet:query>
27     </x:parse>
28 
29     <c:set var="result" trim="true">
30       <x:expr select="$result//OMS/@name" />
31     </c:set>
32 
33     <c:if test="${ result or not justPrimes }">
34       <c:choose>
35         <c:when test="${result}">
36           <c:set var="primetest" value="Prime "/>
37         </c:when>
38         <c:otherwise>
39           <c:if test="${not justPrimes}">
40             <c:set var="primetest" value="No"/>
41           </c:if>
42         </c:otherwise>
43       </c:choose>
44 
45       <c:new className="java.lang.Integer" var="intobj">
46         <c:arg type="int" value="${iterator}"/>
47       </c:new>
48       <c:invoke on="${_result}" method="put">
49         <c:arg value="${intobj}"/>
50         <c:arg value="${primetest}"/>
51       </c:invoke>
52     </c:if>
53 
54     <c:set var="iterator" value="${iterator+1}"/>
55   </c:while>
56 </mdc:fragment>
```

Example 19:

MathDox code in Figure 27 shows a fragment that implements a rule that determines which integers from an interval are prime and which are not. The fragment is called from MathDox code — a MathDox page, or another fragment — and uses three variables. Two of these, a start and end point, define an integer interval. Each integer from this interval is tested for being prime by asking Mathematica Phrasebook [2]. A third variable indicates if the returning list should only contain the primes or should contain all integers. It is then up to the calling code to process the list of integers.

2 note the mathematica#eval in line 11 to indicate a different than default cas

5.4.2 *Pre-defined set queries and rules*

Our implementation of the mathematical context model, comes with a pre-defined set of queries and rules. This set of queries and rules does not claim to be complete, nor does it try to be. Authors are invited to add new as they see fit. The pre-defined set is divided into a few subsets.

5.4.2.1 *Queries*

As stated before in Section 5.4.1, queries are responsible for retrieving information from the user model. The returned data is then used by rules for presentation purposes. We will discuss some queries as implemented in the mathematical context model.

Mastered, available and not reachable nodes.
There are three queries that retrieve information from the knowledge context in relation to the theory and symbol graphs. These queries return node identifiers that can be used by other queries or rules for additional information or alternative content. These rules are:

- *masteredNodes*
  Retrieves all theory or symbol identifiers that have been marked as understood (mastered) or whose total amount of knowledge points surpasses the set threshold for this node in the theory or symbol graph.

- *availableNodes*
  Those nodes that are currently ready to be read without the need of reading prior knowledge first. As a user continues with the document, this set will change.

- *notReachableNodes*
  Returns the nodes from the theory or symbol graph that require knowledge that is listed as (a) predecessor(s) in either the theory or symbol graph. The user is expected to understand the associated knowledge before he or she is expected to read and understand any nodes in the returned set.

These queries apply to both the theory and symbol graph attributes and require some additional parameters; these are:

- *graphType*
  Selects either the theory graph or the symbol graph to work with.

- *returnType*
  Indicates if the results should be returned in either a Java collection, or as a String. Some rules work better if the results are offered as a collection object.

- *var*
  If the result is to be stored into a Jelly variable for later use, *var* will hold the name of that variable.

Listing 28: The use of mastered available and notReachable queries.

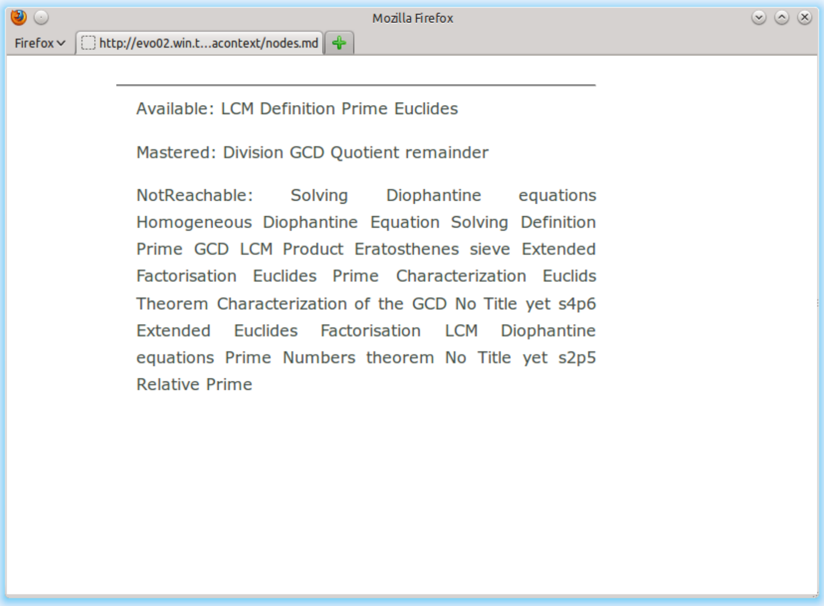
```
<!-- retrieval of node lists -->
<mdc:availableNodes returnType="collection"
                    graphType="theorygraph" var="available"/>

<mdc:masteredNodes returnType="collection"
                   graphType="theorygraph" var="mastered"/>

<mdc:notReachableNodes returnType="collection"
                       graphType="theorygraph" var="outOfReach"/>

<!-- output of the node lists -->
<c:forEach var="nodalpage" items="${available}">
  <mdc:graph-property nodeName="${nodalpage}" graphType="
      theorygraph" property="title"/>
</c:forEach>

<c:forEach var="nodalpage" items="${mastered}">
  <mdc:graph-property nodeName="${nodalpage}" graphType="
      theorygraph" property="title"/>
</c:forEach>

<c:forEach var="nodalpage" items="${outOfReach}">
  <mdc:graph-property nodeName="${nodalpage}" graphType="
      theorygraph" property="title"/>
</c:forEach>
```

See Listing 28 for an example how to use these queries and see Figure 29 where the output is shown. After the first three lines have been executed the Jelly variables *available*, *read* and *outOfReach* have been created, each containing a subset of the nodes from the theory graph. Note that these three sets do not have any elements in common, and together hold all elements. In Listing 28 also the *graph-property* query is being used to look up the title property for each node.

Retrieving variables from context.
Variables stored in the context — mathematical and non-mathematical — contain very important information for adaptation rules and are accessible by means of queries from MathDox code. The `<get-value>` tag takes five attributes of which three are commonly used.

- *symboltableid*
  To access a variable the *symboltableid* has to be specified.

- *variable*
  The variable requested from the mathematical or logistic context.



```
Available: LCM Definition Prime Euclides

Mastered: Division GCD Quotient remainder

NotReachable: Solving Diophantine equations
Homogeneous Diophantine Equation Solving Definition
Prime GCD LCM Product Eratosthenes sieve Extended
Factorisation Euclides Prime Characterization Euclids
Theorem Characterization of the GCD No Title yet s4p6
Extended Euclides Factorisation LCM Diophantine
equations Prime Numbers theorem No Title yet s2p5
Relative Prime
```

Figure 29: Output of code in Listing 28

- *var*
  Sometimes it is required or convenient to store a variable from context in a Jelly variable within the MathDox code itself. If a variable name is given in the *var* attribute then that variable is used. Otherwise the contents of the variable from the context is directly used as output.

- *date*
  The date used to revert the content of pages to a certain date in the past. By default the value 'now' is used which is automatically translated to the current date time stamp.

- *evaluate*
  Only mathematical variables from the variable graph —which are not input variables— can use this attribute. It forces the expression of the variable graph to evaluate. Again in normal use, this is not required as such a variable will check itself to see if any of its input values has been changed before returning a value.

An example is shown in Listing 29. It retrieves the variable *goal* —used to specify a learning goal— from the logistic context and then stores the value in a Jelly variable named *goal*, which is then used as output. It makes no difference whether the requested variable is text or XML (OpenMath) based, as demonstrated by line 6 where an OpenMath value is retrieved. Also note that the result of this second call will be directly used as output as it is not stored in a variable first.

Listing 29: A retrieval of the variable from the logistical context

```
<para>
  <mdc:get-value symboltableid="root"
       variable="goal" var="goal"/>
  ${goal}
</para>
<para>
  <mdc:get-value symboltableid="s1p1" variable="a"/>
</para>
```

5.4.2.2 *Content rules*

The mixture of queries and rules in the MathDox content leads to content that is in control of its own adaptation as opposed to an external rule engine. Probably the most important tag is the *include-fragment* tag as this rule takes care of inclusion of content on a page. However other rules also exist. We will now illustrate two examples that have been implemented, the *include-fragment* and *definition* rules.

Include fragment.
The *include-fragment* tag will include another piece of MathDox code into the current page. The contents being MathDox code, it is of paramount importance to be able to share some of the interpreting context (Jelly context) with the fragment to be included. That said, it is also important to avoid collisions in the same Jelly context to prevent duplicate names of variables or XForms identifiers. For this reason the *include-fragment* takes the element body in which jelly variables can be declared; these are then created in a new Jelly context connected to the calling Jelly context, allowing access to previous and to the new created variables. These new Jelly variables are only valid within the fragment and are discarded when the fragment finishes as the new created Jelly context is no longer available. See Figure 30 for an example. Attributes of the *include-fragment* are:

- *fragment*
  The location of the fragment to include.
- *frgid*
  XForms elements are prefixed with the *frgid* to keep the different XForms elements in different fragments separate from each other regardless of their names. For this purpose each fragment in the same code file needs to have a unique identifier. These identifiers will be prefixed to XForms identifiers. Fragments that include fragments create a chain of prefixed identifiers.

- *args*
  Much like the *static main(String[] args)* method, a fragment can also accept a Java map object with name value pairs to be used as variables inside the fragment. This works exactly the same as if the variables are declared in the body of the *include-fragment* tag.

- *var*
  Sharing a Jelly context also needs means to return variables and their values from inside a fragment back to the calling code. The *var* attribute will hold a name of a variable that is used to store a Java Map with name value pairs containing results of the fragment code. These results only need to be returned if the calling code needs them, as the execution of the fragment itself is directed to the output and thus to the page.

Listing 30: A fragment call a step in the Euclidean algorithm.

```
<mdc:include-fragment
  fragment="fragments/rules/euclides/euclidesstep.mdf"
  frgid="step" var="result">
  <c:set var="a" value="${a}"/>
  <c:set var="b" value="${b}"/>
  <c:set var="step" value="${step+1}"/>
</mdc:include-fragment>
```

Definition Stretchtext.
Imagine a student reading a page in which he or she is reminded of the definition of the Euclidean algorithm. And the frustration when it just does not want to spring in mind what it was. A solution can be offered by offering the student a *stretch text* rule by clicking, for instance the name of the definition, the definition appears on the screen right in between the rest of the page contents, offering the student a chance to catch up without leaving the page. Obviously, the same is possible when the student is offered a statement with the proof hidden only to appear upon demand. Depending on personal preferences, these pieces of content can be set to open by default, inserting it into the page, or to open in a new window or tab.

The *definition* tag actually consists of two tags: the said *definition* tag and the *definition-anchor* tag. The first displays the definition name, while the second indicates where the stretching of the text will happen.

In Listing 31 an example is given; the attributes for *definition* are:

- *Name*
  The name of the node from either the theory graph or the symbol graph. It is used to locate a fragment that summarizes the knowledge that goes with the node.

- *GraphType*
  Indicates in which graph the node as specified at the *name* attribute is to be found.

- *Location*
  A different location can be given in the *location* attribute for the content that is to be included in the stretch text. This approach allows also content that is not bound to the theory or symbol graph.

- *title*
  The title of the definition name as it will appear in the text on the page.

- *on*
  Defaults to *true*, but if the title needs to remain unclickable text — turning effectively the stretch text off — a *false* value will do that.

- *stretch*
  Default setting is *true*, but if set to *false*, the content will appear in a new window instead.

- *var*
  The definition tag will create an XML document that tells the *definition-anchor* what to do. This XML document is stored in the variable with the name of *var*.

Listing 31: An example of a stretch text.

```
<para>
  There is also an extended version of
  <mdc:definition on="true" name="s2p1" graphType="theorygraph"
  title="Euclidean Algorithm" var="EuclidsAlgorithms2p1"/>,
  which determines,
  <!-- more text -->
</para>

<!-- a suitable location for the stretch text to appear -->
<mdc:definition-anchor var="EuclidsAlgorithms2p1" />
```

The *definition-anchor* only takes a *var* attribute containing the constructed XML document from the *definition* tag.

5.4.2.3 *Presentation rules*

In contrast with rules concerning the inclusion of content, presentation rules do not add content, they change content. Examples of these are styling rules, rewriting mathematical notation rules, but also template rules that add styling or a uniform environment to existing content.

Styling multiple choice exercises.
A styling rule that is implemented as a fragment is the *mcfragment*, a fragment that implements a multiple choice question. It takes an XML document which contains the question, the answers and also feedback on each chosen answer. It is then up to the fragment how to render the exercise on the page. The style can either be selected by the reader based on personal preferences in the logistic context or set by the author. There are two formats available, an elaborate presentation and a smaller version. In Listing 32 and example is given how to include a multiple choice exercise from MathDox code. In the listed code, a Java HashMap object is created and named *parameters*. The map is then filled with the XML documents for the question and for the answers of the multiple choice exercise. Next the map is given to the fragment responsible for creating the multiple choice exercise by putting it in the *args* attribute.

Listing 32: Calling the styling rule for a multiple choice exercise.

```
<c:new var="parameters" className="java.util.HashMap"/>
<c:set target="${parameters}" property="question"
       value="${question}"/>
<c:set target="${parameters}" property="answer"
       value="${answer}"/>
<mdc:include-fragment fragment="exercises/mcfragment.mdf"
                      frgid="exercise" args="${parameters}"/>
```

5.4.2.4 *Observer rules*

Observer rules are different from the other types of rules because they do not add or change any content. Instead they update the context of the user and adjust the data to match with the never changing data of the user. Some important rules are the *set-value* tag and the *add-knowledge* tag. The *set-value* tag stores variable values into the logistic or the mathematical context. The *add-knowledge* tag writes knowledge events to the knowledge context. Here the data is available for queries and other rules can draw their conclusions from it.

- *Name*
  The name of the node the current event belongs to.
- *GraphType*
  The graph (theory or symbol) that contains the node in *name*.
- *Action*
  The type of event. *mastered* and *read* are used to determine if the knowledge linked to the node is understood. However any other event type may be used.
- *Points*
  Keeping track of progress involves handing out points, positive

of negative. The sum of the awarded (or subtracted) points is an indication on how well a user understands a certain notion.

belonging to *s1p1* (division) marks itself as read by adding 1 point to the amount of times the page is read. The threshold to consider the page understood is 1, meaning that by reading this page once, the node *s1p1* will be marked as understood. The bug model [139] as discussed in Section 3.1.5 can be implemented with the use of this rule. Add points for correct answers to exercises and subtract points for wrong answers or other signs of misunderstanding.

Listing 33: Adding knowledge events to the knowledge context.

```
<mdc:add-knowledge name="s1p1" type="theorygraph"
                   action="read" points="1"/>
```

5.4.2.5 *Navigation rules*

Deviating from the traditional linear reading through the content (as can be found in a printed book), allows users to pick their own path and some of them will subsequently get lost as a result. This is not surprising because if the student knew enough about the content to find his own path he or she is likely to have already some understanding of the content. Some navigation rules are created to help the reader of the document and guide him or her into the right direction. Examples are an index of used symbols on the page containing links to their descriptions and the visualization of the theory and symbol graph which act like maps of the content in the document. To use a map however one first has to have a destination — a goal — in mind before a route can be created and followed. Determining the goal as described in the plan model [139] of a user is not trivial; see Section 3.1.6. Instead a simpler approach in which the user tells the system what his or her goal is, is implemented. A way to retrieve this goal was already shown in Listing 29. The next step towards a set goal, is always a node that has no unmet requirements to prior knowledge and is therefore marked as available for reading. Leading to $A \subseteq G$, where G is the set of nodes that should be taken to reach the goal and A is the set of nodes that are available for reading. In Listing 34 the set of G is retrieved by using the *availableNodes* tag discussed earlier in Listing 28 with the help of the *goal* variable discussed earlier that was used before in Listing 29. In this example the only new attribute is the *goal* attribute which contains the node name of the goal.

Listing 34: Retrieving the set of nodes leading towards the goal node.

```
<mdc:available-theories returnType="collection"
graphType="theorygraph" var="goalset" goal="${goal}"/>
```

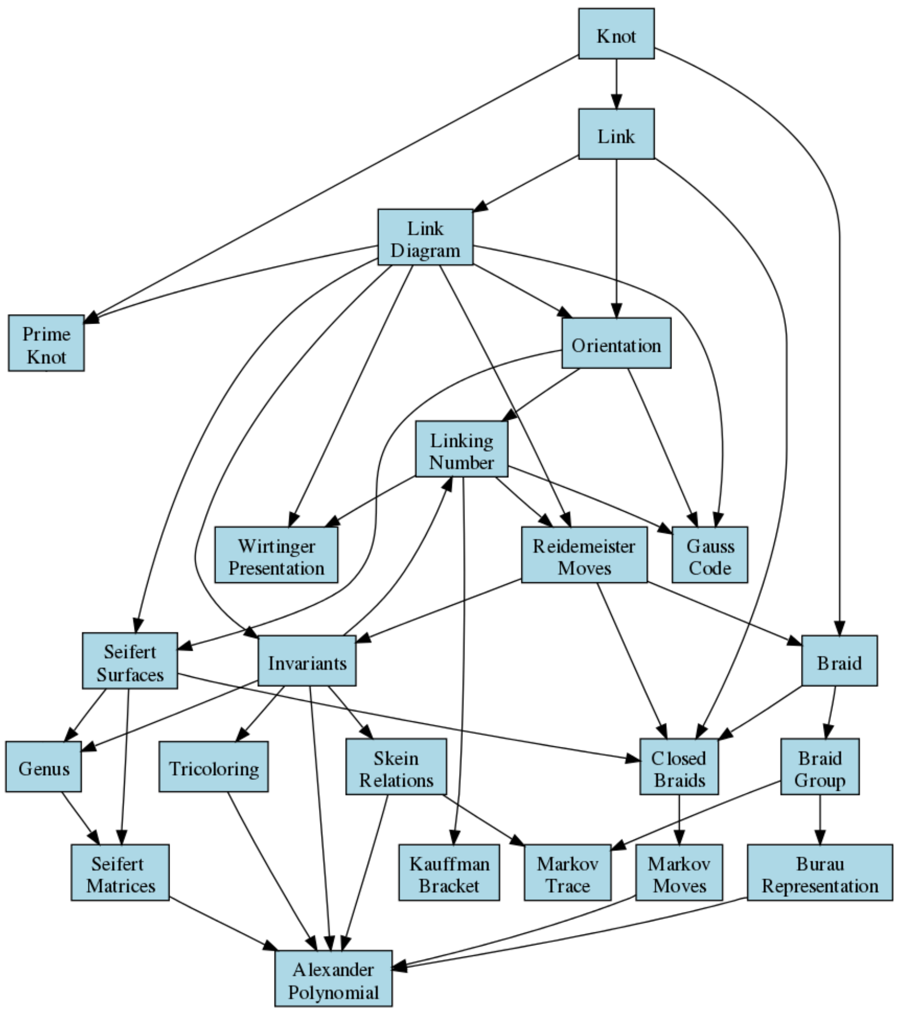


Figure 30: The theory graph as used by the knots document

5.4.3 *The knots example*

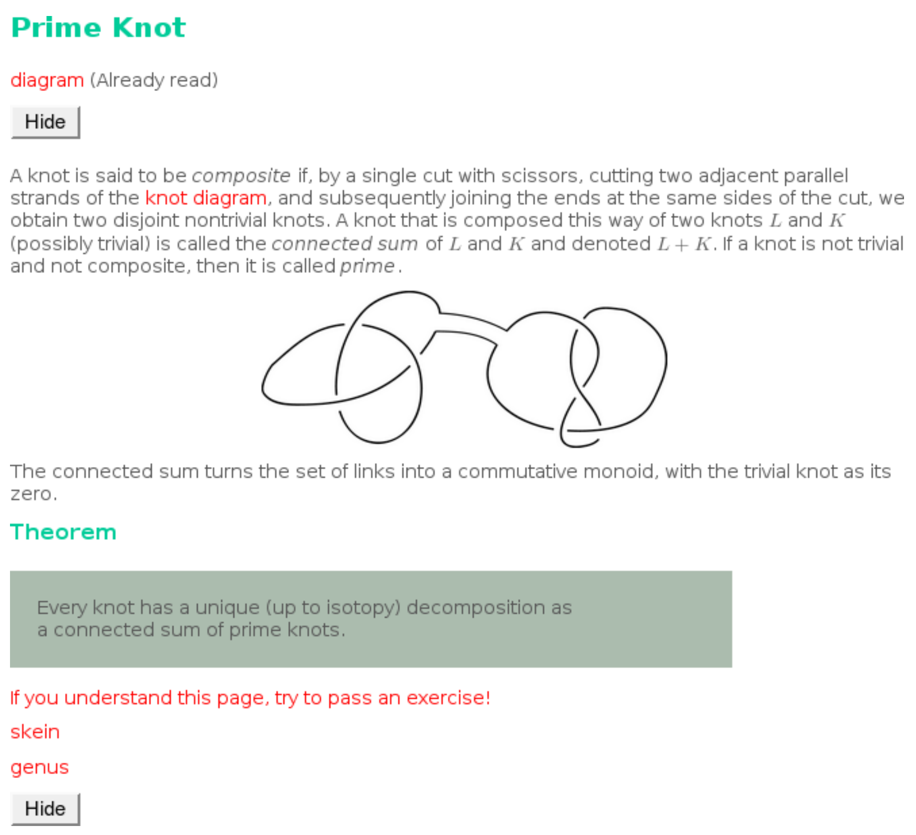
An existing MathDox document on knot theory was transformed into an adaptive mathematical document; see [50]. The main addition to the static knots document was a theory graph — see Figure 30 — that allows users to select their own route through the document. It lacks a symbol and a variable graph and limits its adaptivity to a binary overlay model on the theory graph. It demonstrates the flexibility of the MathDox Context system, as it shows that authors can limit what they use to those parts they are interested in. The example about IDA [148] — an adaption of Interactive Discrete Algebra — (see Section 5.4.4) uses more of the MathDox Context.

theory graph.
Each node of the theory graph in Figure 30, is attached to nodal page. Code of a nodal page typically looks as shown in Listing 35. In Listing 35 a fragment rule *template.mdf* is used to implement the same

basic behavior on all nodal pages. It determines its identity by using the values of the parameters *theoryGraphURL* and *theoryName*, which are pointers to the theory graph and the theory node in this graph [3]. The variable *fragmentAddress* is used to find the fragment with the actual contents that are to be displayed on the page. An exercise *passExercise* is also specified and is used to judge whether the user has understood the content.

Listing 35: An example of a template as constructed with fragments

