Ontology and Reuse in Model Synthesis

Virginia Biris Brilhante

Doctor of Philosophy

School of Informatics

University of Edinburgh

2003

Abstract

A much pursued ontological capability is knowledge reuse and sharing. One aspect of this is reuse of ontologies across different applications, including construction of other ontologies. Another aspect is the reuse of knowledge that is founded on ontologies, which, in principle, is attainable in that an ontology renders well-defined and accessible the meaning of concepts that underly the knowledge. In this thesis we address both these aspects of reuse, grounding the study in the problem of synthesis of structural ecological models.

One basis for the design of ecological models, along with human expertise, is data properties. This motivated the construction of a formal ontology — Ecolingua — where we define concepts of the ecological data domain. The diversity of the domain led us to use the Ontolingua Server as an ontology design tool, since it makes available an extensive library of shareable ontologies. We reused definitions from several of these ontologies to design Ecolingua. However, Ecolingua's specification in the representation language of the Ontolingua Server did not easily translate into a manageable, executable specification in our choice of implementation language, which required us to develop our own complementary tools for scope reduction and translation. Still, a great deal of manual re-specification was necessary to achieve an operational, useful ontology. In sum, reuse of multiple ontologies in ontology construction employing state-of-the-art tools proved impractical. We analyse the practical problems in reuse of heterogeneous ontologies for the purpose of developing large, combined ones.

In its operational form, Ecolingua is used to describe ecological data to give what we call metadata, or, said differently, instantiated ontological concepts. Synthesis is achieved by connecting metadata to model structure. Two approaches were developed to do this. The first, realised in the Synthesis-0 system, uses metadata alone as synthesis resource, while the second, realised in the Synthesis-$\mathcal{R}$ system, exploits Ecolingua to reuse existing models as an additional synthesis resource. Both Synthesis-0 and Synthesis-$\mathcal{R}$ are working systems implemented as logic programs. The systems were empirically evaluated and compared on run time efficiency criteria. Results show a remarkable efficiency gain in Synthesis-$\mathcal{R}$, the system with the reuse feature. We generalise the results to systems for synthesis of structural models informed by domain data. Once a data ontology and ontology-constrained synthesis mechanisms are in place, existing models can efficiently be reused to produce new models for new data.

To the three of us — Sofia, Elias and I — for having so lovingly endured and enjoyed the challenges and discoveries of these PhD years in Edinburgh.

Acknowledgements

This is a moment I longed for: to at last publicly acknowledge those who have contributed, in one way or another, to my PhD work and make my success their own. The effort and care of many people have carried me this far and it is a daunting task to justly thank them all, but I will try my best. Here we go.

I have been most fortunate to have had Dave Robertson as supervisor. Dave's first ever PhD student has coined what to me still best sums up his style of supervision: a rare ability to seamlessly bring the best out of his students. Dave is truly a master, in research and in life.

My gratitude also to my second supervisor and ecological modelling expert, Robert Muetzelfeldt. Robert had to put up with my very basic knowledge of ecological modelling and made an invaluable contribution to the interdisciplinary aspect of my work, especially in the early stages. It was not always easy to reconcile our disparate interests, but now we know that the differences paid off.

Sincere thanks to my examiners Alun Preece and Stuart Aitken, and also to Robert Rae, for their effort, time and feedback on my thesis, and for being at the same time challenging and encouraging during the Viva.

Not long after I started my project, I attended a course on analysis of environmental data with machine learning methods in Ljubljana. There I had the pleasure of meeting Ivan Bratko for the first time, when we enthusiastically talked about my incipient research ideas. I would like to thank Ivan for his comments and interest in my project. This visit also put me in touch with Ljupčo Todorovski to whom I am indebted for all his assistance with the Lagramge system.

Acknowledgment is due to the support team of the Ontolingua Server, in particular James Rice who has been extraordinarily prompt and resourceful in responding to my many e-mail messages.