```
<article>
  <mdc:include-fragment fragment='template.mdf' var='result' >
    <!-- all set variables in this body are parameters for the
        fragment -->
    <c:set var='theoryName'>primeknot</c:set>
    <c:set var='theoryGraphURL'>knottheorygraph.xml</c:set>
    <c:set var='fragmentAddress'>primeknot.mdf</c:set>
    <c:set var='passExercise'>primeknotpassexercise.md</c:set>
  </mdc:include-fragment>
</article>
```



Prime Knot

diagram (Already read)

Hide

A knot is said to be *composite* if, by a single cut with scissors, cutting two adjacent parallel strands of the knot diagram, and subsequently joining the ends at the same sides of the cut, we obtain two disjoint nontrivial knots. A knot that is composed this way of two knots L and K (possibly trivial) is called the *connected sum* of L and K and denoted $L + K$. If a knot is not trivial and not composite, then it is called *prime*.

The connected sum turns the set of links into a commutative monoid, with the trivial knot as its zero.

Theorem

Every knot has a unique (up to isotopy) decomposition as a connected sum of prime knots.

If you understand this page, try to pass an exercise!

skein

genus

Hide

Figure 31: A view of a nodal page of the knot example

3 The implementation of the knots example was done against an older version of the MathDox context. In this version the document file, used to store pointers to the graphs, was not available. Also, some rules have changed in more recent versions.

Listing 36 explains how the page shown to the user is constructed by the template fragment. Some parts of the code listed have been omitted in order to improve readability. Figure 31 shows the resulting page.

The *template* fragment begins at step one, with the title of the page to be rendered, which is derived from the theory node in the theory graph. Next, it assigns a boolean value to the variable 'CanGoOn' which indicates whether the user has the required prerequisite knowledge. According to the value of this boolean, it either serves the contents (in 'fragmentAddress') and an exercise at step two and three, or suggests the user to a theory closer to the root in the theory graph as he or she did not yet understand the prior knowledge.

Listing 36: Contents of the template fragment rule

```
<title>
  <!-- 1 Title of this topic-->
  <mdc:getTheoryProperty theoryGraphURL='${theoryGraphURL}'
        theoryName='${theoryName}' propertyName='verbose'/>
</title>

<!-- if all required knowledge is present CanGoOn is set to true -->
<mdc:available-theories theoryGraphURL='${theoryGraphURL}'
          theoryName='${theoryName}' direction='backward'
          var='CanGoOn'/>
  <c:choose>
    <c:when test='${CanGoOn}'> <!-- preknowledge is present -->
      <!-- 2 show previous topics leading up to the current topic -->
      <mdc:include-fragment fragment='successors.mdf'>
        <c:set var='buttonName'>Predecessors</c:set>
        <c:set var='direction'>Backward</c:set>
      </mdc:include-fragment>

      <!-- 3   show the content identified by the fragment address -->
      <mdc:include-fragment fragment='${fragmentAddress}'/>
      <para> <!-- link to exercise -->
        <ulink url='${passExercise}'>Try to pass an exercise!</ulink>
      </para>

      <!-- 4   shows follow up topics available after current topic -->
      <mdc:include-fragment fragment='successors.mdf'>
        <c:set var='buttonName'>Successors</c:set>
        <c:set var='direction'>Forward</c:set>
      </mdc:include-fragment>
    </c:when>
    <c:otherwise>
      <!-- preknowledge not present, redirect -->
      <para>Sorry, You first need to study the following sections
          .
        <mdc:include-fragment fragment='successors.mdf'>
          <c:set var='buttonName'>Predecessors</c:set>
          <c:set var='direction'>Backward</c:set>
          <c:set var='list'>true</c:set>
        </mdc:include-fragment>
      </para>
    </c:otherwise>
  </c:choose>
```

With the exercise being offered the user can try to establish whether he or she understood the theory. After passing the exercise the user is suggested another page to read, based upon the knowledge context at step four.

5.4.4 *The IDA example*

Interactive Discrete Algebra (IDA) [2] is a book and an interactive mathematical document written in MathBook, an early predecessor of MathDox. It is aimed at teaching freshmen the beginnings of (discrete) algebra. The IDA document is organized as if it were a book; there was no adaptation and the possibilities of the interactiveness were limited.

This made IDA well suited to be adapted to become an adaptive interactive document and show the possibilities of an adaptive mathematical document to the world. We will discuss the properties of the adaptive IDA document: the theory graph, symbol graph and variable graph, the knowledge-, logistic- and mathematical contexts, followed by examples of adaptive techniques implemented in IDA.

Listing 37: A node declaration in a theory graph

```
<node id="s1p1">
  <data key="title">Division</data>
  <data key="threshold">1</data>
  <data key="oms">
    <symbols>
      <OMS cd="integer1" name="quotient"/>
      <OMS cd="integer1" name="factorof"/>
    </symbols>
  </data>
</node>
```

5.4.4.1 *The Domain model*

The *domain model* in the IDA document consists of the theory graph, symbol graph and variable graph but also contains all content files.

Theory graph.
The nodes of the theory graph contain extra information for the pages that belong to that theory. For instance a human readable title instead of an identifier used by the system and which OpenMath symbols are introduced. Other data that can be included is the *threshold*, after how many reads a page is considered understood and *links* which contains links that are of interest to users with a specific background.

In Listing 37 the first node of the theory graph has been defined. This node has been named *s1p1*, after section 1 page 1 of the original IDA. Here, s1p1 is used as identifier of the node in the document. A human readable name has been defined in the *title* property. The page belonging to this node will be marked as understood after reading it once because the *threshold* has been set to 1. The *oms* property specifies the symbols that are marked to be introduced on this page.

The complete theory graph as used by IDA is shown in Figure 32. Note, there has been an overlay with user data added to this graph. Here the nodes that have been marked as understood are colored green. The blue nodes are the nodes that do not require prior knowledge but are not yet understood, while red nodes require knowledge that is still marked as not understood. A goal was set for the user to whom this theory graph belongs, namely the node with the red edge. The blue nodes with the blue edges do not require any knowledge and are on the route towards the set goal. This makes them the best suited nodes to read next.

More information about theory graphs in general can be found in Sections 4.1.1 and 5.2.1.

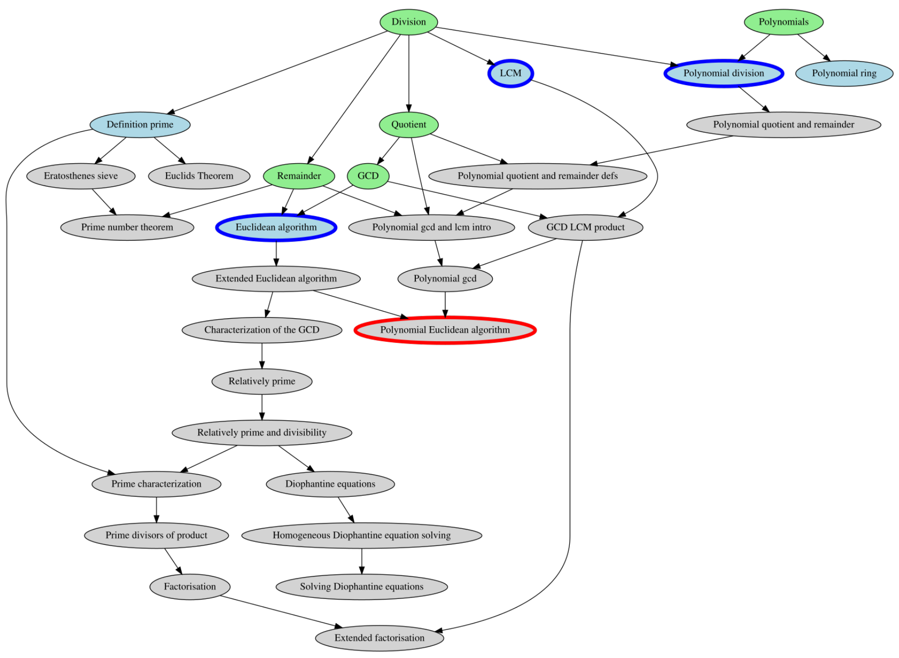

Division
Polynomials
LCM
Polynomial division
Polynomial ring
Definition prime
Quotient
Polynomial quotient and remainder
Eratosthenes sieve
Euclids Theorem
Remainder
GCD
Polynomial quotient and remainder defs
Prime number theorem
Euclidean algorithm
Polynomial gcd and lcm intro
GCD LCM product
Extended Euclidean algorithm
Polynomial gcd
Characterization of the GCD
Polynomial Euclidean algorithm
Relatively prime
Relatively prime and divisibility
Prime characterization
Diophantine equations
Prime divisors of product
Homogeneous Diophantine equation solving
Factorisation
Solving Diophantine equations
Extended factorisation

Figure 32: The theory graph as used in IDA

Symbol graph.
The symbol graph is very similar to the theory graph, again the graph specifies dependencies between nodes and indicates which symbols are marked as being understood. In our implementation of adaptive IDA we specified three extra properties for each node: title, representation and page. In Listing 38 an example of a few nodes as defined in the IDA symbol graph are given. Figure 33 shows a part of the same symbol graph. The title property adds a human readable title to a node, which is used on a page dedicated to the symbol, but also when the symbol needs to be listed and does not possess a suitable rendering. The representation property tells the document how to render the symbol. The *arith1.gcd* node lacking a decent rendering therefore chooses to use the title as rendering, while the *setname1.C* node does have a good rendering and chooses to have this rendering used. Finally, the page property tells IDA where to find a page that explains the symbol into more depth. If this value is set to document, the page is expected to be present in the document. By default the parameter is set to OM, indicating that it should use the OpenMath [77] website to find information about the symbol. More information about symbol graphs can be found in Section 4.1.2 and 5.2.2.

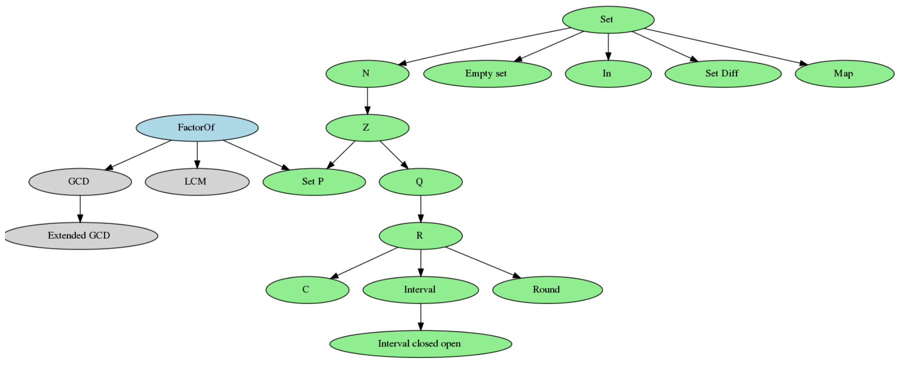


Figure 33: The symbol graph as used in IDA

Listing 38: A node declaration in a symbol graph

```
<node id="arith1 .gcd">
  <data key=" title ">GCD</data>
  <data key="representation">title</data>
  <data key="page">document</data>
</node>

<node id="setname1 .C">
  <data key=" title ">C</data>
  <data key="representation">OM</data>
</node>
```

Variable graph.
The variable graph, as shown in Figure 34, is distilled from the vari-

ables used in the adaptive IDA document. It specifies where variables that are computed should look for input. In Listing 39 two nodes from the variable graph are given. They define the variables *s1p1.a* and *s1p1.ab_q*, or in short *a* and *ab_q*. Name clashes are prevented by prefixing the introducing theory nodes with the identifier. This is not mandatory, it is just for convenience, using a different prefix does not lead to errors. Moreover, the prefix is not used for soundness checks.

Some variables consist of expressions that need to be computed, the functional variables. A property named *cas* allows for the selection of the computer algebra system. Different computer algebra systems have different strengths making exceptions of the default cas necessary.

An *inputNode* does not have an expression to calculate its value, instead it is given by the author or user. To tell the system if a variable is an inputNode, a property can be set. Especially input nodes require conditions to prevent values that cause problems in examples either directly or indirectly by computed variables that use this value as input. Conditions are not limited to input variables, but can be applied to all variables from the variable graph. They come in two types, a CAS-condition and an XPath-condition. The first is an OpenMath expression that needs to return true from a computer algebra system. The second condition type uses XPath to verify that the OpenMath expression meets certain standards, for example restriction of the use of certain OpenMath symbols. Using the *mdc:variable* tag in a condition allows to use the value of a variable. Where conditions are involved it usually is a self reference and the name attribute is omitted. The *mdc:variable* is also used in OpenMath expressions for non input variables. Here it is used to take the values of the named variables and to use these as input. Consequently the variable will use the expression to compute its own value. An expression used to compute a value for a non input node can be found in the variable *s1p1.ab_q* as shown in Listing 39; also an example of both type of conditions can be found there.

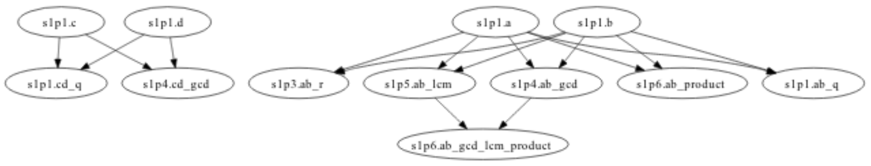


Figure 34: The variable graph as used in IDA

Listing 39: A node from the variable graph

```
<node id="s1p1.a">
  <data key="cas">gap</data>
  <data key="inputNode">true</data>
  <data key="conditions">
```

```
    <conditions>
      <condition type="xpath">
        <xpath>not(//OMS[@cd="arith"])</xpath>
        <message>no symbols from arith1 dictionary are allowed</
            message>
      </condition>
      <condition type="cas" cas="gap">
        <omexpr>
          <OMA>
            <OMS cd='logic1' name='and' />
            <OMA>
              <OMS cd='relation1' name='leq'/>
              <mdcv:variable/>
              <OMI>100</OMI>
            </OMA>
            <OMA>
              <OMS cd='relation1' name='geq'/>
              <mdcv:variable/>
              <OMI>25</OMI>
            </OMA>
          </OMA>
        </omexpr>
        <message>N must be inbetween 2 and 10!</message>
      </condition>
    </conditions>
  </data>
</node>

<node id="s1p1.ab_q">
  <data key="cas">gap</data>
  <data key="OMExpression">
    <OMA>
      <OMS cd="integer1" name="quotient"/>
      <mdcv:variable name='s1p1.a'></mdcv:variable>
      <mdcv:variable name='s1p1.b'></mdcv:variable>
    </OMA>
  </data>
</node>
```

Content.

The remaining component of the domain model is the content itself. Content includes the *document.xml*, the *contextvariables.xml* and all files containing content. The document.xml given in Listing 40 contains important settings for the adaptive mathematical document. It specifies where it can find the theory graph, symbol graph, and variable graph. As an adaptive mathematical document most likely is not the only document —adaptive or non adaptive— running in the MathDox Player installation, it needs to be told where the *documenthome* folder is. It considers everything in this folder as part of the document. The *prior-knowledge* tag indicates which symbols, that occur in the document, are considered prior knowledge. Meaning these will not be introduced or explained. Because it can be tedious to list each and every symbol in an OpenMath content dictionary, it is also possible to list complete content dictionaries instead with the *required-CDs* tag. Per default a symbol that is marked as prior knowledge has no page within the document. However, when OpenMath symbols that are considered prior knowledge are in need of a page explaining them, then they need to be listed under *extraDefinedSymbols*. A symbol listed under *extraDefinedSymbols* tells the MathDox Context system to look locally for an explanation page, instead of referring to an OpenMath page.

Listing 40: A document file.

```xml
<document docid="idacontext-cp1">
  <documenthome>
    http://dam04.win.tue.nl/rikkomathdoxplayer/idacontext
  </documenthome>
  <theorygraph>
    http://dam04.win.tue.nl/rikkomathdoxplayer/idacontext/graphs/
        theorygraph/theorygraph.xml
  </theorygraph>
  <symbolgraph>
    http://dam04.win.tue.nl/rikkomathdoxplayer/idacontext/graphs/
        symbolgraph/symbolgraph.xml
  </symbolgraph>
  <variablegraph>
    http://dam04.win.tue.nl/rikkomathdoxplayer/idacontext/graphs/
        variablegraph/variablegraph.xml
  </variablegraph>

  <prior-knowledge>
    <required-symbols>
      <OMS cd="arith1" name="abs"/>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="sequence1" name="sequence"/>
      <!-- not all symbols are shown -->
      <OMS cd="setname1" name="Q"/>
      <OMS cd="setname1" name="R"/>
      <OMS cd="setname1" name="Z"/>
      <OMS cd="transc1" name="ln"/>
    </required-symbols>
    <required-CDs>
      <required-CD cd="prog1"/>
    </required-CDs>
  </prior-knowledge>

  <extraDefinedSymbols>
    <OMS cd="setname1" name="Z"/>
  </extraDefinedSymbols>
</document>
```

The *contextvariables.xml* file lists all initial settings of the variables from the variable graph and the logistical context. These settings are used whenever a new user is created and makes sure that each new user starts in a well-defined environment.

The content itself is divided into a set subdirectories directly under the documenthome setting. This set lists the folders *definitions*, *examples*, *exercises*, *fragments*, *images*, *nodal pages* and *symbols*. Each subdirectory —except for the symbols directory— contains a set of subdirectories equal to the set of nodes in the theory graph. This allows authors to keep content organized. The symbols directory contains a subdirectory for each symbol in the symbol graph and all symbols listed under *extraDefinedSymbols*. This way of organizing is not required by the soundness verification process.

5.4.4.2 *User model*

The user model consists of three different sub-contexts, the *knowledge*, *logistic* and *mathematical context*.

Listing 41: Knowledge events stored in the database

```
<document docid="idacontext-cp1">
  <action actionid="read">
    <theory name="c3s2p5" date="20150327105555" resource="/
        experimental/idacontext/nodalpages/c3s2p5/index.md"
        graphURL="http://evo02.win.tue.nl/rikkomathdoxplayer/
        experimental/idacontext/graphs/theorygraph/theorygraph.
        xml"/>
    <theory name="c3s2p5" date="20150327105800" resource="/
        experimental/idacontext/nodalpages/c3s2p5/index.md"
        graphURL="http://evo02.win.tue.nl/rikkomathdoxplayer/
        experimental/idacontext/graphs/theorygraph/theorygraph.
        xml"/>
  </action>
  <action actionid="mastered">
    <theory name="c3s1p1" date="20150327133251" resource="/
        experimental/idacontext/nodalpages/c3s1p1/index.md"
        graphURL="http://evo02.win.tue.nl/rikkomathdoxplayer/
        experimental/idacontext/graphs/theorygraph/theorygraph.
        xml"/>
    <theory name="s1p2" date="20150327145248" resource="/
        experimental/idacontext/nodalpages/s1p2/index.md"
        graphURL="http://evo02.win.tue.nl/rikkomathdoxplayer/
        experimental/idacontext/graphs/theorygraph/theorygraph.
        xml"/>
  </action>
</document>
```

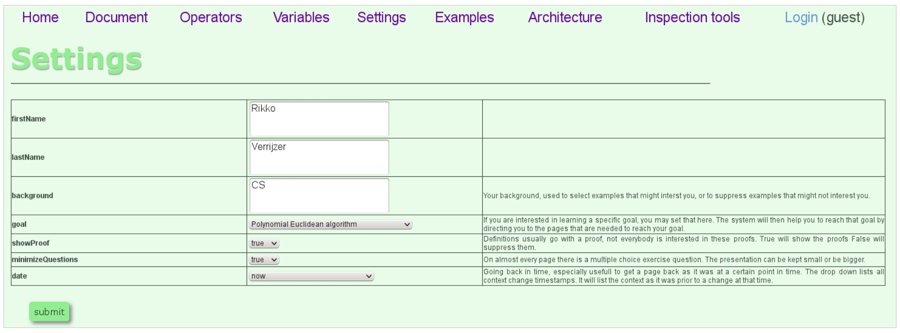
Knowledge context.
The knowledge context is responsible for keeping track of knowledge

events. In our adaptive IDA document we allow sub-categories; most notable are the *knowledge theory context* and *knowledge symbols context*.

But even within these subcategories there is a need to group specific events. Therefore the knowledge context — both theory and symbol — is able to store events labeled with an *action*. In order to indicate when a notion belonging to the theory or symbol graph is read or mastered, the *read* and *mastered* are the most commonly used actions. Other actions are also possible. Authors do not need to define actions before usage. In Listing 41 an example of knowledge context is given as stored in the XML database eXist [22]. Note that it only lists entries for the knowledge theory context and that some lines have been omitted for readability purposes. The knowledge symbol context is stored in a similar fashion in the same XML document.

The *knowledge (symbols) context* is implemented in the same way as the *knowledge (theories) context*.

Logistic context.
The logistic context contains all non-mathematical variables that are important for the adaptive IDA document, see Figure 35. These variables range from the name of the user, to the *date*. Some variables have dropboxes or radioboxes in oder to prevent them from taking an illegal value. It is up to the author to guard the validity of the input into the logistic context.

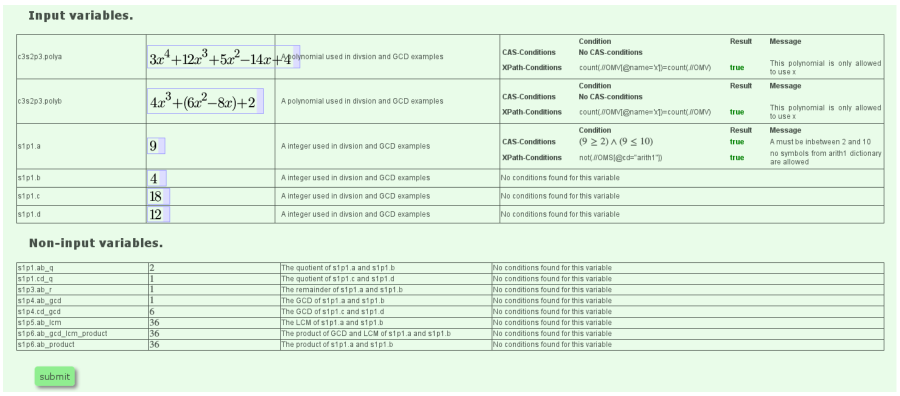


Settings

firstName	Rikko	
lastName	Verrijzer	
background	CS	Your background, used to select examples that might interst you, or to suppress examples that might not interest you.
goal	Polynomial Euclidean algorithm	If you are interested in learning a specific goal, you may set that here. The system will then help you to reach that goal by directing you to the pages that are needed to reach your goal.
showProof	true	Definitions usually go with a proof, not everybody is interested in these proofs. True will show the proofs False will suppress them.
minimizeQuestions	true	On almost every page there is a multiple choice exercise question. The presentation can be kept small or be bigger.
date	now	Going back in time, especially usefull to get a page back as it was at a certain point in time. The drop down lists all context change timestamps. It will list the context as it was prior to a change at that time.

Figure 35: Logistic context of a user

Mathematical context.
In Figure 36 the mathematical context is shown. The input variables of the variable graph —as discussed before in the domain model— can be set by the user and will be stored in the mathematical context. It is here that all values of the variables are visible along with the conditions that mark whether variables have a valid value or not. Because the adaptive IDA document is a prototype we have decided to just warn about invalid values, and let people see what they do regardless the outcome of the conditions. Of course an invalid value could also be denied as the new value for a variable. If an input value gets a new value, and another variable is called that depends on the



Input variables.

c3s2p3.polya	$3x^4+12x^3+5x^2-14x+4$	A polynomial used in divsion and GCD examples		**Condition**	**Result**	**Message**
			CAS-Conditions	**No CAS-conditions**		
			XPath-Conditions	count(.//OMV[@name='x'])=count(.//OMV)	true	This polynomial is only allowed to use x
c3s2p3.polyb	$4x^3+(6x^2-8x)+2$	A polynomial used in divsion and GCD examples		**Condition**	**Result**	**Message**
			CAS-Conditions	**No CAS-conditions**		
			XPath-Conditions	count(.//OMV[@name='x'])=count(.//OMV)	true	This polynomial is only allowed to use x
s1p1.a	9	A integer used in divsion and GCD examples		**Condition**	**Result**	**Message**
			CAS-Conditions	$(9 \geq 2) \wedge (9 \leq 10)$	true	A must be inbetween 2 and 10
			XPath-Conditions	not(.//OMS[@cd="arith1"])	true	no symbols from arith1 dictionary are allowed
s1p1.b	4	A integer used in divsion and GCD examples	No conditions found for this variable			
s1p1.c	18	A integer used in divsion and GCD examples	No conditions found for this variable			
s1p1.d	12	A integer used in divsion and GCD examples	No conditions found for this variable			

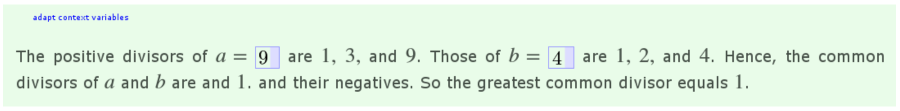
Non-input variables.

s1p1.ab_q	2	The quotient of s1p1.a and s1p1.b	No conditions found for this variable
s1p1.cd_q	1	The quotient of s1p1.c and s1p1.d	No conditions found for this variable
s1p3.ab_r	1	The remainder of s1p1.a and s1p1.b	No conditions found for this variable
s1p4.ab_gcd	1	The GCD of s1p1.a and s1p1.b	No conditions found for this variable
s1p4.cd_gcd	6	The GCD of s1p1.c and s1p1.d	No conditions found for this variable
s1p5.ab_lcm	36	The LCM of s1p1.a and s1p1.b	No conditions found for this variable
s1p6.ab_gcd_lcm_product	36	The product of GCD and LCM of s1p1.a and s1p1.b	No conditions found for this variable
s1p6.ab_product	36	The product of s1p1.a and s1p1.b	No conditions found for this variable

Figure 36: Mathematical context of a user

value of the first variable, then all variable nodes in between are being recalculated. This check is done by assigning a timestamp to each new value. Any variable that is called and depends on a variable with a newer value —directly or indirectly— will be recalculated. Note that this is actually a recursive process.

Another approach, as shown in Figure 37, is to equip an example used in the content with Formula Editor fields where the values of the variables are given. The user can then change the settings without leaving the theory page for the mathematical context. These values are changed everywhere in the document.



The positive divisors of $a = 9$ are 1, 3, and 9. Those of $b = 4$ are 1, 2, and 4. Hence, the common divisors of a and b are and 1. and their negatives. So the greatest common divisor equals 1.

Figure 37: Mathematical context of a user

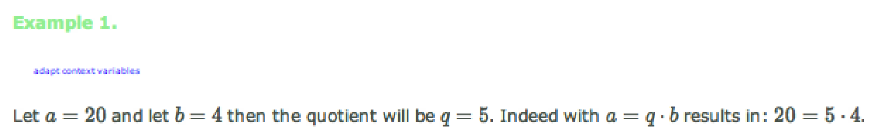
5.4.4.3 *Presentation model*

The content from the domain model combined with the context from the user model create the adapted pages for the user to read. The rules and queries that are needed for the adaptation form together the presentation model. In our context implementation, the rules and queries of the presentation model are implemented within the content. However, there is a distinction between the content and the queries and rules of the presentation model. For instance the Jelly code —including any custom tags—, XForms and some specialized fragments are responsible for any presentation tasks. Content is considered to be written by DocBook, OpenMath and text.

In this section we will focus on the implementation of adaptive techniques in the adaptive IDA document. We will elaborate on a number of examples.

Using variable graph variables in examples.
A seemingly simple but important adaptation technique is the use of

user chosen values as much in the document as possible. This will give the user a feel of control and recognition when a notion is being explained and computed using his or her values. It also grants the user the freedom to play around with the example using different input.

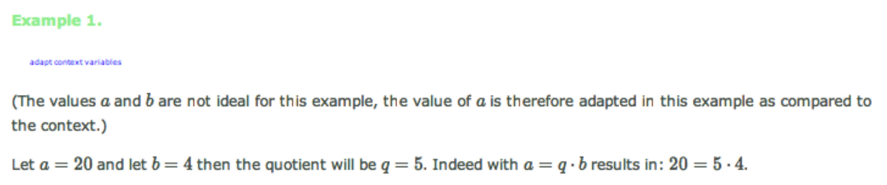

Example 1.

adapt context variables

Let $a = 20$ and let $b = 4$ then the quotient will be $q = 5$. Indeed with $a = q \cdot b$ results in: $20 = 5 \cdot 4$.

Figure 38: An example with variables from the mathematical context.

Figure 38 shows one of the first examples from the adaptive IDA document. Obviously in this example there is a problem when the integers do not properly divide: a situation that can be solved by a CAS-condition that enforces the divisibility on the values entered; however in this case a different solution has been chosen. In this example the code performs a test to determine if the two integers are divisible. If they are not, the remainder is subtracted and forces the two integers to be divisible again. Of course the user needs to know about this as it was promised to only use the user specified values. A screen shot of this altered variable values can be found in Figure 39.


Example 1.

adapt context variables

(The values a and b are not ideal for this example, the value of a is therefore adapted in this example as compared to the context.)

Let $a = 20$ and let $b = 4$ then the quotient will be $q = 5$. Indeed with $a = q \cdot b$ results in: $20 = 5 \cdot 4$.

Figure 39: An example with automatic adapted variables from the mathematical context.

If one remembers the rules for soundness, one may wonder about the example we just described. This example uses the remainder symbol, while this symbol is not yet introduced on this page nor on one of its predecessors in the theory graph, as there are no predecessors for division. Indeed without author action the soundness verification will fail for the division page. For these type of cases where an author needs something computed in the background and will only use the result —not the computation itself— an exception was added to soundness checks. If an OpenMath symbol gets the attribute *ignore* set to true, that symbol will not be taken into account.

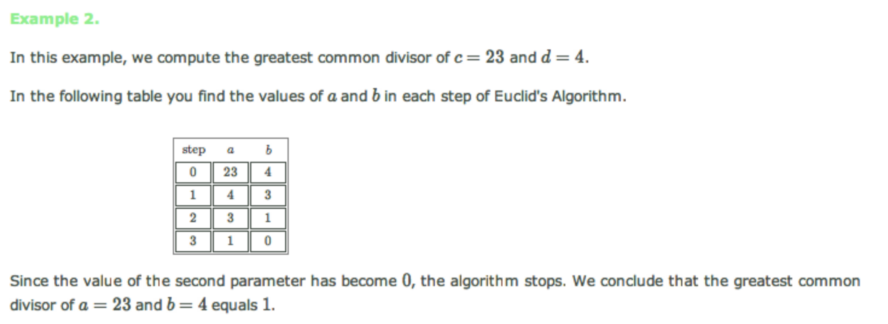
Fragments.
Fragments are used to include the same content more than once. As discussed before this is of use when content refers back to previous read content, but quite often fragments are also used to include content more than once on the same page. For example, exercises that use

random values. By including the same fragment a few times, the user will be presented with a set of similar but different exercises. Another example is a recursive algorithm. Each step may be performed by the same fragment. In Figure 42 the code in the *euclidesStep* fragment is given as used in the Euclidean Algorithm.

Listing 42: A fragment step in the Euclidean algorithm.

```
<mdc:fragment xmlns:mdc="jelly:org.mathdox.context.
    TheoryBlocksTagLibrary"
  xmlns:c="jelly:core" xmlns:x="jelly:xml">

  <!-- the new values for this iteration -->
  <c:set var="c" value="${a%b}"/>
  <c:set var="a" value="${b}"/>
  <c:set var="b" value="${c}"/>

  <!-- row in the table with the new values, with an optional
      step counter -->
  <tr>
    <c:if test="${countsteps eq 'true'}">
      <td class="euclides"><OMI>${step}</OMI></td>
    </c:if>
    <td class="euclides"><OMI>${a}</OMI></td>
    <td class="euclides"><OMI>${b}</OMI></td>
  </tr>

  <!-- determine if end condition is met, if not call the next
      step -->
  <c:choose>
    <c:when test="${b!=0}">
      <mdc:include-fragment
        fragment="fragments/rules/euclides/euclidesstep.mdf"
        frgid="step" var="result">
        <c:set var="a" value="${a}"/>
        <c:set var="b" value="${b}"/>
        <c:set var="step" value="${step+1}"/>
      </mdc:include-fragment>
      <!-- the found gcd is stored in the return variable result
          -->
      <c:set var="_result" value="${result}"/>
    </c:when>
    <c:otherwise>
      <!-- end of algorithm store the gcd-->
      <c:new var="_result" className="java.util.HashMap"/>
      <c:set target="${_result}" property="gcd" value="${a}"/>
    </c:otherwise>
  </c:choose>