I have very much enjoyed being a member of the Software Systems and Processes (SSP) research group. These folk just will not let your research project be solitary or too introspective. You can always count on a knowledgeable audience eager to hear about your latest and wildest ideas (not necessarily research-related) in our delightful weekly meetings, which I was gladly in charge of organising for over one year. Thank you all, past and current members — Daniela Carbogim, Alberto Castro, João Cavalcanti, Luigi Ceccaroni, Jessica Chen-Burger, Stefan Daume, Yannis Kalfoglou, Renaud Lecœuche, Siu-Wai Leung, Chris Lin, Jarred McGinnis, Steve Polyak, Stephen Potter, David Robertson, Marco Schorlemmer, Jonathan Tonberg, Wamberto Vasconcelos, Richard Walker and Chris Walton — for sharing so much and contributing to the meetings while I was the organiser. For the last year or so, particularly, Wamberto Vasconcelos has been someone I constantly turned to for advice — technical or otherwise — and I cannot thank him enough for all he has given me. Chris Walton also, had the most appreciated disposition to thoroughly proofread, in a just-in-time basis, all the chapters I produced during the last months of thesis writing.

I am also grateful to other fellow PhD students — Marcio Brandão, Marco Carvalho, Alzira Leite, Natasha Lino, Alison Pease, Manuel Marques Pita, Sonia Schulenburg, Clauirton Siebra and Henrik Westerberg — for their camaraderie grown out of the many chats we had and the long days and nights spent together in the office.

My work has been supported in many ways by the administrative, computing and library staff of the Centre for Intelligent Systems and their Applications (CISA) and the School of Informatics: Donna Bolland, John Berry, Neil Brown, Johann Bryant, Jean Bunten, Deirdre Burke, Jennie Camps-Douglas, David Dougal, Olga Franks, Janice Gailani, Neil McGillivray, Jane Rankin, Gordon Reid, Michelle Siszczuk and Craig Strachan. They have always been friendly and efficient beyond belief and all their help is warmly acknowledged.

The College of Science and Engineering has provided complementary programmes — Transferable Skills, the Mentoring and Springboard programmes for women in science and engineering — that significantly enriched my skills and prospects as a researcher. Through these programmes I met my mentor Claire Grover who generously shared with me her experience in combining motherhood with being a researcher in informatics.

I acknowledge the Brazilian funding agency CAPES — Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — for the indispensable financial support under the grant BEX1498/96-7, and especially Vanda Lucena for her unfailing efficiency and courteousness in dealing with all matters concerning my grant.

While I was here in Edinburgh, my colleagues at the Department of Computing Science, University of Amazonas, kept up with all the teaching and administration work. I am glad to go back and take my turn so as they too can have the same opportunity. Special thanks to Ana Lúcia dos Santos for her friendship, most of all, and for kindly looking after my affairs in Manaus during my absence.

Towards the end of my studies, when steam was running low, I had a providential encounter with Helen Campbell. It was under her loving care that I made it through.

I am one of those few lucky people who have their parents, brothers and sisters, standing by them no matter what they set themselves to do in life. And not long ago I had a valuable addition to the team, my mother-in-law, Sofia Biris, the most unselfish person I have ever known. I will always be deeply grateful to my father, Benedito Brilhante, for teaching me to be brave, to my mother, Nancy Brilhante, for teaching me to be kind, and to them both for giving me the freedom and the means to find my own ways in between.

I am overjoyed that my husband, Elias Biris, and I are completing our PhDs together. I have an enormous admiration for him for having achieved that while also working full-time for the last four years as a software engineer. We carried each other through the years. He says he owes me his PhD, but the truth is, even though he does not take me seriously when I say so, I owe him mine and much more.

Our daughter, Sofia, has been our greatest joy and source of inspiration, amazing us each day with her grace. When we started this journey she was only a baby. She has learned to walk, to talk (in three languages, by the way), started her schooling — in all, discovered herself and a whole new world — in the time it took us to complete just one degree. We are humbled by you.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Virginia Biris Brilhante*)

Table of Contents

List of Figures **xv**

List of Tables **xvii**

I Introduction and Background 1

1 Motivation and Research Scenario **3**

1.1 Ontology-Driven Knowledge Reuse 3

1.2 Synthesis of Conceptual Models from Ontological Data Descriptions . 4

1.3 Setting the Research Scenario 6

1.4 Thesis Outline 7

2 Literature Survey **9**

2.1 Ontologies and Knowledge Reuse 9

2.1.1 The Ontolingua Server 10

2.1.2 Approaches and Experiences on Ontology-Driven Knowledge Reuse 11

2.2 Environmental Ontologies 16

2.3 AI Techniques in Design and Synthesis of Ecological Models 19
2.3.1 Object-Oriented Modelling 20
2.3.2 Logic-Based Modelling 21
2.3.3 Model Induction and Equation Discovery 24
2.3.4 Compositional Modelling 27
2.4 An Introduction to System Dynamics Modelling 29