</mdc:fragment>
```

In Figure 40 the example as it occurs on the Euclid page is displayed. Again this example works with variables from the mathematical context. Because this implementation of the algorithm performs the actual computation the result of this computation can be used by passing it back as a result value. This allows the fragment also to be used as an example that is disconnected from the mathematical context.



Example 2.

In this example, we compute the greatest common divisor of $c = 23$ and $d = 4$.

In the following table you find the values of a and b in each step of Euclid's Algorithm.

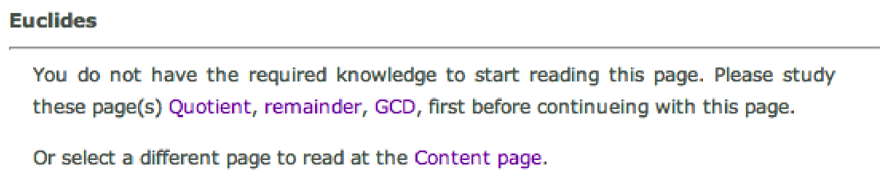
step	a	b
0	23	4
1	4	3
2	3	1
3	1	0

Since the value of the second parameter has become 0, the algorithm stops. We conclude that the greatest common divisor of $a = 23$ and $b = 4$ equals 1.

Figure 40: A Euclid example performed by recursive fragments.

This example demonstrates how fragments can call themselves or other fragments in order to create content. As such it is evidence how smaller pieces of content can be taken together to form a larger piece without losing coherence in the content.

Navigation.
Without a set order in which the theory pages have to be read, a user may need help in deciding what to read next.



Euclides

You do not have the required knowledge to start reading this page. Please study these page(s) Quotient, remainder, GCD, first before continueing with this page.

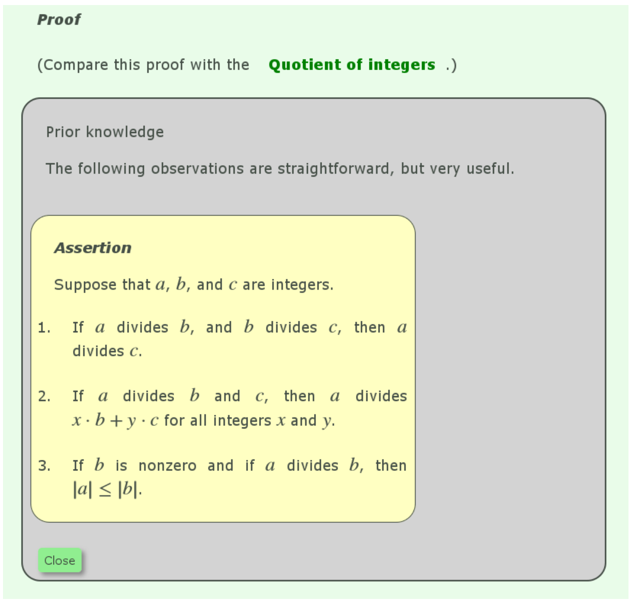
Or select a different page to read at the Content page.

Figure 41: Denied access because not all required knowledge is understood.

A solution to prevent a user from reading a page he or she does not possess all the required knowledge for is simply denying him or her access as done in Figure 41. Of course these redirections are directed at pages that do not need any prior knowledge to be read.

Instead of showing prior knowledge only when the page refers to it, a page can also decide to include all prior knowledge that is not considered to be understood by the reader at the opening of the page. This content is then sorted along the theory graph according to the following algorithm.

1. Take all available nodes (nodes that do not require any prior knowledge that is not yet understood); if this set is empty then the algorithm stops.


Proof

(Compare this proof with the **Quotient of integers** .)

Prior knowledge

The following observations are straightforward, but very useful.

Assertion

Suppose that a, b, and c are integers.

1. If a divides b, and b divides c, then a divides c.
2. If a divides b and c, then a divides $x \cdot b + y \cdot c$ for all integers x and y.
3. If b is nonzero and if a divides b, then $|a| \leq |b|$.

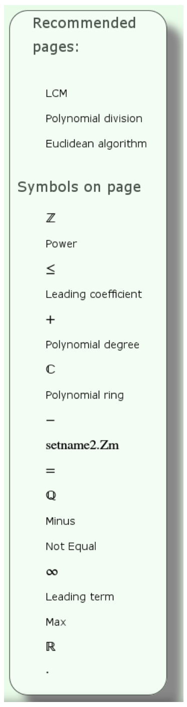
Figure 42: Content made available by means of stretchtext.

2. Choose one of these and include the definition of that page into the current page.
3. Mark the chosen page as pseudo understood.
4. Continue at step one.

Pseudo understood means that that page is only marked as understood during the execution of this algorithm. After the algorithm execution a page marked as pseudo understood, regains the not understood status.

Yet another way to help users to find content is to include a symbol list on a page. Users can choose to have this list displayed and it lists all symbols that are used on the page. The listed symbols link to pages of the symbol graph or OpenMath pages, each explaining the symbol in more depth. In Figure 43 a list of symbols is shown.

The IDA document also assists users in finding a route through the document towards a goal set by the user. The theory graph in the knowledge context as seen in Figure 32 is colored so as to indicate where the user should go next. Green indicates already read and mastered, blue a suggested choice to continue reading and red not yet recommended due to lack of prior knowledge. In combination with the theory graph it can then be determined which nodes are not yet mastered, and which subset has predecessors that are no longer understood. This subset will be the group with available theories while the remaining set will be marked as not yet available. If a learning goal is specified in the logistic context then those nodes that are both in the predecessor list of the goal and in the set of available nodes are



Recommended pages:

LCM
Polynomial division
Euclidean algorithm

Symbols on page

$\mathbb{Z}$
Power
$\leq$
Leading coefficient
$+$
Polynomial degree
$\mathbb{C}$
Polynomial ring
$-$
setname2.Zm
$=$
$\mathbb{Q}$
Minus
Not Equal
∞
Leading term
Max
$\mathbb{R}$
$\cdot$

Figure 43: A list of symbols as they occur on the theory page.

advised as the next step to reach the set goal. A plan model (see Section 3.1.6) has not been implemented in the adaptive IDA document, as it is considered to be too difficult to determine the goal of the user without asking.

Navigation also includes navigation to external pages. Based on the background of a user, links can be offered to external pages that provide more information about the topic at hand.

However, readers may also introduce error situations when they enter invalid values for variables. In such cases conditions should warn the users at the moment of entry.

5.4.4.4 *Soundness*

While creating an adaptive document an author needs tools to verify the soundness. It is easy to create consistency errors while developing and changing an adaptive document. Adaptivity will help users to understand content, but only if the content is adapted in a correct manner. Possible errors include the use of symbols or variables at an earlier page than where they were introduced. The chance for this to happen is considerable, especially since content is fragmented and used by a lot of different pages. The building process of a page is dynamic and because the content is being reused elsewhere in the document, its hard to keep everything correct. It is easy to lose track and let these kind of errors slip in.

To this end all pages are scanned to determine what content, operators and variables have been used. These results are then compared to what the document file, the theory and symbol graph indicate is legal to use for each page. As user context cannot and should not[4] be taken into account the scan process includes all possibilities.

The results from the scan are summarized and checked for inconsistencies with regard to the pages, operators and variables. The findings are then shared with the author as a graph or a list.

Soundness also applies to the values of variables. However, this is a different kind of soundness check as the variable values belong to a user and not to the document. Therefore the result differs from user to user. The values of a user are verified by the conditions as listed on variables input page.

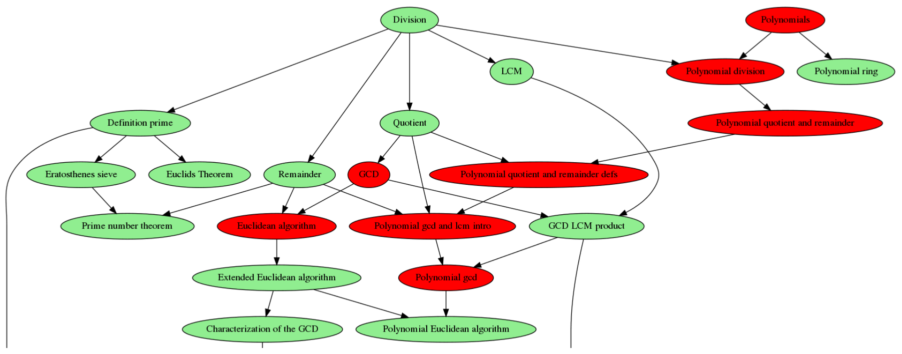


Figure 44: Theory nodes that fail soundness checks.

Figure 44 shows a (partial) theory graph as can be found on the Theories inspection page. All nodes colored green are considered sound, while the red nodes have problems associated with them. More elaborate information in regard to these nodes can be found further on the page. In relation to Figure 44 there are three different kinds of problems that can be identified. Firstly, most notable, the operator ring3.poly_ring is used in Polynomials and subsequent nodes. However the Polynomials node in the theory graph failed to introduce it. As a consequence whenever the operator is used, it is marked as a potential error. Secondly, the node GCD fails to connect to Polynomial gcd and lcm intro as a result the operator arith1.gcd which is introduced at GCD is not available in Polynomial gcd and lcm intro. Clearly this is an error in the graph and should be repaired. Thirdly, the Euclidean algorithm is being used by Euclidean algorithm but also by Polynomial gcd. In the first case the Euclidean algorithm is only applied to integers while in the second case it applies to polynomials. Since both pages use the same code, the polynomial3.remainder symbol ends up on the Euclidean algorithm page.

4 Soundness check occurs at the document level for two main reasons. Firstly as it is impossible to verify soundness at user level with ever changing user contexts and secondly as to warn the author to any (future) error condition that might occur when a user goes through the document.

The page limits the example to integers only so for users there is not a problem for the users. Still the author is being alerted to the existence of the potential error.

6 CONCLUSION

"Complexity is often spawned from simplicity."
–author unknown

Complexity is a relative notion, allowing quite often for a breakdown in more basic parts. Obviously the reverse also holds. Combine simple concepts and a more complex notion is created, which, when the action is repeated, will become more and more complex while all its subparts remain basic.

The mechanism of knowledge decomposition and knowledge composition is an important aspect of our project. Knowledge (de)composition happens in the minds of authors and users [173]. It also plays a paramount role in our adaptive documents, where it is decided which content is combined and how it will be represented to help the user in the knowledge composition process.

The MathDox Context has been designed (see Chapter 4) and created as a working prototype (see Chapter 5) with this process in mind. The various components like the theory graph, symbol graph, variable graph, fragments, queries and rules reflect the knowledge decomposition.

An advantage of the decomposition is the reusability of the individual components. This addresses one of thirteen requirements for an adaptive mathematical system as listed in Section 3.4. In this conclusion we will first discuss to what extent we meet these requirements. Next we discuss the need for a user evaluation and finally we present four suggestions for further research.

6.1 MEETING THE REQUIREMENTS OF A MATHEMATICAL ADAPTIVE SYSTEM

In Section 3.4 requirements were stated for an adaptive mathematical system. Obviously the MathDox Context needs to be measured to these requirements as well. Because the context implementation is built directly on top of the MathDox Player it benefits from all characteristics of the MathDox format, see Section 2.4.

Interactivity.
The MathDox player offers interactivity by including the Jelly [47] format and the MathDox Formula Editor [62]. The MathDox Formula Editor grants the ability to process answers given by the user. Thanks to the MathDox Context, the answer can take into account the capabilities of the user as well as the knowledge gathered by the user. The

context also creates the possibility of running examples. The soundness checks enables guidance of the user by keeping newly set variables and values consistent throughout the document.

Usable in multiple formats.
The context implementation inherits its format from MathDox. As such it benefits from the XML structure that allows transformations to other formats such as LaTeX and PDF. Naturally a transformation to a paper version will lose adaptivity. However, these transformations can be applied to a MathDox document which has been created on basis of information in the context. In this way preferences set by the user as well as running examples created by the user can be transformed.

Being extendable.
External services can be included and used from within MathDox code. Several examples exist including services for Geogebra, JSXGraph and computer algebra systems. New external services, like for instance to a theorem prover, are also possible. New fragments and Jelly custom tags are easy to write and to add to existing libraries. If the provided API does not suffice, it can be extended by new classes that either extend the current ones or interfaces with them to add extra functionality.

The MathDox Context has been implemented in a general way. For instance, the theory-, symbol-, variable graphs are author defined, and are not mandatory. The use of variables or event logging is not necessarily used nor limited to a predefined standard set.

Finally, the MathDox format itself is open source, and suitable to any kind of extension, as proven by the LeActiveMath [51] and the WebALT [104] projects.

The MathDox Context can be used as a starting point for future extensions towards documents that work with abstract objects that are well structured and may be encountered in chemistry, computer science and physics.

Representation of mathematics.
The MathDox player provides good ways to render mathematics. This rendering can be dependent on context information. For example the complex number i can be rendered as j for students in electrical engineering.

Ease of usage.
The MathDox format consists of several sub-formats and the implementation of the MathDox Context adds another. It is not surprising that the mixture of these sub-formats is a bit daunting and complex. MathDox code is hard to work with for inexperienced authors. Furthermore, the sub-formats require separate translation steps, slowing

down performance of the server. The addition of fragments — which requires extra translations — does not improve performance. Yet, at the same time, fragments and the Jelly custom tags do offer users ready to use functionality, like queries, rules and templates.

Use of open standards.
The MathDox format, the MathDox Player and the MathDox Context are all based on open (XML) standards and are all open source. This includes external applications that are being used, such as Tomcat [101] and eXist [22]. Open source code allows for inspection and verification of the code, low-cost installations and changes to the code when it is deemed necessary for applications.

Reusability of content.
The MathDox Context contains a library of Jelly custom tags and fragments, which can be expanded upon. For example, a fragment containing an exercise or explanation about a definition can be plugged into any MathDox page or document.

Also the context of documents can be reused. Documents can share one or more of the context graphs. To promote reusability the files of the theory graph, symbol graph and variable graph are identified and retrieved by URIs. Quite often this means they are directly requestable from the MathDox Player that serves the document they belong to. Reuse of parts of the context of course requires compatibility and soundness checks. The soundness checks that have been implemented can be used to verify if a (partial) document is suitable for reuse.

Stability.
Stability is about how well the document performs seen form the user's perspective. Technical problems, like bugs and performance issues are an obvious threat. Our context implementation has still room to grow in terms of technical issues and performance. Another threat comes with changing nature of adaptive documents, as they might — again from the user's perspective — unexpectedly change content. It is up to the author to decide on the level of stability in the document, as adaptivity on a MathDox page is the direct result of the code on that page. Of course, this is strongly related to the quality of adaption.

Quality of adaption.
To ensure this quality of adaptation, a document requires sufficient and meaningful data. To this end an author is at liberty to add extra information to the context as needed or not to use some tools and aspects of the context at all. The theory, symbol and variable graphs are designed to be used independently from each other and may be omitted. The author also decides which variables are used and are needed to be in the user model.

Sharing user data.
Sharing data between adaptive mathematical documents eliminates the need to ask the same questions when a new document is being visited by the user. Before data can be shared the author should be aware of any compatibility issues. In case of data as stored in the logistical context this is rather straightforward process. Any data stored in relation to the theory graph, symbol graph or the variable graph, needs first to be verified, both in compatibility as in correctness. (Various of these checks will be done automatically by the system.)

Management of user data.
The user is responsible for the contents of the logistical data. The mathematical user data consists of the values of the variables appearing in the variable graph. Our context implementation provides various checks on these values to ensure they are within the boundaries of the document. The knowledge context is somewhat different, as the knowledge context contains data (mostly events) that are analyzed by the system. The user has no influence on this analysis.

Privacy.
Authentication and storage of data in MathDox Context is delegated to the eXist [22] database. Each request to the database requires the credentials of the user, keeping the responsibility of the data with the database. MathDox does not store any user passwords in between sessions, only during a session. This gives access of user data only to the user and the administrator.

Robust.
The MathDox Context has been built on top of reliable software components such as Orbeon Forms, MathDox Player, eXist and Tomcat. The context itself consists of tag libraries and three Java classes, the ContextDocument object, DatabaseAccessObject and the GraphManager. These classes are independent of each other and their responsibilities are divided. This modular approach makes the design of the system robust.

6.2 USER EVALUATION

Our research focused on the design of an adaptive mathematical document and a prototype, rather then the effect of the adaptivity on users. Nevertheless we seized the opportunity to see system at work and gain some experience from potential authors, teachers and users. Feedback on the MathDox Context implementation, as available in our proof of concept [37], was asked for and received by a group of mathematics and physics teachers at the Dutch high school Copernicus Scholen Gemeenschap in Hoorn. These teachers have students from 12 till 18 years old. The general consensus in this group of teachers was that the MathDox Player, enriched with context, is a powerful tool that certainly helps students in comprehending and speeding up their studies. Some well-liked features included tests to verify whether a concept was understood or not, the visualization of the theory graph with colors indicating whether concepts were mastered or not and (running) examples that could be adapted to work with user specified values. Change of contents as caused by adaptivity was not noticed straightaway, but appreciated after it was pointed out. However, some drawbacks were also mentioned, notably that younger students are not expected to be mature and disciplined enough to work with electronic materials without teacher supervision, ruling out MathDox materials as a primary source of content for their studies.

A recommendation of the group is that materials are considered to be more useful if the adaptivity is able to adjust to as many different situations as possible, which were predicted to be quite a lot. This suggests a very fine grained theory graph and increased analyzing and adaptive possibilities in used fragments.

The group of teachers was also shown some code examples. The consensus was they would love to use MathDox materials in their classes, but were hesitant to create any themselves because of the required coding. These observations confirmed our expectations. At the same time the observations emphasized the different roles that come in play when MathDox materials are used in classroom conditions. Among these — in increasing levels of technical experience — are the roles of a classroom supervisor, teacher, author and programmer. Especially the author and programmer roles are needed for the creation of the required materials.

6.3 FURTHER RESEARCH

Mathematical documents come roughly in three flavors:

- *Static*: content does not change or adapt;
- *Interactive*: content allows for interaction, usually limited to exercises, or changeable example.
- *Adaptive*: the content in documents adapts to the user, enabling them to read and interact with the document within a context best suited for the individual user.

There are not many adaptive mathematical systems available. This prompted us to design and implement the MathDox Context. it is a first step towards a truly adaptive and interactive mathematical document. The changes and adaptations to the user's most current context are achieved by utilizing the interpreter nature of the MathDox Player, which enables document pages to change upon each page request and gives users the freedom of navigation in a document.

As we have seen in Subsection 3.4, MathDox documents enriched with context meet more of the requirements that we have formulated for interactive mathematical documents. Context enriched MathDox documents are highly interactive and adaptive. However, work still needs to be done in some areas.

We have implemented a proof of concept of our design [37]. In order to turn this proof of concept into production, several tasks have to be done. First of all, technical issues like performance and optimization, as well as issues concerning reliability have to be taken care of. Moreover, extending our system with various new external services should be addressed. These services can range from databases to proof checkers and theorem provers.

Secondly, how to make best use of adaptivity in mathematical documents has to be investigated. Adaptive mathematical documents can be used in various situations, but due to the lack of good examples there has been very limited information on the effectiveness of their use. A study on how to use adaptive mathematical documents in for example a classroom or a blended learning environment, would be of high interest to us and could provide valuable information on how to use and extent our content implementation. Our user evaluation as described in Subsection 6.2 is a first tiny step in this direction.

Usage of MathDox documents and context by authors should also be addressed. Although, we have tried to make authoring of context enriched documents easier by adding macros, tags and fragments, we still lack a good authoring environment. In order to make our system available and attractive to a larger set of authors it is essential to have good authoring tools. What a good authoring system is, needs also further research.

Finally, to improve our context implementation, one might consider to use proof checkers and theorem provers in the creation and soundness checks of the various context graphs. Indeed, the soundness of a theory graph can possibly be verified by involving such a proof checker. But such a theory graph and the corresponding symbol and variable graph could also be produced by a theorem prover. The use of provers for adding and checking a context of a document is a topic to investigate further.

Part I

APPENDIX

A MANUAL

To assist authors in creating documents in the MathDox Context system we provide some extra information regarding the technical aspects of the system. We explain how these documents can be created and maintained. We limit ourselves to the MathDox context system. The basics of the MathDox system can be found in the MathDox Player manual [61]. We will provide references to the MathDox Player manual when they are of importance.

In Section A.1 we start with a list of steps that are required to create a new document. This list is meant as an overview and does not contain much background information. In Section A.2 the reader is presented with a tutorial to implement the steps from Section A.1. In Section A.3 more attention to the technical details is given. Finally in Section A.5 a list of predefined queries and rules is presented along with documentation on how to use these within a document.

A.1 HOW TO SET UP A NEW DOCUMENT

We present a list of steps that help authors to create a new document. These steps aim to cover all aspects of a new document but are considered guidelines as they are not meant as a fixed set of rules. The list is divided into subparts, the first deals with the installation of the required software, the second addresses the necessary preparations before content can be written. In the third step content is written which is then finalized and prepared for deployment in the fourth step. At step five the content is deployed and can then be viewed by the MathDox Player. The first two steps are optional if they have been executed earlier.

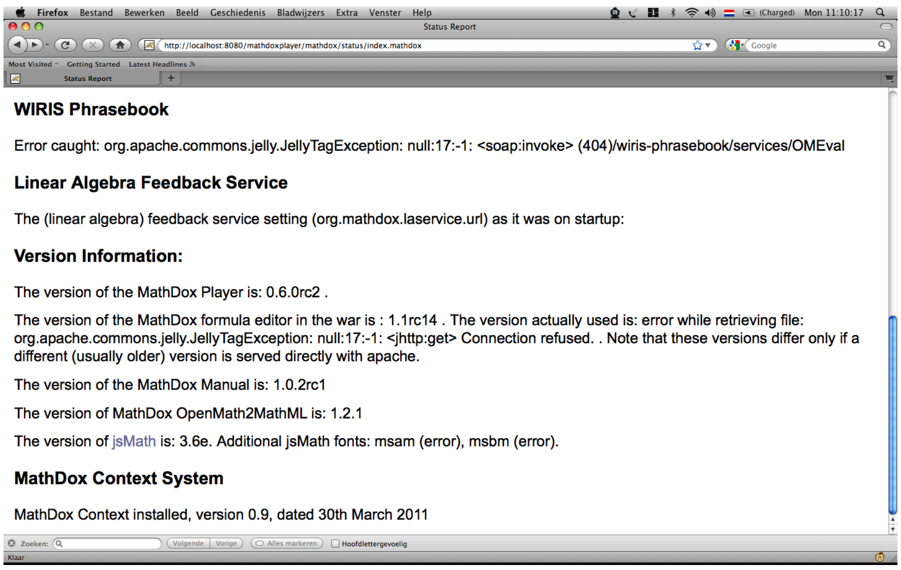
Setup of the Software.
These steps guide the reader through the installation and deployment of the MathDox Player and the MathDox Context add-on.

- *Install required software.*
  Before the MathDox Player is installed it is required to install some supporting software packages. These include Java [42], Ant [4], Tomcat [101], eXist [22] and phrasebooks for computer algebra systems as Mathematica [63], Maxima [69] or GAP [27]. Refer to their respective manuals for the instructions.

- *Install and deploy the MathDox Player.*
  Install the MathDox Player. Refer to the MathDox Manual [61] for instructions.

- *Add the MathDox Context add-on.*
  Copy the MathDox Context jar to the `libraries/context` directory of the MathDox Player and restart the Tomcat server.

- *Test if the MathDox Player works correctly.*
  Open up the status page, as shown in Figure 45 within the MathDox Player and verify that all tests successfully pass. If one or more fail, fix those problems before continuing the next steps. Refer to the MathDox Player manual [61] for assistance.


Firefox Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Venster Help
(Charged) Mon 11:10:17
Status Report
http://localhost:8080/mathdoxplayer/mathdox/status/index.mathdox
Google
Most Visited Getting Started Latest Headlines
Status Report
Zoeken: Volgende Vorige Alles markeren Hoofdlettergevoelig
Klaar

WIRIS Phrasebook

Error caught: org.apache.commons.jelly.JellyTagException: null:17:-1: <soap:invoke> (404)/wiris-phrasebook/services/OMEval

Linear Algebra Feedback Service

The (linear algebra) feedback service setting (org.mathdox.laservice.url) as it was on startup:

Version Information:

The version of the MathDox Player is: 0.6.0rc2 .

The version of the MathDox formula editor in the war is : 1.1rc14 . The version actually used is: error while retrieving file: org.apache.commons.jelly.JellyTagException: null:17:-1: <jhttp:get> Connection refused. . Note that these versions differ only if a different (usually older) version is served directly with apache.

The version of the MathDox Manual is: 1.0.2rc1

The version of MathDox OpenMath2MathML is: 1.2.1

The version of jsMath is: 3.6e. Additional jsMath fonts: msam (error), msbm (error).

MathDox Context System

MathDox Context installed, version 0.9, dated 30th March 2011

Figure 45: The MathDox Player status page

Preparations for content.
At this point one must have a working MathDox Player with the MathDox Context add-on before continuing with the preparations for content creation. If this is not the case please consult the previous steps. The steps in the remainder of this section are meant as general guidelines, a more precise tutorial can be found in Section A.2.

- *Create a document home location.*
  To organize all files belonging to a context document a new folder needs to be created. Regardless the name the folder has been given, the folder will be called the *document home folder* in this manual as it is the home of all files belonging to the document.

- *Create the folder structure in the document home folder.*
  We advise to copy the layout as described in Section A.3.1. This layout is assumed in the remainder of this manual. A script has been written to help in creating this structure. To use this script copy the files *contextstructure.zip* and *build.xml* followed by the

command `ant setuplayout`, executed from the document home folder.

- *Create the theory graph.*
  The next step actually is the very first step in terms of creating content for the document. Creating the theory graph is important in the sense that it determines the next layout steps in the folder structure, but more importantly it also requires the author to explicitly divide the document contents into separate parts. The theory graph file needs to be placed in the folder `documenthome/graphs/src/xml/theorygraph`. There are no restrictions on the name.

- *Create the file structure for the nodalpages.*
  Having created the theory graph in the previous step, it is time to use this theory graph with a script that expands the file structure to encompass the nodalpages that have to be created. To do this; run an ant script from the document home folder: `ant expand-nodalpages`.

Creation of content.
Now all preparations have been done, it is time to start creating some content that is to be included from the various pages.

- *Create nodalpages & fragments.*
  Each nodalpage has a separate folder in the nodalpages folder. Inside each of these folders a file called *index.md* is located. This is the starting point of a nodalpage.

- *Create the symbol and variable graphs.*
  When some content is written, symbols and variables will undoubtedly have been used. These have their own graphs, and these graphs need to be created.

  .

- *Repeat previous two steps.*
  Repeat from step *Create nodalpages & fragments* until this step is more or less done in the rough form.

Finalize and prepare for deployment of content.
Before the content can be deployed and tested in the MathDox Player, a few more steps have to be taken. Among these is the previously postponed addition of the edges in the variable and symbol graphs.

- *Create edges in the graphs.*
  The edges in the theory and symbol graphs need to be created. Note that the edges for the variable graph are automatically created based on the edges content.

- *Finalize the graphs.*
  With the edges and properties added to the graphs, the graphs need to be finalized by having them converted and encoded into the right formats ready to be used by the document. These conversions are done by an ant script and is executed by typing `ant buildall`. Mind that the result files are placed in the *graphs/build/encoded* and *graphs/build/jellyprepared*.

- *Create document file* The document needs a database identifier, a home folder: the locations of the graphs it will use and which symbols are considered to be prior knowledge to this document. Specify this in the document file.

Deploy content.
With the content written and the graphs created in previous steps, it is time to deploy the content so that it becomes accessible through the MathDox Player.

- *Copy the document into the MathDox Player.*
  Create a link or copy the documents contents into the MathDox Player content folder. Pay attention to non content files, such as ant files and style sheets. These should not be made available in the MathDox Player.

- *(Re)start Tomcat.*
  Restart the Tomcat server to clear the cache. Verify if Tomcat has successfully started by going to the document in a browser. To increase performance the theory, symbol and variable graphs are cached as are the values of variables. When new content comes active, purge the cache as to force the MathDox Context system to use the latest versions of the graphs and variables.

- *Create users.*
  To create users first specify the default values for the variables as stored in the logistic and mathematical context. Then visit the document in a browser and go to `inspectiontools` and to `Create new user`.

- *Fine tune and finalize.*
  All major steps have been done in order to create a document. Now fine tune your document. As some issues may only become visible after inspection of the pages in the MathDox Player, this step is quite important.

A.2 TUTORIAL

In Section A.1 a list of steps is presented that guide authors through the required steps to create a context enriched mathematical document. In this section we are going through the steps of the previous section again but with more detail. While doing so we create an example document. In this tutorial it is expected that the author has installed all required software as described in Section A.1.

Preparations for content.

- *Create a document home location*
  Create a document about group theory. This folder will be called the *home folder* and should not be located within the content directory of the MathDox Player. The folder will contain some files that are not suitable to the MathDox Player. At a later stage the contents of this folder will be copied to the MathDox Player.
- *Create the folder structure in the document home folder*
  Having created the home folder `grouptheory`, create the sub folders that are to be used for content files later on. This task is automated and can be done by unzipping the zip file `contextstructure.zip`. Along with sub folders an ant file is also unzipped. This ant file will offer automation to some of the other tasks at hand.
- *Create the theory graph*
  The next step is to create the first graph from the domain model, the theory graph. Adding a partial ordering in the graph at this point is optional and can be done later if desired. Go the the folder `graphs\src\xml\` and create a new folder called `theorygraph`. Go into the newly created `theorygraph` and create a new file called `theorygraph.xml`. A theory graph takes the form of a GraphML file with added graph properties and nodes and edges as shown in Listing 43. In Listing 44 the required properties are listed, place them directly under the comment line

  `<!-- GRAPH PROPERTIES -->`.

  Do the same with the nodes as listed in Listing 45 and place that code under the comment line

  `<!-- GRAPH NODES -->`.

  Finally the edges in Listing 46 go under

  `<!-- GRAPH EDGES -->`.

Listing 43: The layout of a theory graph

```
<graphml
  xmlns="http://graphml.graphdrawing.org/xmlns"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
   http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd"
>

```

```
  <!-- GRAPH PROPERTIES -->

  <!-- Graph definition -->
  <graph edgedefault="directed">

    <!-- GRAPH NODES -->

    <!-- GRAPH EDGES -->

  </graph>
</graphml>
```

Listing 44: The properties of a theory graph

```
<!-- property declarations -->
<key id="links" for="node" attr.name="links"
     attr.type="string"/>

<!--
  url property, allows external links to be attached to a
      nodal page
-->
<key id="url" for="node"
     attr.name="url" attr.type="string">
  <default>false</default>
</key>

<!--
  OMS property, defines all required symbols that need to
  be understood by the reader when reading the current
  nodal page.
-->
<key id="oms" for="node" attr.name="oms"
     attr.type="string"/>
```

Listing 45: The nodes of a theory graph

```
<!--
  Node definitions
-->
<node id="group"/>
<node id="subgroup"/>
<node id="permutation_group"/>
<node id="orbit"/>
<node id="permutation"/>
<node id="symmetric_group"/>
<node id="orbit_stabilizer_theorem"/>
<node id="stabilizer"/>
```

Listing 46: The edges of a theory graph

```
<!--
  Edge definitions
-->
<edge id="subgroup—permutation_group"
  source="subgroup" target="permutation_group"/>
<edge id="permutation_group—orbit"
  source="permutation_group" target="orbit"/>
<edge id="permutation_group—stabilizer"
  source="permutation_group" target="stabilizer"/>
<edge id="stabilizer—orbit_stabilizer_theorem"
  source="stabilizer" target="orbit_stabilizer_theorem"/>
<edge id="orbit—orbit_stabilizer_theorem"
  source="orbit" target="orbit_stabilizer_theorem"/>
<edge id="group—subgroup"
  source="group" target="subgroup"/>
<edge id="group—symmetric_group"
  source="group" target="symmetric_group"/>
<edge id="permutation—symmetric_group"
   source="permutation" target="symmetric_group"/>
<edge id="symmetric_group—permutation_group"
   source="symmetric_group" target="permutation_group"/>
```

When the theory graph is written, it needs to be processed in order to deal with encodings. To this end, type the following command: `ant encode` in the home folder. It will create an encoded theory graph in the `graphs/build/encoded` folder. Create a folder called `graphs/theorygraph` and copy the `theorygraph.xml` from the `graphs/build/encoded` folder to the `graphs/theorygraph` folder. This will be the location the document later will look for. Visual verification of the theory graph is done by creating a SVG file from the theory graph. This is done by running the following ant script: `ant dot-theorygraph`. The result can be found in `graphs/build/svg`. For more information about the theory graph and the other domain graphs see Section A.4

- *Create the structure for the nodalpages*
  Having the theory graph created, the folders that will contain the nodalpages are to be created in accordance with the theory graph. Each node from the theory graph will have its own folder that contains all files directly belonging to that nodalpage. Creation of these nodalpage folders is automated and can be done by calling an ant script from the document home folder like this: `ant expand-nodalpages`. This script will also create an `index.md` file in each nodalpage folder with the most used namespaces defined and a title that corresponds with the node.

Create content.
At this point all preparations for a new document have been done. We created the location where the document will be developed, created a

theory graph and created the file structure required by the MathDox Context System. The logical next step is to start writing the content that goes into the document.

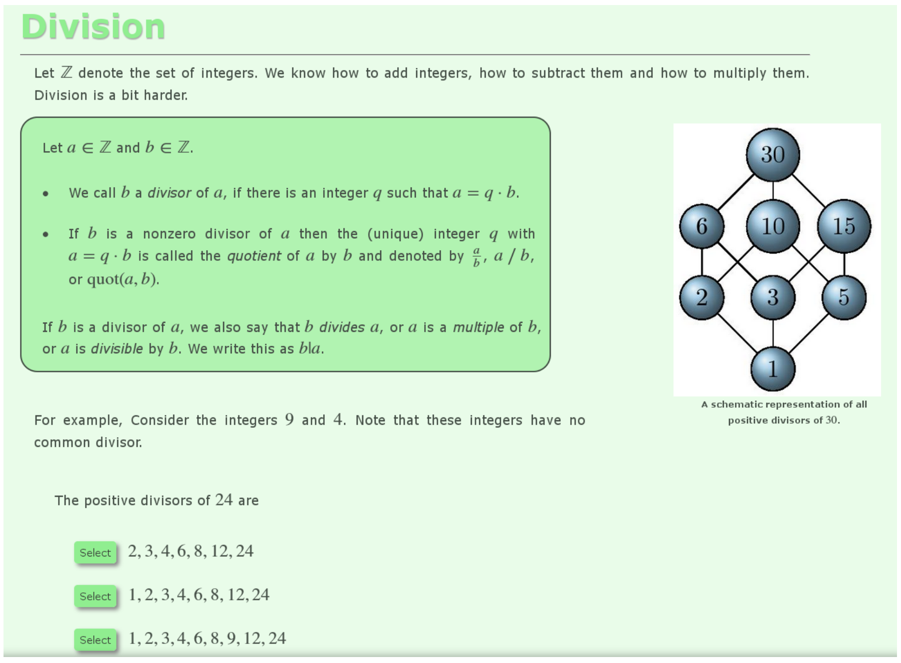
- *Create nodalpages & fragments*
  Nodalpages are the starting points to the various sections of a document. Each nodalpage should therefore be designed as if it were a small document of its own. To this end it is good to observe that while a nodalpage suggests just one page, there can be more than just the one page for a node. If more pages are required, it is wise to have at least one of these pages called `index.md` as this page will be seen as the starting point.

  As each nodal page is seen as a separate document there will be cases in which content is repeated. Do not fear repetition but instead use it wisely. A good way to do so is by defining a fragment file that includes the content that is to be repeated. Instead of writing the same content again later, it will suffice to just include the fragment where ever the contents are needed. Fragments can include fragments of their own, however prevent fragment circle inclusion. All content — be it in a nodal page or in a fragment — is written in MathDox.

  In this tutorial we create a nodalpage for the theory node group, see Listing 47.

Listing 47: Nodal page group

```
<article xmlns:mdc='jelly:org.mathdox.context.
    TheoryBlocksTagLibrary' xmlns:c='jelly:core' >
  <title>Division</title>

  <mdc:include-fragment
       fragment="fragments/division/intro.mdf"
       frgid="intro"/>

  <mdc:include-fragment
       fragment="definitions/division/index.mdf"
       frgid="defquot"/>

  <para>
    For example, Consider the integers
    <mdc:get-value variable="a" symboltableid="division"/>
    and
    <mdc:get-value variable="b" symboltableid="division"/>.
    Note that these integers have no common divisor.
  </para>

  <mdc:include-fragment fragment="exercises/division/
      exercise1.mdf" frgid="exercise1"/>

</article>
```



Division

Let $\mathbb{Z}$ denote the set of integers. We know how to add integers, how to subtract them and how to multiply them. Division is a bit harder.

Let $a \in \mathbb{Z}$ and $b \in \mathbb{Z}$.

- We call b a *divisor* of a, if there is an integer q such that $a = q \cdot b$.
- If b is a nonzero divisor of a then the (unique) integer q with $a = q \cdot b$ is called the *quotient* of a by b and denoted by $\frac{a}{b}$, $a\,/\,b$, or $\mathrm{quot}(a, b)$.

If b is a divisor of a, we also say that b *divides* a, or a is a *multiple* of b, or a is *divisible* by b. We write this as $b|a$.

A schematic representation of all positive divisors of 30.

For example, Consider the integers 9 and 4. Note that these integers have no common divisor.

The positive divisors of 24 are

$2, 3, 4, 6, 8, 12, 24$

$1, 2, 3, 4, 6, 8, 12, 24$

$1, 2, 3, 4, 6, 8, 9, 12, 24$

Figure 46: The division nodal page

The use of fragments.
In Listing 47 and Figure 46 an example group nodalpage is shown. The code of this page includes a fragment `<include-fragment fragment="intro.mdf" frgid="intro">`. The `fragment` attribute contains the web address of the fragment, located within the same document or elsewhere on the internet. The `frgid` attribute is used to identify included fragments. One should make sure that all fragments included from the same source file have non identical names. Identical names but included from different files — i.e. two different fragments or a nodalpage and a fragment — are not a problem. The code of this fragment is shown in Listing 48. Fragments may take arguments, work with local variables and may return values, how this works is explained in more detail in Section A.5.1.3.

Listing 48: Fragment code intro

```
<mdc:fragment
     xmlns:mdc="jelly:org.mathdox.context.
         TheoryBlocksTagLibrary">
  <para>
    Let
    <OMOBJ>
      <OMS cd="setname1" name="Z"/>
    </OMOBJ>
    denote the set of integers.
    We know how to add integers, how to subtract them and
        how to multiply them.
    Division is a bit harder.
  </para>

  <mediaobject>
    <imageobject>
      <mdc:get-dochome var="docHome"/>
      <imagedata
        fileref="${docHome + '/images/s1p1/hasse30.jpg'}"/>
    </imageobject>
    <caption>
      <para>
        A schematic representation of all positive divisors
            of
        <OMOBJ>
          <OMI>30</OMI>
        </OMOBJ>
      </para>
    </caption>
  </mediaobject>
</mdc:fragment>
```

The use of a variable.
There are two kind of variables that can be used in a fragment or on a page. The first kind are the normal Jelly variables, while the second are the context variables such as the mathematical variables and logistic variables. Here we will only discuss the context variables. The use of a context variable is done by `<mdc:get-value variable='a' symboltableid='division'/>` as shown in lines 12 and 14 in Listing 47. There are two attributes that are needed for this tag, `variable` and `symboltableid`. The first, `variable` is used to indicate the name of the variable requested, while the second, `symboltableid` indicates at which nodal page this variable is introduced. This implies that variables can be reused in nodal pages that come after the current nodal page in the theorygraph.

Logistic context variables.
Logistic context is accessible in the exact same way as the variables from the variable graph, with the only difference that they all are linked to the `symboltable='root'` setting. the `root` value indicates that these variables are not to be found in the mathematical context or variable graph but in the logistic context. As a consequence these logistic variables are in each and single nodalpage accessible.

- Create the symbol and variable graphs
  The content created so far has used variables and symbols. These need to be put in symbol and variable graphs. To assist with this

process, content written is to be analyzed and used symbols and variables will be placed into the symbol and variable graphs. To start this process the author needs to type the commands

`ant create-vargraph-nodes`

and

`ant create-symbolgraph-nodes`.

Run these commands in the home directory of the document and it will create a folder `tempgraphs` in the `graph` directory with inside the `tempgraphs` folder the two graph files will be created. The resulting symbol graph and variable graph files will only contain the nodes, they will not contain any edges, nor do they contain any properties. These need to be added at a later moment because at the moment the content is still changing, and with changing content the symbol and variable graphs will change also. Be careful that the graphs can be accidental overwritten when content changes and the unfinished graphs are replaced by newer versions.

- *Repeat previous two steps*

  As content is changed and adapted there is also a need for updating the symbol and variable graphs. By repetitions of the last two steps content can be formed to the wishes of an author. At this point the author cannot review the content in the MathDox Player, this will come at a later moment.

Finalize and prepare for deployment of content.

The content has been written and the theory, symbol and variable graphs have been partial written. The next step is to prepare the document for deployment in the MathDox Player. For this we need to finalize the graphs.

- *Create edges in the graphs*
  Nodes in the theory, symbol and variable graph need to be connected to each other by edges. As this depends heavily on the readers work, for example the amount of used theory nodes, variables and symbols, it is not possible to give a step by step guideline. Therefore we will describe the general process.

  Copy the latest graph files from the directory `theorygraph` for the theory graph and `tempgraphs` for the symbol and variable graphs to `graphs/src/xml/variablegraph` and `graphs/src/xml/symbolgraph`.

  To add edges to the theory graph, refer to Listing 46. Added edges should contain an identifier a source and a destination. Edges for the symbol graph are added in a similar way.

The variable graphs is a bit different, as it does not need edges to be added manually. Instead the edges of the variable graph are determined from the mathematical expressions that need to be added to the nodes. It is these expressions that tells the MathDox Context system how to calculate the values every time when a variable is requested.

- *Finalize the graphs*
  All graph source files are now located in sub folders of the `graphs/src/xml/` directory. Before they can be used by the document some values of the properties in the graphs need to be encoded. For this there is another ant script available. Since we also want our graphs to be visible in MathDox documents, just encoding will not be enough, SVG versions of the graphs are also needed. The command `ant build-all` will produce the document ready graphs and write them to the respectively `graphs/build/theorygraph`, `graphs/build/symbolgraph` and `graphs/build/variablegraph`folders. If the result is acceptable they then need to be copied to the `graphs/theorygraph`, `graphs/symbolgraph` and `graphs/variablegraph` folders. The `build-all` command will also create the SVG representations of the graphs. These will be stored in the `graphs/buid/svg` folder. Some Jelly enriched variations of SVG graphs will also be created. These are for sake of soundness checks and tracking user progress through the document.

- *Create the document file*
  Before the content can be copied the MathDox Player content folder it first needs a valid `document.xml` file. This file should look similar to Listing 49.

Listing 49: The contents of the document.xml file

```
<document docid="grouptheory">
  <documenthome>grouptheory</documenthome>
  <theorygraph>grouptheory/graphs/theorygraph/theorygraph.
      xml</theorygraph>
  <symbolgraph>grouptheory/graphs/theorygraph/symbolgraph.
      xml</symbolgraph>
  <variablegraph>grouptheory/graphs/theorygraph/
      variablegraph.xml</variablegraph>
  <prior-knowledge>
    <required-symbols>
      <OMS cd="arith1" name="minus"/>
      <OMS cd="arith1" name="plus"/>
    <required-symbols>
    <required-CDs>
      <required-CD cd="prog1"/>
    <required-CDs>
  </prior-knowledge>
```

```
</document>
```

Important elements are:

– `<document docid="grouptheory">`

  Which gives each document an unique identifier to be used by the MathDox context system to distinguish between different documents.

– `<documenthome>grouptheory</documenthome>`

  Tells the MathDox context system where the files of the document are located.

– `<theorygraph/>`
  `<symbolgraph/>`
  `<variablegraph/>`

  The location of the theory symbol and variable graphs.

– `<required-symbols/>`
  lists all symbols that are considered to be understood and will not need any extra attention.

– `<required-CDs/>`
  lists all content dictionaries that are considered understood or not necessary to explain.

Deploy content.
Having finalized the contents and the graphs we continue with the necessary steps in order to deploy the document.

- *Copy the document into the MathDox Player*
  Until this point all the material for our document was created and located in separate directory. In order to see the results of the document we need to copy the document materials into the document folder of the MathDox Player.

  To do so, copy the document folder called `grouptheory` into the content folder of the MathDox Player. The `graphs/src` and `graph/build` folder should be omitted.

- *(Re)start Tomcat*
  We are now ready to start up the Tomcat server, and with that our document. Verify that the document is indeed active. Remember, if small changes are made to the graphs, a restart of Tomcat is necessary to clean the cache.

- *Create users* The MathDox Context System is user focused, that implies our document is only usable when we add users for it. Before user can be added, a file named `contextvariables.xml`. needs to be created. In this file all default values for new users are listed, see Listing 50. After the creation of this file, users can be added with a tool in the document. Go to `inspectiontools` and then to `create user`, see Figure 47. Fill out the user name and password and optionally the group the user belongs to. The group is important for authors and teachers, as these are given rights to add new users later on.

Listing 50: The contextvariables.xml

```
<properties>
    <entry key="root.goal">nogoal</entry>
    <entry key="root.notMathContextLink">false</entry>
    <entry key="root.background">empty</entry>
    <entry key="root.date">now</entry>
    <entry key="root.firstName"></entry>
    <entry key="root.firstVisit">true</entry>
    <entry key="root.lastName"></entry>
    <entry key="s1p1.a">&lt;OMI&gt;9&lt;/OMI&gt;</entry>
    <entry key="s1p1.b">&lt;OMI&gt;4&lt;/OMI&gt;</entry>
</properties>
```

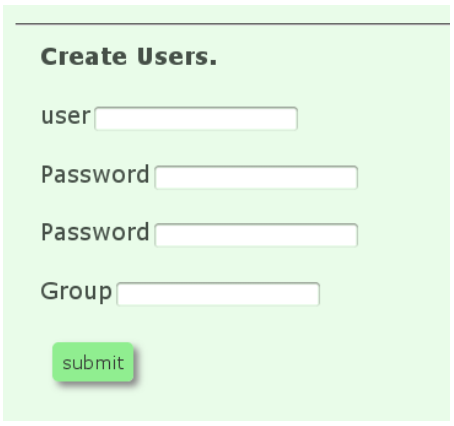


Figure 47: The create user tool

- *Fine tune and finalize*
  The document has been written and deployed, and is accessible through the MathDox Player. It does however need to be tested and verified if everything works as supposed. The author has some tools at his or her disposal to check for soundness and verify the relations defined in Section 4.1.4. These soundness checks are available in the document at `inspection tools`.

A.3 THE LAYOUT OF A MATHDOX CONTEXT DOCUMENT

In the previous Section A.1 and Section A.2 the creation process of a document has been discussed. In this section we will pay more attention to some technical details that were kept a minimum to previously.

Being; the files and layout in Section A.3.1, the document file A.3.2 and the database in section A.3.3.

A.3.1 *File structure*

To structure the files as needed for a document, a folder structure is advised located directly in the home folder of the document. This structure includes at least the folder *nodalpages*. Other folders that are recommended are *fragments*, *exercises*, *definitions* and *examples*. Each of these folders contain a separate folder for each nodalpage that exists in the document. In these sub-folders fragments — that are included in the nodalpages– are located. One should keep in mind that fragments can be included in different nodalpages. Convention in such cases dictate that such fragments belong to the nodalpage that is the highest in the theory graph.

Another recommended folder in the document home folder is the *graphs* folder. This folder is the place where the source files for the theory graph, symbol graph and variable graph are located. These are discussed in Section A.4

A.3.2 *Document file*

The document file is used to identify the current document within the MathDox context system. By placing the document file in the root of the document, all content belonging to the document is able to find and use the document identifier inside this document file.

This document identifier is then used by the contents to link all actions — as performed by the document — to the document in the database, allowing to distinguish between actions from different documents. Besides the document identifier the document file also contains links to the domain graphs such as the theory graph, symbol graph and variable graph, which form the domain model of the document together with the content and thereby define the document; in essence the values used in the document file give all the separate pieces a location in the bigger whole that is the document.

An author needs to specify the boundaries of a document. E.g. what is to be expected as entry knowledge from the readers. These boundaries are taken in the form of OpenMath symbols, as they are defined in a standard as opposed to theories from any theory graph. A trivial example would be the use of the addition or subtraction operators. Any mathematical document will assume that these operators are known to the reader. The *required-symbols* and *required-CDs* fields in the document file are to be used to tell the system which mathematical operators are expected to be known. Allowing for both single operators, or complete *cd*s.

A.3.3 *Database structure*

The MathDox Context system stores all important actions and decisions in an eXist database. The eXist database is an XML oriented database, meaning all data is stored as an XML document. Each user has his own folder in the database containing two documents. The `knowledge` document contains all actions in regard to knowledge gain of the user. The `variables` document contains all values of the variables and the `logs` contain all log events made by the document.

User verification in the MathDox context system is done by checking for the user in the eXist database, so each reader of a document needs to have an account in eXist. These credentials are used in the MathDox context system to allow readers access. This approach also protects the privacy of the user data in the database.

Listing 51: A knowledge XML document in eXist

```
<knowledge>
  <theorygraph>
    <documents>
      <document docid="IDA20100810">
        <action actionid="mastered">
          <theory
            name="permutation_group"
            date="20110131111521"
            resource="/experimental/orbits/nodalpages/example.md"
            graphURL="http://localhost:8080/mathdoxplayer/
                experimental/orbits/graphs/orbit-theorygraph.xml"
          />
        </action>
        <action actionid="read">
          <theory
            name="permutation_group"
            date="20110131111521"
            resource="/experimental/orbits/nodalpages/example.md"
            graphURL="http://localhost:8080/mathdoxplayer/
                experimental/orbits/graphs/orbit-theorygraph.xml"
            points="2"
          />
          <theory
            name="permutation_group"
            date="20110131125229"
            resource="/experimental/orbits/nodalpages/example.md"
            graphURL="http://localhost:8080/mathdoxplayer/
                experimental/orbits/graphs/orbit-theorygraph.xml"
            points="2"
          />
          </action>
        </document>
        <document docid="grouptheory">
        </document>
      </documents>
    </theorygraph>
    <symbolgraph>
      <documents>
        <document docid="IDA20100810">
          <action actionid="mastered">
            <symbol
              name="group.group"
              date="20110131125229"
              resource="/experimental/orbits/nodalpages/example.md
                  "
              graphURL="http://localhost:8080/mathdoxplayer/
                  experimental/orbits/graphs/orbit–theorygraph.xml
                  "
            />
          </action>
        </document>
      </documents>
    </symbolgraph>
  </knowledge>
```

Knowledge document.
In Listing 51 an example of a knowledge document is given. This example is kept short, as they may be considerably longer for an active user. The knowledge document stores both knowledge gained from the variable graph and the symbol graph. The `theorygraph` and `symbolgraph` contain an element called `documents` which in turn contains child elements for each specific document the reader has visited. These `document` elements need to contain the document identifier as specified in the document file. Note that the structure within the `theorygraph` and `symbolgraph` elements are similar. Within a `document` element come `action` elements. The `action` elements come in different flavors, there are `mastered` and `read` actions. But custom actions can also be created. An `action` element contains child elements and depending if the action is part of the theory or symbol graph these child elements are either called `theory` or `symbol`. The `theory` and `symbol` elements contain a set of attributes consisting of values of interest for this action. The standard set of attributes are:

- *name*, the name of the node from the theory or symbol graph.
- *graphURL*, the location of said theory or symbol graph.
- *date*, indicates when this action was added to the context and allows for time sensitive replay.
- *resource*, contains the resource file responsible for adding this action to the context.

This standard set can be expanded upon. A `read` action for example contains an attribute `points` that indicate the amount of reading points the system as awarded to the reader by just reading a page.

Variable document.
The variable document in Listing 52 contains the user values of the variables that occur in the document. All variables are grouped in `symboltable` elements. The name `symboltable` comes from the field of compiler construction [124] and should not be confused with symbols from the symbol graph. A `symboltable` is linked to a nodalpage and contains all variables that are introduced in that nodalpage. The exception is the symboltable with the name attribute set to `root`. This symboltable does not contain any variables from a nodalpage. Instead it contains all variables that occur in the logistic context. All symboltables belonging to one document are then grouped in the `document` element. Again this `document` element needs to contain the document identifier.

All variables contain the following attributes:

- `name`, contains the name of the variable. Note that this name is without the nodalpage prefix, as this prefix is already given in the `name` of the `symboltable` element.
- `date`, specifies the date that the value of the variable has been set. Usually the variable with the most recent date is used, unless a different date was specified
- `type`, indicates the type of the variable. This can either be `text` for traditional programming variables, `XML` for XML snippets, or `OpenMath` for mathematical variables expressed in OpenMath
- `vargraphvar`, is used to indicate if the variable occurs in the variable graph. If the variable does occur in the variable graph the system needs to verify if its value is still valid or not, and needs to know how to recalculate the value.
- `input`, only applies to variables from the variable graph and contains a boolean value to indicate if the variable is a so called *inputvariable* and may contain a direct value or if its value needs to be computed from other variables in the variable graph.

It is important to realize that all variables declared in the variable graph also need to be present in the *variable* document in the database.

Listing 52: An example of a variable XML document in eXist