II Ontology Engineering **35**

3 The Ecolingua Ontology **37**
3.1 Forms of Concept Definitions 39
3.2 The Ecological Data Ontology 40
3.2.1 Quantity 42
3.2.2 Contextual Data 54
3.3 The Ecological Model Requirements Ontology 56

4 On the Engineering of Ecolingua **59**
4.1 Conceptual Ecolingua in the Ontolingua Server 61
4.2 Translating Conceptual into Executable 61
4.2.1 Translation by the Ontolingua Server into Prolog Syntax . . . 63
4.2.2 Cleaning up Extraneous Clauses 64
4.2.3 Pruning the Class Hierarchy 65
4.2.4 Pruning Clauses 69
4.2.5 Transforming Logical Sentences into Horn Clauses 73

4.3 Experiment Discussion 75

4.3.1 Employing the Ontolingua Server 76

4.3.2 Ontology Reuse 77

III Model Synthesis and Reuse 81

5 Metadata: Sources and Ecolingua Descriptions 83

5.1 Metadata from Field Data 85

5.2 Metadata from Preliminary Modelling Information 87

5.2.1 Model Objectives 87

5.2.2 Model Assumptions 89

6 Heuristic Knowledge for Synthesis of Model Components 93

6.1 Metadata Constraints 96

6.1.1 Underlying Ontology Constraints 96

6.1.2 Library of Metadata↔Model Association Rules 97

6.1.3 Influence Constraints 102

6.1.4 Evidence Classes of Model Elements 106

6.2 Integrity Constraints 108

6.3 Synthesis Constraints of Model Components 110

6.3.1 Constraints of Model Elements 110

6.3.2 Summary of (Non-)Conflicts between Model Elements 120

6.3.3 Constraints of Model Connections 123

6.3.4 Summary of Model Connections 137

7 The Synthesis-0 System 141

7.1 System Architecture . . . 141

7.2 Algorithms . . . 142

7.3 Multiple Model Solutions . . . 152

7.4 Patterns and Mechanisms of Inference and their Implementation . . . 154

7.4.1 Ontological Checking: Establishing Ecolingua-Compliant Metadata . . . 155

7.4.2 Abduction: Connecting Metadata to Model Structure . . . 158

7.5 Worked Example . . . 162

7.5.1 Tracing the Synthesis Search Tree . . . 164

7.6 Meta-Interpreting Synthesis-0 (and Synthesis-$\mathcal{R}$) . . . 167

7.6.1 The Meta-Interpreter . . . 168

8 Reuse via Synthesis and the Synthesis-$\mathcal{R}$ System 173

8.1 Reuse via Synthesis . . . 174

8.2 System Architecture . . . 177

8.3 Algorithms . . . 177

8.4 Multiple Model Solutions . . . 188

8.5 Inference Pattern . . . 191

8.6 Reuse-Synthesis Constraints of Model Components . . . 192

8.6.1 Constraints of Model Elements . . . 195

8.6.2 Constraints of Model Connections . . . 198

8.7 Worked Example . . . 206

8.7.1 Tracing the Reuse-Synthesis Search Tree . . . 209

IV Evaluation and Conclusions **215**

9 Empirical Evaluation of Synthesis Run Times **217**

9.1 Comparative Featural Characterisation of Synthesis-0 and Synthesis-$\mathcal{R}$ 219

9.2 Experimental Hypothesis and Measured Variable 222

9.3 Experimental Procedure 223

9.3.1 Other Controls and Simplifications 228

9.4 Data 229

9.4.1 Models Sample 229

9.4.2 Generating Artificial Metadata 230

9.4.3 Collecting Run Time Measurements 232

9.5 Results 234

9.5.1 Synthesis-0 Run Times 235

9.5.2 Synthesis-$\mathcal{R}$ Run Times 237

9.5.3 Comparative Result 239

9.5.4 Generalisation 241

10 Looking Back and Looking Ahead **243**

10.1 Contributions 243

10.1.1 A Mechanism for Model Reuse via Synthesis and Data Ontologies 243

10.1.2 A Procedure for Empirical Evaluation of Synthesis Systems with Reuse 245

10.1.3 A Complete Process of Ontology Engineering and Application 246

10.2 Further Work 247

10.2.1 Availability, Selection and Modularity of Reusable Models . . 248
10.2.2 Web-Enabling and Tools 249
10.2.3 Domain Modelling Languages 249
10.2.4 Link with Quantitative Modelling 250