```
<variables>
  <document docid="myfirstdoc">
    <symboltable name="root">
```

```
      <variable name="lastName" date="20100816140000" type="text"
                vargraphvar="false">Verrijzer</variable>
      <variable name="date" date="20101107200112" type="text"
                vargraphvar="false">now</variable>
      <variable name="lastName" type="text" date="20101224150338"
                vargraphvar="false">Verrijzer</variable>
      <variable name="readThresshold" date="20091106202153"
                type="text" vargraphvar="false">5</variable>
    </symboltable>

    <symboltable name="permutation">
      <variable name="g" input="true" date="20110411173000"
                type="openmath" vargraphvar="true">
        <OMI>9</OMI>
      </variable>

      <variable name="h" date="20110111173000" type="openmath"
                vargraphvar="true">
        <OMI>6</OMI>
      </variable>

      <variable name="h" type="openmath" date="20110422001843"
                vargraphvar="true">
        <OMI xmlns="http://www.openmath.org/OpenMath">18</OMI>
      </variable>
    </symboltable>
  </document>

  <document docid="idacontext">
    <symboltable name="root">
      <variable name="lastName" date="20100816140000" type="text"
                vargraphvar="false">Verrijzer</variable>
      <variable name="date" date="20101107200112" type="text"
                vargraphvar="false">now</variable>
      <variable name="readThresshold" date="20091106202153"
                type="text" vargraphvar="false">5</variable>
    </symboltable>
  </document>
</variables>
```

A.3.4 *Author & inspection tools*

Author and inspection tools encompass all that assist the author in the creation of a document. An example of these tools are the scripts used in the tutorial in Section A.2. Another example are the inspection tools that are used to verify the relations in the document and indicate whether a document is sound. It are these tools that show the author the Σ_V, Σ_S, Θ_V, Θ_S, Γ_S and Φ_V relations. Invaluable in creating a document and keeping track of the used variables and symbols on pages and variables. Even more useful for an author are the `theory graph inspection page`, the `symbol graph inspection page` and `variable graph inspection page` pages.

Theory inspection page.
The `TheoryGraph Status` page tells the author if everything related to the theory graph is considered sound. This mainly focuses on the nodalpages.

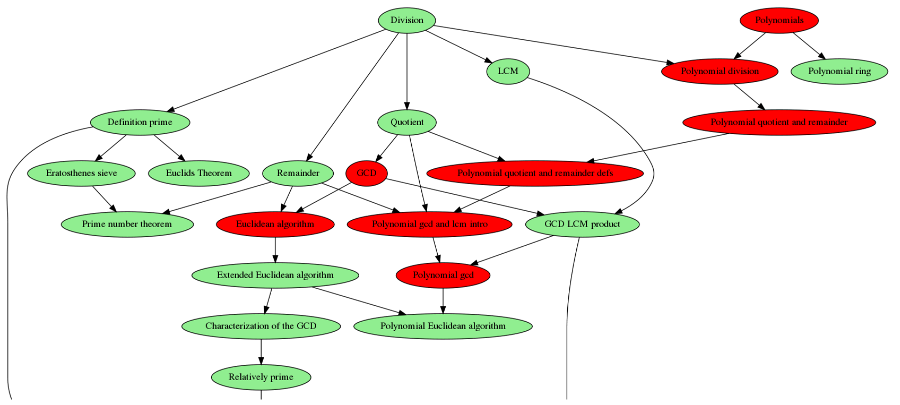


Figure 48: A theory graph with unsound nodes

In Figure 48 a theory graph is shown in which the nodes are colored either green, if the nodalpage associated to that node is sound, or red when the nodalpage is not sound. Any node that is considered not sound must be checked by the author. To assist in this task each node from the theory graph has a separate status table on the *TheoryGraph Status* page. This status table contains the following checks:

- *All resources used by the nodal page*
  A list of all content files as used by this nodalpage. This list includes all normal MathDox pages and fragments. All conditional fragment inclusions are also included in this list.

- *All variables available for use by this nodal page (Δ_V)*
  The map $\Delta_V(\mathsf{theory})$ includes all variables that are available to the current nodalpage *theory*, it will also include the value of the variable and where that variable is introduced.

- All symbols available for this nodalpage (Δ_S*)*
  Similar to $\Delta_V(\mathsf{theory})$. The map $\Delta_S(\mathsf{theory})$ includes the symbols available to the current nodalpage theory and specifies where the symbol was introduced. If the symbol was specified as required knowledge in the document file, it will show as *prior-knowledge*.

- Variables used on the nodal page (Σ_V*)*
  $\Sigma_V(\mathsf{theory})$ is a map that includes all used variables on the nodalpage associated with the theory. As added information a check is done to verify $\Sigma_V(\mathsf{theory}) \subseteq \Delta_V(\mathsf{theory})$ holds.

- Symbols used on the nodal page (Σ_S)
  Similar to Σ_V. It is a map that lists all used symbols on the current nodalpage. The check $\Sigma_S(\mathtt{theory}) \subseteq \Delta_S(\mathtt{theory})$ is also performed.

- The symbols occurring in the variables on this nodal page ($\Phi(\Sigma_V(\mathtt{theory}))$). Is a map that is gotten by using the results of $\Sigma_v(\mathtt{theory})$ as input for Φ. The resulting list are all symbols occurring in the variables that are used on the current nodalpage. This list again should be a subset of $\Delta_S(\mathtt{theory})$ which is checked.

- *Theory properties*
  Lists all properties associated to the theory node in the theory graph.

If any check fails it will be indicated with a red bar. Also the status table header itself will turn red in such a case. A failed check is only an indication something might be wrong and needs author attention.

Variable inspection page.
The `variable inspection page` has the same function as the `theory inspection page`, but as the name suggests it focuses on the the variables instead. The `variable inspection page` starts with a variable graph, which, unlike the theory inspection page, is not colored. Coloring could be done based on variable conditions, however these apply on the values of a user, while this page operates on document level.

For each variable there is a status table that shows all related properties for each variable. These status table includes the following:

- Variable is used in the following nodal pages
  It lists all nodal pages that this particular variable is used on.

- *Introducing theory is*
  Will tell the author in which nodalpage the variable is introduced, first used.

- *Variable uses the following symbols*
  Lists all symbols occurring in the variable. If the variable is an input variable it will not contain any symbols.

- *Variable properties are the following*
  Lists all properties associated with the variable in the variable graph. These typically include (but are not limited to)
  - *cas*, indicates which computer algebra system should be used for the computation of the current variable. This allows for having some operations done on a different — more specialized — computer algebra system.

  - *inputNode*, is an attribute that is set when the variables value is not computed but given by the reader because it has no dependencies.
  - *OMExpression*
    Contains the OpenMath expression needed to calculate the value of this variable. Any variable containing this attribute is not an input variable.
  - *description*
    Gives a short human readable description of the variable.
  - *inputnode*
    Indicates if the current variable is considered an input node in the variable graph. Meaning, whether the user is allowed to adapt its value.

- *Conditions*
  Lists all conditions that apply to this variable. It will display the name of the condition, the mathematical expression of the condition and the result of the condition. Note however, that the current value is for the current user and might be different for other users.

A.4 DOMAIN MODEL GRAPHS

The theory, symbol and variable graph all make use of the GraphML format and are therefore all similarly constructed. For this reason we suffice with explaining the construction of a theory graph file as this is the main graph of the MathDox Context system. It helps to guide the user through the document and also provides the document and user a partial ordering to the various subtopics that are being addressed in the document. Later in this section (see Section A.4.4) some attention is paid to specific differences in the variable graph as compared to the theory and symbol graph.

The main summarized form of a domain model graph is given in Figure 49. This summarization only displays the key tags as needed in a GraphML file, namely the *graphml* and *graph* tags. Also, all required namespaces as they may be needed in a domain graph file have been omitted for the sake of readability. One exception has been made for the GraphML namespace itself. Domain graphs are directed, which is indicated at the attribute *edgedefault="directed"*. We will discuss each section denoted in comments in Figure 49 in the following sections, starting with properties for nodes and edges in Section A.4.1.

A.4.1 *Properties for nodes or edges*

The layout of the different components of a theory graph file is discussed one by one, starting with the head, right after the *graphml*

```
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <!-- properties -->
  <graph edgedefault="directed">
    <!-- nodes -->
    <!-- edges -->
  </graph>
</graphml>
```

Figure 49: The main form of a domain model graph

opening tag are the declarations of the properties. In Figure 53 an example of such property declarations can be found. These declarations define what properties can be used in the nodes and edges defined later in the GraphML file. The syntax is straightforward: an opening *key* tag with included attributes stating the identifier *id* of this particular property, a name for the property given in attr.type, a definition whether it is a property for a node or edge (either *for='node'* or *for='edge'*) and finally the type of the property value can be set by supplying the attribute *attr.type*. These typings correspond to XMLSchema. Inside the key tags there is an optional *default* tag. The value defined in this *default* tag is used as the default value when the property is not actively given at the node or edge it belongs to.

Listing 53: The start of a theory graph file

```
<key id="links" for="node" attr.name="links" attr.type="string"/>

<key id="url" for="node" attr.name="url" attr.type="string">
    <default>false</default>
</key>

<key id="oms" for="node" attr.name="oms" attr.type="string" />
```

A.4.2 *Defining nodes*

The next part of the graph we are going to discuss is the definition of nodes. In Listing 54 we are going to address how a node in a theory graph is defined. The contents of Listing 54 show a few nodes of a theory graph that immediately connects to the previous part of the theory graph as shown in Listing 53 and is a possible implementation of the `<!-- nodes -->` comment in Listing 49.

Listing 54: Definition of graph nodes

```
<node id="subgroup">
  <data key="links">
    <links>
      <link category="cs">
```

```
        <description>test url</description>
        <url>https://en.wikipedia.org/wiki/Alan_Turing</url>
        <name>Alan Turing</name>
      </link>
    </links>
  </data>
</node>

<node id="permutation_group">
  <data key="oms">
    <symbols>
      <OMS cd="permutation1" name="cycle"/>
      <OMS cd="permutation1" name="permutation"/>
      <OMS cd="permgp2" name="symmetric_group"/>
      <OMS cd="permgp2" name="cyclic_group"/>
      <OMS cd="groupname1" name="dihedral_group"/>
      <OMS cd="fns4" name="maps_to"/>
      <OMS cd="fns4" name="assign_to"/>
      <OMS cd="fns1" name="lambda"/>
      <OMS cd="fns1" name="inverse"/>
      <OMS cd="fns3" name="function"/>
      <OMS cd="fns3" name="restriction"/>
    </symbols>
  </data>
</node>
```

The two `node` tags shown in Listing 54 defines nodes in a graph, in our case a theory graph. It is required to uniquely name each node by the use of the `id` attribute. As discussed in Section A.4.1 optional properties can be attached to a node. When this is desired the `key` tag is to contain a child `data` tag with a `key` attribute. The value specified for the `key` attribute needs to refer to a previously declared property. Multiple properties can be added to one node, however, each property added requires a separate `data` tag.

Theory graph properties.
The properties as attached to the nodes in Listing 54 are predefined properties in the MathDox Context system. Therefore we briefly explain what each of these properties is used for.

- The `links` property defines potential useful links based on the background of the reader. The syntax is as that of an XML document. It starts with a `links` root element in which one or more `link` elements can be placed. Each `link` element in its turn consists of a `description` and `url` element, containing a description and the URL of the link. Since this embedded XML document will be parsed as a separate XML document later on by the MathDox system, it is important to make sure the document is valid on its own. Meaning all used namespaces need to be (also) declared inside this XML document. In this example the node `subgroup` can present to its readers from the category `cs` a link to useful information outside the document. Needless to say, if the reader has a different background than computer science (cs) it will not show this link. Also, the functionality to show such a link is left to the code in the nodal page or in one of its fragments. This code is of course free to overrule any indications given here.

- The `oms` property is specified within the `data` tag element. The `oms` property specifies which OpenMath symbols are introduced. This in is fact a tool for the system to check if the nodal pages are sound. Due to the many different conditional included fragments it is hard for an author to keep track of all the symbols used. By specifying these symbols the author is telling the system that they will be introduced in this nodal page. By combining all symbols from this page and all predecessor pages a set can be constructed that indicates all introduced symbols so far. If by chance a piece of content (nodal page or a fragment included by the nodal page) uses a symbol that is not in this set, then the page is unsound. A situation an author wants to address and possibly repair. As such this is a mechanism aimed at the author and it assists the author in keeping the pages correct.

A.4.3 *Edges in a graph*

In Listing 55 an example is given on how to connect nodes from the theory graph with edges. The contents of Listing 55 connect to the previous Listings 53 and 54 and is a possible implementation of the `<!-- edges -->` comment in Listing 49. This section and example only applies to the theory graph and symbol graph and not to the variable graph. The variable graph gets edges created in a preparation script, more about this in Section A.4.5. As one can see in Listing 55, edges are defined by an empty element `edge`. This tag does however require three attributes; `id` which is required by the GraphML format to uniquely define each edge, `source` to indicate the source of the edge and `target` to likewise indicate the target of the edge. If so required properties can be added to edges in the same way as was done in Section A.4.2 for nodes.

Listing 55: Definition of theory graph edge

```
<edge
  id="subgroup-permutation_group"
  source="subgroup"
  target="permutation_group"
/>

<edge id="permutation_group-orbit"
  source="permutation_group"
  target="orbit"
/>

<edge id="permutation_group-stabilizer"
  source="permutation_group"
  target="stabilizer"
/>
```

A.4.4 *Variable graph properties*

The basic variable graph syntax is similar to that of the theory and symbol graph, the graph syntax therefore needs no extra explanation as the definition of the properties and nodes are identical. However the definition of the edges differ, as they are created by the graph preparation script, see Section A.4.5. All edge information is stored inside the `OMExpression` property attached to each node. It is this information that is used to create the necessary edge definitions in an automatic manner. The following properties only apply to nodes of the variable graph;

- OMExpression
  As the nodes in the variable graph represent variables which are in most cases constructed from input from other variables in the variable graph, an OpenMath expression is required to define the formula responsible for determining the value of the current node. As before this requires a `data` tag element. The required attribute is `OMExpression`. The XML entered into this `data` element is required to be valid as a separate XML document, hence the namespace declaration. In Listing 56 an example of an `OMExpression` property is given. Note that when another variable is used this variable is referenced with the tag `variable`. The `name` attribute refers to the symboltable and variable separated by a dot.

Listing 56: An example of the OMExpression property

```
<node id="symmetric_group.S">
  <data key="cas">gap</data>
  <data key="OMExpression"
        xmlns:mdcv="http://mathdox.org/mathdoxcontext/
            variables"
  >
    <OMA>
      <OMS name="symmetric_group" cd="permgp2"/>
      <mdcv:variable name="symmetric_group.n" />
    </OMA>
  </data>
</node>
```

- inputNode
  There are variables that are not depending on other variables, they are at the root of the graph. These variables can be marked as input variables by the `inputNode` attribute. Doing so will allow readers to change their values while reading through the document.

- cas
  The `cas` property is used as means to indicate which computer algebra system needs to be used to calculate the `OMExpression`. If this attribute is not specified, then the default is used as specified in the MathDox properties file, refer to the MathDox manual for more information on the MathDox properties file. An example can be seen in Listing 56

- conditions
  The soundness of the variable graph, and its values as they will be used in the document need to be watched over. For this purpose conditions can be supplied. Conditions can be given in two different flavors. Cas conditions are OpenMath expressions which may refer to variable values (current variable included) that need to pass when they are calculated by a computer algebra system. For this purpose the OpenMath expression needs to return a boolean value.

  A different kind of condition is given by means of an XPath expression. An XPath expression succeeds when the result is not empty or not *false*. An example of conditions is given in Listing 57. Again the XML inside the `omexpr` element needs to be a valid XML document as it will be parsed as such by the system.

Listing 57: A conditions example

```
<conditions>
  <condition type="xpath">
    <xpath>not(//OMS[@cd="arith"])</xpath>
    <message>
      no symbols from arith1 dictionary are allowed
    </message>
  </condition>
  <condition type='cas' cas='gap'>
    <omexpr>
      <OMA>
        <OMS cd="permgp1" name="is_in"/>
        <mdcv:variable/>
        <mdcv:variable name="symmetric_group.S"/>
      </OMA>
    </omexpr>
    <message>
```

```
      permutation.G needs to be an element of
          symmetric_group.S!
    </message>
  </condition>
  <condition type='cas' cas='gap'>
    <omexpr>
      <OMA>
        <OMS name="neq" cd="relation1"/>
        <mdcv:variable/>
        <mdcv:variable name="permutation.h"/>
      </OMA>
    </omexpr>
    <message>
      permutation.g needs to be different than permutation.h
          !
    </message>
  </condition>
</conditions>
```

In Listing 57 it is demonstrated that one can create as many conditions as required. Each condition requires a *type* that specifies if the condition is either a `cas` or `xpath` condition. If the condition is a cas condition an extra optional attribute named `cas` may also be provided. This extra attribute specifies which computer algebra system is to be used for the evaluation. Both types of conditions require a message which contains a description message to be shown to the author when the author checks for soundness.

A.4.5 *Graph preparation script*

The graphs of the domain model; theory, symbol and variable graph, all are XML documents that need some further preparation before they are suitable for the MathDox Context system. Among these tasks is the conversion of the XML documents into SVG files that can be colored with some Jelly statements. This particular step exports the GraphML format towards the dot format, which in turn is converted into a SVG file with a proper layout by the program GraphViz. The conversion is then concluded by the addition of the proper Jelly statements that provide the logic for the coloring of the nodes. This last step is performed for each set of logic required.

The conversion script is implemented by an ant script. The script is run with the command `ant buildall`.

A.5 QUERIES & RULES

The MathDox Context system comes with a set of predefined queries and rules implemented by custom Jelly tags or MathDox fragments.

This set is by no means fixed or final. New queries or rules can be added or existing can be expanded upon. First we present a summarization of the current available custom tags, their uses and properties.

A.5.1 *Custom tags*

The predefined custom Jelly queries and rules all are packaged into the same tag library jar. They therefore all share the same namespace, usually abbreviated to `mdc` (MathDox Context),
`xmlns:mdc="jelly:org.mathdox.context.TheoryBlocksTagLibrary"`
The predefined custom tags are divided into separate subgroups of functionality. Each sub-group is discussed in the next sections.

A.5.1.1 *Graph queries*

Graph queries are Jelly custom tags that retrieve data from a theory, symbol, or variable graph. They do not change the context nor do they change the content, they merely retrieve data that are to be used as input for rules that do take action upon the provided data. For this reason these tags usual have an attribute that lets the author specify and variable name. It is this variable name that is used to create a variable with the result of the tag. Omitting the variable name lets the results be used as text in the constructed page.

neighbour-theories.

The tag `neighbour-theories` returns a set with all neighboring nodes of the current graph. This includes all predecessors and all successors. Attributes are:

- `name`
  The name or identifier of the node we are looking for
- `type`
  The type of graph in which the node is to be found. Accepted values are `theory`, `symbol` and `variable`

available nodes.
`available-theories` returns a set with all theories or symbols that have not yet been mastered while all their requirements have been met. In other words, the reader is ready and able to start reading the associated nodal pages. Attributes are:

- type
  Accepts either theorygraph or symbolgraph and affects the kind of graph this tag will work on.

- var
  Specifies the variable in which the results need to be stored.

masterednodes.
The tag `earned-theories` returns a set with all mastered theories or symbols from the theory or symbol graph. Attributes are:

- `type`
  The type of graph in which the nodes are to be found. Accepted values are `theory` and `symbol`

notReachableNodes.
The tag `notReachableTheories` returns a set with all nodes from the theory or symbol graph that can not yet be understood by the reader because he or she lacks the required knowledge. In other words, one or more of the predecessors have not yet been mastered Attributes are:

- `type`
  The type of graph in which the node is to be found. Accepted values are `theory`, `symbol` and `variable`

required-theories.
The tag `required-theories` returns a set with all predecessor nodes that have not yet been mastered. This tag works on both the theory and symbol graphs. Attributes are:

- `name`
  The name of the node from the theory or symbol graph that identifies the node the required knowledge is wanted from.

- `type`
  The type of graph in which the node is to be found. Accepted values are `theory` and `symbol`

graph-property.
The tag `graph-property` returns the value of the named property of a node. The node can be a node from the theory, symbol or variable graph. Attributes are:

- `name`
  The name of the node we are looking for

- `type`
  The type of graph in which the node is to be found.

- `property`
  The name of the property Accepted values are `theory`, `symbol` and `variable`

get-nodes.
The tag `get-nodes` returns all nodes from the named graph. Attributes are:

- `node`
  The name or identifier of the node we are looking for
- `type`
  The type of graph in which the node is to be found. Accepted values are `theory`, `symbol` and `variable`

get-variablenames.
The tag `get-variablenames` returns all variable names in a given symboltable. Attributes are:

- `jellyvariable`
  The variable name in which the result is to be stored.
- `symboltableid`
  The name of the symbol table of interest

A.5.1.2 *Knowledge tags*

The next group of custom tags are tags that aim at the knowledge logistic. Hence this group of tags all have something to do with either checking the knowledge level of the user, or setting or changing the knowledge of the user in the context.

add-knowledge.
The tag `add-knowledge` tag records that the user has gained knowledge from either the theory or symbol graph.
Attributes that go with this tag are:

- `name`
  The concept that is to be recorded as being mastered by the user.
- `type`
  Indicates whether the concept graph is a theory or a symbol graph.
- `action`
  Allows for different types of learning points. For example the author can make a difference in `read` or `understood`. If this attribute is left blank it will get assigned the value `read`.
- `points`
  An integer indicating the points the user is to be awarded or penalized if the integer is negative. If left blank zero points will be added.

if-knowledge-available.
The tag `if-knowledge-available` tag allows conditional behavior based on the knowledge of the user. The body of this tag is only executed if the knowledge of the user passes a certain threshold in points or the knowledge has been registered as being mastered. Attributes are:

- `name`
  The concept that is to be recorded as being mastered by the user.
- `type`
  Indicates whether the concept graph is a theory or a symbol graph.
- `action`
  Allows for different types of learning points. For example the author can make a difference in `read` or `understood`
- `aggregate`
  Tells the tag what to do with the entries for the given action. `count` counts the number of entries, `pos` sums all positive points, `neg` sums all negative points, `all` sums all points positive and negative.
- `count`
  returns the number of entries in the context.
- `var`
  If a value is given then it stores the result in the variable with the given name. Otherwise the result is returned to the MathDox page.

knowledge-count.
This tag will gathers all events that belong to a certain action and to a node in either the theory or the symbol graph. The result is either the total events found, or the points that have been added to each event summed.

- `type`
  Specifies whether the theory or the symbol graph is to be used.
- `action`
  The action that needs to summed.
- `name`
  Which node the tag will work on.
- `aggregate`
  Whether to count the occurrences in the database, or to add/-subtract all assigned points

- `var`
  The variable to store the result in.

A.5.1.3 *Fragments*

include-fragment.
The tag `include-fragment` includes the contents of a fragment onto the page that is being created for the user. The fragment uses already set variables from the code that includes the fragment, but also accepts parameters. It does so a number of different ways; variables can be put as name value pairs into a map or variables can be set in the body of the fragment. In both cases the variables will be created in a separate Jelly context meaning, the fragment will see and use them as normal variables, but the code including the fragment will not be able to access them. When it is desired that the code inside the fragment passes a result value back to the including code outside the fragment, then a variable name can be passed along that is used to store any returning values as name value pairs and returning these to the code outside of the fragment.

Attributes are:

- `fragment`
  The location of the fragment that is to be included. Local addresses are allowed.

- `frgid`
  An identifier used to tell fragments apart. Nested fragments append their identifiers to that of their parents, creating a package like structure that is used to link post values from XForms to the correct fragment.

- `args`
  May contain a variable name of a map with name-value pairs. These name-value pairs are converted to variables in the Jelly context of the fragment.

- `var`
  Stores any variables that need to be returned after execution of the fragment towards the calling code. The attribute value is used as the name of the map that contains name-value pairs representing the returned values. A fragment does need to create this map and its content before it is possible to return values.

fragment.
The tag `fragment` only invokes its body and sets trim to false. The tag has no other functionality but is required as a root element for the fragment MathDox code and serves as recognition point for the MathDox Player to recognize a fragment as a fragment.

Attributes are:

- No attributes.

A.5.1.4 *Variables*

Variables as used in the variable graph need means to read or write associated values.

set-value.
The `set-value` tag is used to assign values to variables in the logistic or mathematical context. As such this tag is closely related to the `get-value` tag which retrieves these values for usage in the MathDox document.

- `variable`
  Indicates which variable is to be used.
- `symboltableid`
  The identifier of the symboltable the variable belongs to.
- `value`
  Leaving this attribute blank means the value is specified as the body of this element.

get-value.
The `get-value` tag retrieves a value from a variable. It does not matter if this variable is used for the mathematical or logistic context. Basic variable types can be `text`, `OpenMath` or `XML`.

- `variable`
  Indicates which variable is to be used.
- `symboltableid`
  The identifier of the symboltable the variable belongs to.
- `jellyvariable`
  The value of the variable is stored into the provided Jelly variable. Allowing adjustments to be made locally and without the need to store these.
- `date`
  This field works with a number indicating the date and time. When a date is specified the value retrieved is the value that was present on the specified date, regardless of any changes made to the variable afterwards. In other words the value that has been set before the given date with the smallest possible time difference is used. If no date is specified the current date is used

- `evaluate`
  This attribute takes a boolean value. If the boolean value is set to true it will recalculate the value of the variable according to its OpenMath expression and values of variables it depends on. This may result in recalculation of these depending variables as well. Recalculation is performed automatically when a predecessor contains a newer value that the current variable value. Therefore default value for this attribute is false.

get-coloring.
When a graph is displayed on a MathDox page it is helpful to have its nodes colored as to help the reader to understand his current context.

- `soundness`
  A true value will tell this tag to check if the nodes are sound and adapt the coloring of these nodes accordingly. Soundness can only be checked on the theory graph, where it actually applies to the nodal pages associated with the theory graph or with the variable graph where it verifies if the values held by the variables from the variable graph all meet the specified conditions. The default value is false

- `pointsRequired`
  The MathDox Context system can judge a node from the theory graph as being understood based upon the times it was visited by the user. Nodes from the theory graph that have points associated to them can be colored with a different color when they have more points that the threshold specified here. If no threshold is used the predefined threshold in the logistic context is used.

- `notOwnedTheoriesColor`
  The setting for this attribute will determine the color for the nodes that failed the conditions or have not been mastered yet. The color is given as a string and can be anything that is recognized by the SVG format. The default color is red.

- `ownedTheoriesColor`
  The setting for this attribute will determine the color for the nodes that passed the conditions or have been mastered. The default color is green

- `availableTheoriesColor`
  All neutral nodes will be colored with color specified in this attribute. The default color is blue.

A.5.1.5 *Relations*

The relations as described in Section 4.1.4 have been implemented and are shown on the authoring pages. In this section we will discuss the implementations and their interfaces.

delta-var.
The `delta-var` tag lists all variables the specified nodal page is able to access. This includes all variables linked to earlier nodal pages. A earlier nodal page is a nodal page that is linked to a higher node in the theory graph than the current specified node from the theory graph.

- `theory`
  The theory node from the theory graph that is used as a starting point. All nodes higher in the graph than this theory node will be examined for linked variables.

- `suppress`
  Normally results are written in a table. However a list is also possible when this attribute is given a true value. The default value is false

- `toTop`
  When the scope of interest is limited to variables that are linked to the current nodal page a false value can be specified. The default value is true, meaning that by default all variables linked to higher nodal pages are included.

delta-sym.
The `delta-sym` is very similar to the `delta-var`, instead of returning variables it returns the symbols introduced at the given nodal page or at one of its predecessors.

- `theory`
  The theory node from the theory graph that is used as starting point. All nodes higher in the graph than this theory node will be examined for symbols.

- `suppress`
  Normally results are written in a table. However a list is also possible when this attribute is given a true value. The default value is false

- *toTop*
  When the scope of interest is limited to symbols on the current nodal page a false value can be specified. The default value is true, meaning that by default all symbols used on higher nodal pages are included.

used-resources.
The *used-resources* tag makes an inventory of all content files used. Especially with the use of fragments which in turn can use other fragments, it is hard to keep track of the contents of a page or document. Such an inventory is used to determine which symbols or variables are introduced on what page. It is therefore an important aspect of the soundness constraints that apply to a document. The only attribute that is used:

- *theory*
  The theory node that indicates which nodal page should be inspected for used resources.

occurrence-var.
Lists all variables that occur on the nodal page, given by the theory node. The difference between *delta-var* –with the *totop* setting to false– and this tag is that the *delta-var* tag lists all variables in the variable graph linked to the nodal page, whereas this tag only lists those variables actually used on this nodal page. This tag only takes one attribute namely:

- `theory`
  The theory node indicating which nodal page is to be inspected for actual occurrences of variables.

occurrence-sym.
The `occurence-sym` tag is very similar to the `occurence-var` tag. It too looks for occurrences on the nodal page, but this tag looks for symbols instead of variables. Furthermore, `occurence-sym` is almost identical to `sigma-var`, with the difference that the occurrence tags are used for internal use, while the sigma tags presents the results. This tag takes one attribute:

- `theory`
  The theory node indicates which nodal page is to be inspected for actual occurrences of variables.

sigma-var.
This tag has the same functionality as the `occurence-var` tag. Both list the variables that occur on a nodal page. However this tag also takes care of presentation and shows each result as a link to the appropriate section within the document.

- `theory`
  The theory node indicates which nodal page is to be inspected for actual occurrences of variables.

sigma-sym.
This tag has the same functionality as the `occurence-sym` tag. Both list the symbols that occur on a nodal page. However this tag also takes care of presentation and shows each result as a link to the appropriate section within the document.

- `theory`
  The theory node indicating which nodal page is to be reviewed.

sigma-var-complete.
The `sigma-var-complete` tag does the same as the `sigma-var` tag, but instead of taking just one nodal page, it considers the complete document. For this reason it does not accept any attributes.

sigma-sym-complete.
This tag is similar to the `sigma-sym` tag, but performs the task on all theories instead.

theta-var.
Indicates on which nodal pages the current variable occurs. The nodal pages are presented as links.

- `variable`
  The variable from the variable graph that occurs on nodal pages.

theta-var-complete.
The same as the `theta-var` tag, but now applies to variables from the variable graph. It does not accept any attributes.

theta-sym.
Similar to the `theta-var` tag but now for the symbols. This tag searches which nodal pages contain this symbol.

- `symbol`
  The symbol we search on the nodal pages of the document.

theta-sym-complete.
The same as the `theta-sym` tag, but now for all symbols occurring anywhere in the document. This tag does not accept any attributes.

small-theta-var-complete.
This tag returns an oversight of all variables and where they were used for the first time in the theorygraph associated nodal pages. It does not look at the symboltable identifiers to do so, but scans the document to determine this. This tag does not accept any attributes.

small-theta-var.
Similar to `small-theta-var-complete`. However, it now accepts a variable name and determines where the variable was first used in the theory graph associated nodal pages.

- `variable`
  The variable to determine its first use.

small-theta-sym-complete.
This tag returns an oversight of all symbols and where they have been introduced in the nodal pages. Any symbols that are specified as prior knowledge in the document file, will be labeled 'prior knowledge'. This tag has no attributes.

small-theta-sym.
Given a symbol this tag returns the introduction nodal page or prior-knowledge.

- `symbol`
  The symbol for which to determine the introduction page.

phi-complete.
Returns a presentation of symbols as used in the variables. The symbols are given as links to the symbol pages in the document. This tag does not accept any attributes.

phi-var.
Given a variable the tags returns the used symbols in the variable.

- `variable`
  The variable to be examined.

gamma-complete.
This tag returns for each symbol used in the document a list of variables that make use of the symbol. This tag has no attributes.

gamma-sym.
Given a specific symbol, this tag returns in which variables the symbol is used.

- `symbol`
  The symbol that is sought for in variables.

gamma-sym-list.
Unlike the previous gamma relations that are supposed to be used in oversights, this tag returns a list with variables that uses the given symbol. This means that this tag is meant as a query unlike the other tags which are meant to be used as rules.

- `symbol`
  The symbol that is sought for in variables.

sigma-subset-delta-var.
The `delta-var` tag produces a list of all variables the nodal page should have access to. The `sigma-var` is the list the nodal page actually accesses. In a sound document the second list is always a sub-set of the first. This tag returns whether that is really true.

- `theory`
  The nodal page that is to be examined.

sigma-subset-delta-sym.
The `delta-sym` returns a list of symbols that are available to be used by the nodal page. The `sigma-sym` gives the list of symbols that are used by the nodal page. The latter needs to be a subset of the first. This tag returns whether this is the case.

- `theory`
  The nodal page that is to be examined.

A.5.1.6 *Conditions*

Conditions apply to the variables of the variable graph, mostly to the input variables. However, they may also apply to other non input variables. The conditions are used to make sure the user does not enter values that lead to unwanted or illegal states later on.

test-conditions.
This tag checks all conditions — CAS conditions and XPath conditions — available for the given variable and determines whether they hold or not. The conditions and the results are presented in a table.

- `variable`
  The variable which conditions are to be checked.

A.5.1.7 *Mathematics adaptation rules*

definition & definition-anchor.
The `definition` tag works together with the `definition-anchor` tag. The `definition` tag inserts a link in the text connected to a stretchtext elsewhere on the page, usually just below the paragraph. The `definition-anchor` marks the location where the stretchtext will be.

The attributes for the `definition` tag are;

- `url`
  Contains the location URL for the content in the stretchtext. Either the `url` or the `name` has to be specified. If both are specified, the location URL takes precedence.

- `name`
  The `name` should be given if there is no location `url` or the document should determine the location itself. If there is no location URL, the name is used to determine the location of the right definition fragment.

- `graphType`
  The `name` attribute only has meaning if the document knows which graph to use.

- `title`
  Contains the name used in the link in the text.

- `on`
  With the `on` attribute an existing `definition` `definition-anchor` pair can be turned off. Useful for example in a global setting.

- `stretch`
  Indicates if the content should be a stretchtext or a separate tab in the browser instead.

- `var`
  The `definition` tag creates the content and puts it into a variable, specified here, so that the `definition-anchor` can pick it up and show the contents.

The attributes for the `definition-anchor` tag are;

- `var`
  The variable that contains the content that should be shown in the stretchtext.

link.
If there are links available for the users background in the theory graph, this tag will show them on the page.

- `category`
  The category to look for in the theory graph under the current node. If no category is given all links found at the current node in the theory graph are shown.

recursive-Sigma-Phi.
This tag returns a table with symbols that occurs in the variables that are used in a given theory.

- `theory`
  The nodal page to be examined.

recursive-Sigma-Phi-is-Subset-DeltaSym.
The results as returned by `recursive-Sigma-Phi` should not contain any symbols that were not introduced in an previous or current node in the theory graph. This tag verifies that is indeed not the case.

- `theory`
  The nodal page to be examined.

A.5.1.8 *Soundness*

theory-sound.
Verifies if all soundness checks hold for the given theory.

- `theory`
  The nodal page to be examined.

A.5.2 *Creating new queries and rules*

When the standard set of queries and rules do not suffice, new queries or rules can be added to implement new behavior. Extending behavior is done by writing MathDox code (see the MathDox manual [61]), creating a fragment or creating a custom tag.

Creating fragments queries and rules.

Listing 58: A fragment call with arguments set in the body

```
<mdc:include-fragment fragment="fragments/rules/euclides/
    euclidesstep.mdf" frgid="step" var="result">
  <c:set var="a" value="${a}"/>
  <c:set var="x" value="${x}"/>
  <c:set var="y" value="${y}"/>
  <c:set var="b" value="${b}"/>
  <c:set var="u" value="${u}"/>
  <c:set var="v" value="${v}"/>
  <c:set var="step" value="${step+1}"/>
</mdc:include-fragment>
```

Listing 59: A fragment call with arguments in a name value map

```
<c:new var="parameters" className="java.util.HashMap"/>
<c:set target="${parameters}" property="question"
      value="${question}"/>
<c:set target="${parameters}" property="answer"
```

```
      value="${answer}"/>

<mdc:include-fragment fragment="exercises/mcfragment.mdf"
    frgid="exercise" args="${parameters}"/>
```

A fragment offers functionality to another fragment or a MathDox page. Included fragments can have their own set of arguments that is used inside the fragment and affects the functionality of the fragment. In Listing 58 a fragment is being called. Note the definition of the variables in the `include-fragment` tag. These variables are only valid inside the fragment and are not accessible outside the `include-fragment` tag, making them effectively local variables for the fragment and preventing collisions with variables used by the calling code or other fragments. A different way of passing arguments to a fragment is shown in Listing 59. Here a list of name value pairs is passed along as an attribute value. Upon receiving this list in the fragment the contents are converted to variables, again only valid inside the fragment. A third option is possible by having a fragment access the variables from the calling code directly. These variables are however not just limited to the fragment and may also be used in other places. As such the first two approaches are preferred and the author is required to be more careful when the last approach is used.

Fragments may also return (intermediate) results to the calling code. This is especially useful if the called fragment performed a subtask of a larger yet unfinished task. The intermediate results of this subtask may be required to complete the larger task. In Listing 58 there is also an example of returning results. Note that the author is free to choose the variable that will hold the results. Again this is important to avoid collisions with other variables already in use. The fragment itself makes use of a reserved variable name `_return` as the MathDox code in the fragment is unaware of the variable given in the call. Instead the `_return` value is relabeled with the given `var` name by the implementation of the `include-fragment` tag.

Creating new tags.
With new custom tags new behavior is implemented. Jelly custom tags are Java classes programmed to perform a task and return the results either as XML into the MathDox document, as a value of a variable, insert into a database, or not return anything. Tags written in Java can perform a lot of different and complex tasks.

Listing 60: An example of a custom tag.

```
package org.mathdox.context.util;

import org.apache.commons.jelly.TagSupport;
import org.apache.commons.jelly.XMLOutput;

/**
```

```
 * A tag to specify the current version
 * of the MathDox Context tag library
 */
public class StatusTag extends TagSupport
{
  public void doTag(XMLOutput output) {
    try {
      output.write("MathDox Context installed,"+
                   "version 1.0,"+
                   "dated October 13th 2015");
      output.flush();
    } catch (Exception exception) {
      exception.printStackTrace();
    }
  }
}
```

In Listing 60 an example of a custom Jelly tag is given. This is a very simple tag since its only function is to return the current version into the page that is being served by the MathDox Player. Consider the code and note that this tag is implemented in a class that extends from the `org.apache.commons.jelly.TagSupport` class. Extending this class is required as it provides an important part of the interface for a Jelly tag and allows any Jelly tag to be seen as an object from this type. Something else that needs attention is the `doTag` method. It takes as argument and `XMLOutput` object. This object is used to write any XML towards the MathDox page being processed. In this example there are no XML elements, just text, being written to the output. If however XML elements do need to be written the XMLOutput object has `startElement(String)` and `endElement(String)` methods to do just that.

Listing 61: The MathDox Context tag library class.

```
public class TheoryBlocksTagLibrary extends TagLibrary {
  /**
   * Tags are registered in the constructor, the
   * constructor is called from jelly after
   * inclusion in a mathdox document.
   */
  public TheoryBlocksTagLibrary() {
    // returns all variable names in a given symboltable.
    registerTag("get-variablenames", GetVariablesNamesTag.class);

    // returns all variable types in a given symboltable.
    registerTag("get-variabletype", GetVariableTypeTag.class);

    // Returns current version, ideal to verify if the context
    // system is installed from the mathdox status page
    registerTag("get-version", StatusTag.class);
```

Once one or more tags have been created, a tag library has to be created as well. In this tag library all tags included are associated with a namespace for use from MathDox, as well as a name for each tag as they will be known in the MathDox system. The tag library as used for the standard MathDox Context tag set is given in Listing 61. Creating is done by sub classing the class. A tag library will be similar to MathDox Context library partly shown in Listing 61.

Listing 62: A snippet from the status page code with a call to a custom Jelly tag.