A Low-Level Ecolingua **251**
A.1 Ecolingua Quantities and their Units and Scales 251
A.2 Units and Scales of Measure 253
A.2.1 Scales of Measurement 255
A.3 Dimensions of Units and Scales 256
A.4 System of Units 257
A.5 Reals and Real-Valued Expressions 259

B Metadata Specifications of the Pond Management Example **261**

C Synthesis-$\mathcal{R}$'s Intermediate Algorithms **267**

D Input and Output Settings for Artificial Metadata Generation **277**
D.1 Input Settings 277
D.2 Output Settings 277

Bibliography **281**

List of Figures

1.1 Model synthesis and reuse scenario. 6

2.1 The Machine Learning task from a modelling perspective. 25

2.2 System dynamics diagrams exemplified: a fish pond management model. 32

3.1 Ecolingua's composition: the Ecological Data Ontology and the Model Requirements Ontology. 38

3.2 Class hierarchy of the Ecological Data Ontology. 41

3.3 Class hierarchy of the Model Requirements Ontology. 56

4.1 Extract from early conceptual Ecolingua. 62

4.2 KIF axioms (left) and their Ontolingua Prolog syntax translation (right). 64

4.3 Extract of Ecolingua's class hierarchy in the Ontolingua Server and pruning paths. 67

7.1 The architecture of the Synthesis-0 system. 142

7.2 Ecolingua compliance checking mechanism. 157

7.3 Prolog specification of the abductive mechanism in Synthesis-0. . . . 161

7.4 A portion of a Synthesis-0 search tree. 163

7.5 Prolog meta-interpreter shared by Synthesis-0 and Synthesis-$\mathcal{R}$. 169

7.6 Prolog specification of the Ecolingua compliance checking mechanism. 171

8.1 The metadata-only and the reuse-synthesis approaches. 175

8.2 The architecture of the Synthesis-$\mathcal{R}$ system. 177

8.3 Forms of flows. 178

8.4 Forms of links. 179

8.5 Hierarchy of Synthesis-$\mathcal{R}$'s algorithms. 179

8.6 A portion of a Synthesis-$\mathcal{R}$ search tree. 208

9.1 Ontology-founded knowledge levels. 218

9.2 Experimental procedure for measuring Synthesis-0 run times. 223

9.3 Experimental procedure for measuring Synthesis-$\mathcal{R}$ run times. Part I: $M_{ref} \supseteq M'_k = M_{target_k}$. 226

9.4 Experimental procedure for measuring Synthesis-$\mathcal{R}$ run times. Part II: $M_{ref_k} = M'_k \subseteq M_{target}$. 227

9.5 Sample model M_1 in system dynamics diagrammatic notation. 231

9.6 Artificial metadata set generated for sample model M_1. 231

9.7 Automated procedure for collecting Synthesis-0 run time measurements. 232

9.8 Automated procedure for collecting Synthesis-$\mathcal{R}$ run time measurements. 232

9.9 Synthesis-0 run times. 235

9.10 Synthesis-$\mathcal{R}$ run times. 237

List of Tables

5.1 Excerpt of data on processes occurring within the pond system. 86

6.1 Library of Metadata↔Model association rules. 99

6.2 State variables vs intermediate variables. 121

6.3 State variables vs parameters. 121

6.4 Intermediate variables vs parameters. 122

6.5 Driving variables vs state variables, intermediate variables and parameters. 122

9.1 Featural characterisation of Synthesis-0 and Synthesis-$\mathcal{R}$. 220

9.2 Models sample. 229

B.1 Data on historical states of the pond system. 266

B.2 Data on processes occurring within the pond system. 266

D.1 Input settings for generation of artificial metadata. 278

D.2 Output settings for generation of artificial metadata: model requirements descriptions. 278

D.3 Output settings for generation of artificial metadata: data descriptions. 279

Part I

Introduction and Background

Chapter 1

Motivation and Research Scenario

Ontologies have been proposed as a means of specification that enables knowledge reuse and sharing (Neches et al., 1991). They make explicit what is meant by terms representing concepts and relations, which is necessary for knowledge to be appropriately reused in different contexts and by different users (humans or computer systems). This thesis contributes towards turning such proposal into reality.

We start this chapter with an introduction to the endeavour of knowledge reuse enabled by ontologies. Next, we identify a modelling problem and a knowledge reuse scenario for development of the research. Finally an outline of the thesis is given.