```
<para>
  <c:catch var='http.exception'>
    <mdc:get-version xmlns:mdc="jelly:org.mathdox.context.
        TheoryBlocksTagLibrary" />
  </c:catch>
  <c:choose>
    <c:when test='${not empty http.exception}'>
      <phrase role='error'>The MathDox Context system has not (
          properly) been installed</phrase>.
    </c:when>
  </c:choose>
</para>
```

Once a tag library with tags has been created the classes need to be compiled and stored in a jar file. This jar file then needs to be added to the jar library of the MathDox Player web application in the Tomcat server (or any other Java application server used).

In Listing 62 is shown how to call the `Status tag` the results were visible in Listing 45. Note that the namespace closely resembles the Java path of the tag library file.

A.6 PEOPLE WHO HAVE WORKED ON MATHDOX

The quality of the context implementation depends on the quality of the MathDox Player and format. It is therefore only fair to credit the people that have been working on the development of MathDox over the years: Arjeh Cohen, Hans Cuypers, Dorina Jibetan, Karin Poels, Manfred Riem, Olga Caprotti, Mark Spanbroek, Zubair Afzal, Mike Boldy, Matthijs Brouwer, Jan Willem Knopper and Rikko Verrijzer.

BIBLIOGRAPHY

[1] Activemath. http://www.activemath.org.

[2] Algebra interactive. http://www.mathdox.org/ida.

[3] Amazon. http://www.amazon.com.

[4] Ant. http://ant.apache.org.

[5] Apache project. http://www.apache.org.

[6] ASP.NET. http://www.asp.net/.

[7] Blackboard. http://www.blackboard.com.

[8] Blogger. http://www.blogger.com.

[9] Bol.com. http://www.bol.com.

[10] Chrome. https://www.google.com/chrome.

[11] The coq proof assistant. http://coq.inria.fr.

[12] Ctan latex packages. http://www.ctan.org/tex-archive/macros/latex/contrib/acrotex/.

[13] Cython. http://www.cython.org.

[14] Digital mathematics envirmonment. http://www.fi.uu.nl/dwo.

[15] Discrete algebra en geometry. http://www.mathdox.org/new-web/about.html.

[16] Docbook. http://www.docbook.org.

[17] Dragmath. http://www.dragmath.bham.ac.uk.

[18] Ebay. http://www.ebay.com.

[19] Eclipse. http://www.eclipse.org/.

[20] Elm-art. http://art2.ph-freiburg.de/Lisp-Course/.

[21] Emilea-stat. http://www.emilea.de.

[22] exist-db open source native xml database. http://exist.sourceforge.net.

[23] Experience mathness. https://www.tue.nl/en/education/tue-bachelor-college/education-structure/experience-mathness/.

[24] Facebook. http://www.facebook.com.

[25] Flicker. http://www.flicker.com.

[26] Formsplayer plugin for internet explorer. http://www.formsplayer.com.

[27] Gap. http://www-gap.mcs.st-and.ac.uk.

[28] Gap —groups, algorithms, programming— a system for computational discrete algebra. http://www.gap-system.org/gap.html.

[29] Geogebra. http://www.geogebra.org.

[30] Gmail. http://www.gmail.com.

[31] Gnu octave. http://www.gnu.org/software/octave.

[32] Gnu project. http://www.gnu.org/gnu/thegnuproject.en.html.

[33] Google. http://www.google.com.

[34] Graphml. http://graphml.graphdrawing.org/.

[35] Hotmail. http://www.hotmail.com.

[36] Html 4 forms. http://www.w3.org/TR/html401/interact/forms.html.

[37] Ida context implementation. http://evo02.win.tue.nl/rikkomathdoxplayer//experimental/idacontext/nodalpages/index.md.

[38] ING eBanking. https://mijn.ing.nl/internetbankieren.

[39] Instagram. https://instagram.com/.

[40] Intelligent feedback. http://uu.academia.edu/ChristianBokhove/Papers/219911/Intelligent_feedback_to_digital_assessments_and_exercises_in_Dutch_.

[41] Internet Explorer. http://www.microsoft.com/windows/internet-explorer.

[42] Java. http://java.sun.com.

[43] Java server pages. http://java.sun.com/products/jsp/index.jsp.

[44] Java servlet 2.4 specification. http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html.

[45] Javascript programming language. http://www.iso.org/iso/catalogue_detail.htm?csnumber=33835.

[46] Jboss. http://www.jboss.org.

[47] Jelly. http://commons.apache.org/jelly.

[48] Jgrapht. http://jgrapht.org/.

[49] JSXGraph. JSXGraphhttp://jsxgraph.uni-bayreuth.de/wp/.

[50] Knot theory. http://evo01.win.tue.nl/mathadore/knots.

[51] Leactivemath. http://www.leactivemath.org/.

[52] Lgpl. http://www.lgpl.org.

[53] Linkedin. http://www.linkedin.com.

[54] Macaulay2. http://www.math.uiuc.edu/Macaulay2.

[55] Magma. http://magma.maths.usyd.edu.au.

[56] Maple. http://www.maplesoft.com.

[57] Marktplaats. http://marktplaats.nl.

[58] Math player. http://www.dessci.com/en/products/mathplayer/.

[59] Math4all. http://www.math4all.nl.

[60] Mathadore. http://www.mathadore.nl.

[61] Mathdox. http://www.mathdox.org.

[62] Mathdox formula editor. http://mathdox.org/formulaeditor.

[63] Mathematica. http://www.wolfram.com.

[64] Mathematica player. http://www.wolfram.com/products/player.

[65] Mathematical markup language (mathml) version 2.0. http://www.w3.org/TR/MathML2.

[66] Mathematical markup language (mathml) version 3.0 (proposal). http://www.w3.org/TR/MathML3.

[67] Mathjax. http://www.mathjax.org.

[68] Matlab. http://www.mathworks.com/products/matlab.

[69] Maxima. http://maxima.sourceforge.net.

[70] Monet. http://monet.nag.co.uk/monet.

[71] Moodle. http://www.moodle.org.

[72] Mozilla firefox. http://www.mozilla.org/firefox.

[73] Nlg. http://webalt.math.helsinki.fi/content/results/generator/index_eng.html.

[74] Olat. http://www.olat.org.

[75] Onbetwist website. http://www.onbetwist.org/.

[76] Oncourse website. https://oncourse.tue.nl/2015/.

[77] Openmath. http://www.openmath.org.

[78] Openmath. www.openmath.org/cd.

[79] Openmath to mathml xslt translation scripts. http://www.openmath.org/standard/omxsl/index.html.

[80] Opera. http://www.opera.com.

[81] Orbeon forms. http://www.orbeon.com.

[82] Pari/gp. http://pari.math.u-bordeaux.fr.

[83] Perl. http://www.perl.org.

[84] PHP: Hypertext Preprocessor. http://www.php.net.

[85] Pragma ade. http://www.pragma-ade.nl.

[86] Python. http://www.python.org.

[87] Rabobank eBanking. https://bankieren.rabobank.nl/klanten.

[88] Rss 2.0 specification. http://www.rssboard.org/rss-specification.

[89] Ruby. http://www.ruby-lang.org.

[90] Safari. http://www.apple.com/safari/.

[91] Sage. http://www.sagemath.org.

[92] Sakai. http://www.sakaiproject.org.

[93] Scheme. http://www.scheme.com/tspl4/.

[94] The science project. http://www.symbolic-computation.org/The_SCIEnce_Project.

[95] Scorm. http://www.adlnet.gov/Technologies/scorm/default.aspx.

[96] Singular. http://www.singular-uni-kl.de.

[97] SOAP 1.2. http://www.w3.org/TR/soap12-part1/.

[98] Surf website. https://www.surf.nl/en.

[99] Telmme project. http://www.surffoundation.nl/nl/themas/innovatieinonderwijs/studiesucces/Documents/SURF.

[100] Texmacs. http://www.texmacs.org/.

[101] Tomcat. http://tomcat.apache.org.

[102] Trigonometric functions. http://www.univie.ac.at/future.media/moe/galerie/wfun/wfun.html.

[103] W3c. http://www.w3.org.

[104] Webalt. http://www.webalt.org.

[105] The webkit open source project. http://webkit.org.

[106] webmathematica. http://www.wolfram.com/products/webmathematica.

[107] Webwork. http://wwrk.maa.org.

[108] Wiris. http://www.math4more.com.

[109] Wiris graphical editor. http://www.wiris.com/content/view/20/3/lang,en.

[110] Wiskunde d. http://www.win.tue.nl/wiskunded/.

[111] Wolfram alpha. http://www.wolframalpha.com.

[112] Wortel tu/e. http://wortel.tue.nl.

[113] XForms. http://www.w3.org/MarkUp/Forms.

[114] XForms 1.0 recommendation. http://www.w3.org/TR/2003/REC-xforms-20031014/.

[115] Xforms 1.1 recommendation. http://www.w3.org/TR/xforms11/.

[116] XForms project at Mozilla. http://www.mozilla.org/projects/xforms.

[117] XInclude. http://www.w3.org/TR/xinclude.

[118] Xsl transformations (xslt) version 1.0. http://www.w3.org/TR/1999/REC-xslt-19991116.

[119] Yahoo! http://www.yahoo.com.

[120] YouTube. http://www.youtube.com.

[121] Core J2EE Patterns - Data Access Object, 2010.

[122] *Math-Bridge, bridging the math gap between high school and universities, In Proceedings EADTU annual conference 2011, pages 177 - 185*, 2011.

[123] Acrobat. Acrobat reader. http://get.adobe.com/uk/reader/, void.

[124] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers. Principles, Techniques and Tools*. Addison Wesley, 1986.

[125] AplusMath. Aplusmath flashcards. http://www.aplusmath.com/Flashcards.

[126] J. Axelsson and M. B. et al. Xhtml 2.0 - w3c working draft. http://www.w3.org/TR/xhtml2.

[127] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, and J. Simeon. Xml query language - w3c recommendation. http://www.w3.org/TR/xquery.

[128] B. Bos, C. Tantek I. Hickson, and H. W. Lie. Cascading style sheets - w3c recommendation. http://www.w3.org/TR/CSS.

[129] M. Bourne. Interactive mathematics. http://www.intmath.com/.

[130] C. Boyle and A. Encarnacion. Metadoc: An adaptive hypertext reading system. *User Modeling and User-Adapted Interaction*, 4:1–19, 1994.

[131] P. D. Bra. Design issues in adaptive web-site development. In *Proceedings of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW*, 2001.

[132] P. D. Bra, A. T. M. Aerts, B. Berden, B. de Lange, B. Rousseau, T. Santic, D. Smits, and N. Stash. Aha! the adaptive hypermedia architecture. In *Hypertext*, pages 81–84, 2003.

[133] P. D. Bra and J.-P. Ruiter. Aha! adaptive hypermedia for all. In *WebNet*, pages 262–268, 2001.

[134] P. D. Bra, D. Smits, and N. Stash. Creating and delivering adaptive courses with aha! In W. Nejdl and K. Tochtermann, editors, *EC-TEL*, volume 4227 of *Lecture Notes in Computer Science*, pages 21–33. Springer, 2006.

[135] P. D. Bra, D. Smits, K. van der Sluijs, A. Cristea, and M. Hendrix. Grapple: Personalization and adaptation in learning management systems. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010*, pages 3029–3038, Toronto, Canada, June 2010. AACE.

[136] P. Brouwer, H. Cuypers, and J. Knopper. Mathdox editor. In *Proceedings of the 5th Mathematical User-Interfaces Workshop 2009, Canada*, pages 1–8, 2009.

[137] P. Brusilovsky. Developing adaptive educational hypermedia systems: From design models to authoring tools.

[138] P. Brusilovsky, A. Kobsa, and W. Nejdl, editors. *The Adaptive Web, Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*. Springer, 2007.

[139] P. Brusilovsky and E. Millán. User models for adaptive hypermedia and adaptive educational systems. In *The Adaptive Web*, pages 3–53, 2007.

[140] A. Bunt, G. Carenini, and C. Conati. Adaptive Content Presentation for the Web. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, chapter 13, pages 409–432. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[141] S. Burbeck. Applications programming in smalltalk-80(tm): How to use model-view-controller (mvc). `http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html`, 1992.

[142] O. Caprotti, A. Cohen, and M. Riem. Java phrasebooks for Computer Algebra and Automated Deduction. In *Sigsam Bulletin*. 2000.

[143] F. Cena, L. Console, C. Gena, A. Goy, G. Levi, S. Modeo, and I. Torre. Integrating heterogeneous adaptation techniques to build a flexible and usable mobile tourist guide. *AI Commun.*, 19(4):369–384, 2006.

[144] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *CHI*, pages 17–24, 2000.

[145] J. Clark and S. DeRose. Xml path language - w3c recommendation. `http://www.w3.org/TR/xpath`.

[146] A. Cohen, H. Cuypers, J. Knopper, M. Spanbroek, and R. Verrijzer. Mathdox : A system for interactive mathematics. In *Proceedings ED-MEDIA 2008, Vienna*, 2008.

[147] A. Cohen, H. Cuypers, K. Poels, M. Spanbroek, and R. Verrijzer. WExEd - WebALT Exercise Editor for Multilingual Mathematics Exercises. `http://www.mathdox.org/wexed/wexed.html`.

[148] A. Cohen, H. Cuypers, and H. Sterk. *Algebra Interactive!* Springer-Verlag, Berlin, New York, etc., 1999.

[149] A. Cohen, H. Cuypers, and R. Verrijzer. Mathematical context in interactive documents. *Mathematics in Computer Science*, 3(3):331–347, 2010.

[150] A. M. Cohen, H. Cuypers, and E. R. Barreiro. Mathdox: Mathematical documents on the web contribution to the omdoc book, 2005.

[151] A. Cristea, S. D., and P. de Bra. Towards a generic adaptive hypermedia platform: a conversion case study. *Journal of Digital Information*, 8(3), 2007.

[152] A. Cristea and A. de Mooij. LAOS: Layered WWW AHS Authoring Model and their corresponding Algebraic Operators. In *WWW*, 2003.

[153] H. Cuypers, J. W. Knopper, M. Spanbroek, and R. Verrijzer. http://www.mathdox.org/mathdoxplayer/mathdox/manual/index.mathdox.

[154] H. Cuypers, J. W. Knopper, and H. Sterk. Mess: the mathdox exercise system, 2009.

[155] P. de Bra and N. Stash. Hypermedia structures and systems. adaptive course text offered at the tu/e, 2009.

[156] D. M. Deepak Alur, John Crupi. *Core J2Ee Patterns: Best Practices and Design Strategies*. Prentice Hall Professional, 2003.

[157] P. Dolog, N. Henze, W. Nejdl, and M. Sintek. Towards the adaptive semantic web. In *PPSWR*, pages 51–68, 2003.

[158] M. Dougiamas and P. Taylor. Moodle: Using learning communities to create an open source course management system. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, 2003.

[159] J. Edwards and K. Jones. Linking geometry and algebra with geogebra. *Mathematics Teaching*, 194, 2006.

[160] E. Fernandes and A. N. Kumar. A tutor on scope for the programming languages course. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 90–93, New York, NY, USA, 2004. ACM.

[161] T. Fischer, F. Bakalov, and A. Nauerz. Towards an automatic service composition for generation of user-sensitive mashups. In *LWA*, pages 14–16, 2008.

[162] E. T. Freeman, E. Robson, B. Bates, and K. Sierra. *Head First Design Patterns*. O'Reilly Media, 2004.

[163] E. Gamma, R. Helm, R. E. Johnson, and J. M. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison Wesley, 1995.

[164] X. Gang. Wims: An interactive mathematics server. `http://mathdl.maa.org/mathDL/23/?pa=content&sa=viewDocument&nodeId=354`, 2001.

[165] P. Gärdenfors. How to make the semantic web more semantic. In A. C. Varzi and L. Vieu, editors, *Formal Ontology in Information Systems: proceedings of the third international conference (FOIS-2004)*, volume 114 of *Frontiers in Artificial Intelligence and Applications*, pages 17–34. IOS Press, 2004.

[166] F. Ghali, R. I. Cristea, and C. Stewart. My online teacher 2.0.

[167] J. Hartley and D. Sleeman. Towards more intelligent teaching systems. *International Journal of Man-Machine Studies*, 5(2):215 – 236, 1973.

[168] E. Hilf, M. Kohlhase, and H. Stamerjohanns. Capturing the content of physics: Systems, observables, and experiments. In *Mathematical Knowledge Management, number 4108 in LNAI*. Springer, 2006.

[169] J. W. Knopper. http://mathdox.org/mathdoxplayer/mathdox/latex/index.mathdox.

[170] D. E. Knuth. *TeX: The Program*. Addison-Wesley, 1986.

[171] D. E. Knuth. *The TeXbook*. Addison-Wesley, 1986.

[172] A. Kobsa. User modeling: Recent work, prospects and hazards, 1993.

[173] A. Kohlhase and M. Kohlhase. Semantic knowledge management for education. In *Proceedings IEEE*, 2008.

[174] M. Kohlhase. *OMDoc - An Open Markup Format for Mathematical Documents [version 1.2]*, volume 4180 of *Lecture Notes in Computer Science*. Springer, 2006.

[175] A. Konovalov and S. Linton. Scscp symbolic computation software composability protocol version 1.1.4, 2009.

[176] M. Lebrun, F. Docq, and D. Smidts. Claroline, an Internet Teaching and Learning Platform to Foster Teachers' Professional Development and Improve Teaching Quality : First Approaches. *AACE*, 17:347–362, 2009.

[177] H. Lieberman and H. Liu. Adaptive linking between text and photos using common sense reasoning. pages 2–11. Springer, 2002.

[178] D. Marquès, R. Eixarch, G. Casanellas, B. Martínez, and T. J. Smith. Wiris om tools: A semantic formula editor. In *Proceedings of the 2006 Mathematical User-Interfaces Workshop, St Anne's Manor, Workingham, United Kingdom*, page 06, 2006.

[179] C. C. Marshall and F. M. Shipman. Which semantic web? In *Hypertext*, pages 57–66, 2003.

[180] E. Melis, E. Andres, J. Budenbender, A. Frischauf, E. M. E. AndrÃšs, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Activemath: A generic and adaptive web-based learning environment. *International Journal of Artificial Intelligence in Education*, 12:385–407, 2001.

[181] E. Melis, G. Goguadze, P. Libbrecht, and C. Ullrich. Activemath - a learning platform with semantic web features, 2009.

[182] E. Melis and J. H. Siekmann. Activemath: An intelligent tutoring system for mathematics. In *ICAISC*, pages 91–101, 2004.

[183] C. Müller. *Adaptation of Mathematical Documents*. Ph.D. Thesis, Jacobs University, Bremen, Germany, May 2010.

[184] C. Müller and M. Kohlhase. panta rhei. In *LWA*, pages 318–323, 2007.

[185] C. Müller and M. Kohlhase. Context-Aware Adaption A Case Study on Mathematical Notations. In *Information Systems Management*, 2009.

[186] L. Nguyen and P. Do. Learner model in adaptive learning. `http://www.waset.org/journals/waset/v45/v45-70.pdf`, 2008.

[187] L. Nguyen and P. Do. Learner model in adaptive learning. 2008.

[188] L. Nguyen and P. Do. Learner model in adaptive learning, 2008.

[189] S. Pemberton and D. A. et al. Xhtml 1.0 the extensible hypertext markup language - w3c recommendation. `http://www.w3.org/TR/xhtml1`.

[190] M. C. Polson and J. J. Richardson, editors. *Foundations of intelligent tutoring systems*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1988.

[191] C. Sangwin. Stack: making many fine judgements rapidly, 2007.

[192] W. Scholl, C. König, B. Meyer, and P. Heisig. The future of knowledge management. an international delphi study. *Knowledge Management, 8*, 2004.

[193] M. Seppälä, S. Xambó, and O. Caprotti. Novel Aspects of the Use of ICT in Mathematics Education. In *Proceedings of the International Conference on Engineering Education, Instructional Technology, Assessment, and E-learning (EIAE)*, 2006.

[194] K. Sierra and B. Bates. *Head first EJB - passing the Sun certified business component developer exam: a brain-friendly study guide.* O'Reilly, 2003.

[195] A. Stevenson. *Oxford Dictionary of English.* Oxford reference online premium. OUP Oxford, 2010.

[196] H. Stuckenschmidt and M. C. A. Klein. Reasoning and change management in modular ontologies. *Data Knowl. Eng.*, 63(2):200–223, 2007.

[197] R. Verrijzer and M. Spanbroek. http://mathdox.org/new-web/manuals/mathdoxmanual.pdf, 2007.

[198] W3C. http://www.w3.org/News/2009#item119, 2009.

[199] E. Wenger. *Communities of Practice.* Cambridge University Press, 1999.

[200] O. Yasuhisa, W. Kenzi, and K. Hiroki. An implementation of an intelligent tutoring system (its) on the world-wide web (www) : Individualized tutoring mechanism in the www framework. *Educational technology research*, 19(1):35–44, 1996-12.

CURRICULUM VITAE

Rikko Verrijzer was born on February 7th, 1975 in Alkmaar, the Netherlands.

He received his MAVO (1991) and HAVO (1993) diplomas at the secondary school Bernardus Alfrink College in Schagen, the Netherlands. Later, in 1993, he started his computer science bachelor at the Hogeschool Enschede, which he obtained in 1997. He continued his computer science studies with a study of Technical Computer Science at the Universiteit Twente, also in Enschede. He graduated in 2002 in the group Distributed Systems with the thesis "Het beschermen van mobiele agents tegen kwaadwillende agentservers" (protection of mobile agents against malicious hosts).

In 2000, on a parallel track, Rikko started his professional career at the startup company Tryllian, where he was introduced to working on mobile agents. In 2002 he switched to a telecom company to work with texting systems for television shows. He started working at the Technische Universiteit Eindhoven in 2005 as a developer within the European WebALT project.

Rikko started his PhD on context in mathematical documents in 2007 at the TU/e under the supervision of Prof. Dr. Arjeh M. Cohen and Dr. Hans Cuypers. The experience with interactive mathematics, OpenMath and MathDox gained during the WebALT project and the work of Prof. Paul De Bra gave him the motivation to address problems that arise while studying mathematical texts and which have their roots in the background and context of the reader of those texts.

Having gained experience as a teacher during his PhD studies, Rikko took up a computer science teaching position at the Copernicus Scholen Gemeenschap, a secondary school in Hoorn. He worked there from 2010 till 2011 and continued teaching and tutoring at the Anglo American School, an international high school in Sofia from 2011 till 2014. After the summer of 2014 he started a computer science teacher position at Bertrand Russell College, a high school in Krommenie. He still holds this position.

The present dissertation contains the results of Rikko's PhD work from 2007 to 2015.