1.1 Ontology-Driven Knowledge Reuse

In a broad perspective, reused knowledge may consist, at a first level, of ontologies themselves, or more extensively, of knowledge that is founded in ontologies. Most of the literature deals with reuse of ontologies themselves, either in the construction of new ontologies or as part of knowledge-based applications. In works on knowledge engineering using foundational ontologies, the knowledge is largely *intended* for reuse, but little has been reported on its actual reuse or on tangible benefits obtained thereby.

Furthermore, the use of formal definitions of concepts in an ontology has been under-exploited, not going much further beyond its role as a knowledge specification language. In principle, ontologies can play a more active and direct part in knowledge-based systems, for example, in the identification of adequate pieces of knowledge for reuse, or in inference as the system performs tasks.

It is the pursuit of ontologies as a technology that enables knowledge reuse that generally motivates this thesis. We aim at exploring such potential and demonstrating ways in which it can be realised in practice to add value to applications.

In the next section we identify a modelling problem to give scope to the exploration.

1.2 Synthesis of Conceptual Models from Ontological Data Descriptions

The intrinsic relationship between models and data is amply acknowledged in the empirical sciences:

> Science consists of confronting different descriptions of how the world works with data, using the data to arbitrate between the different descriptions, and using the best description to make additional predictions or decisions. These descriptions of how the world might work are hypotheses, and often they can be translated into quantitative predictions via models. (Hilborn and Mangel, 1997)

Data is particularly valuable in substantiating models when the real-world systems they represent are too complex to be perfectly understood and described without bias. Using data derived from observation to inform model design allows model simulation results to be accounted with respect to the data which adds credibility to them. Also, a common methodological approach that facilitates understanding of complex systems is to firstly design a conceptual (or qualitative) model which is subsequently used as a framework for specification of a quantitative model. These are reasons that motivate our choice of synthesis of models in the environmental domain as the intelligent task where we ground our investigation on knowledge reuse (see Section 1.3).

Data sets given to support modelling of complex systems, however, contain mainly quantitative data which is not of much help in the creative early stages of conceptual model formulation. Quantitative data is usually only directly applied in later stages, to estimate parameters and calibrate the model. During conceptual model formulation, modellers subjectively assess whether the data at hand will support their understanding of the system. In doing so, they focus not on data values as such, but rather on higher-level properties of the data, for example, availability, meaning and types of measurements, how the data is structured, how representative the samples are, etc. But in practice it is difficult for modellers to make such assessment and design decisions consistently with the available data because in its low representational level quantitative data does not directly connect to high-level model conceptualisation. As a result conceptual models are often led astray from their supporting data sets.

Here is where ontologies come in. An ontology of properties of domain data can provide a unifying conceptual vocabulary for representation of data sets, by way of which the data's level of abstraction is raised to facilitate connections with conceptual models. We refer to data that is represented through such a vocabulary as *metadata*. To illustrate data, metadata, ontology, and their interrelations in the context of this research project, let us suppose a forest logging operation where the amount of timber of each logged tree is measured. The actual measures of tree timber, e.g., 500 kg, are *data*. The data property level, in turn, does not concern data values as such but, say, their existence, or the entities they are attributes of. The ontology vocabulary comprises terms representing concepts that belong to this level. For example, later in the thesis we present the *ontology* term $amt\_of\_mat(A, Mt, E, U)$ used to represent a relation between an amount A of a material Mt in an entity E expressed in the unit of measure U. Our amount of timber variable can be represented as $amt\_of\_mat(t, timber, tree, kg)$, an example of *metadata*.

The interesting problem of model synthesis from data ontologies provides the scope for our investigation on ontology-driven knowledge reuse. Having a data ontology giving support to conceptual model synthesis can be beneficial in a number of ways. First, the ontology provides for suitably represented data, as already mentioned. Second, a synthesis process that transforms ontological data concepts, as opposed to specific data, into model elements and structures has its applicability widened. And third, when represented through the ontology the data becomes more concise, in that many data values can be collectively described through a single ontological concept. Synthesis can then start from this concise account of the data, avoiding the problems associated with manipulating large data volumes as in model induction approaches (Kubat et al., 1997).

1.3 Setting the Research Scenario

Consider a scenario of synthesis of models of a certain domain as depicted in Figure 1.1. The ontology provides a formally defined vocabulary of data properties in the domain, through which data sets about systems of interest are described giving metadata. Based on such metadata, models of the systems are synthesised by reusing existing models.

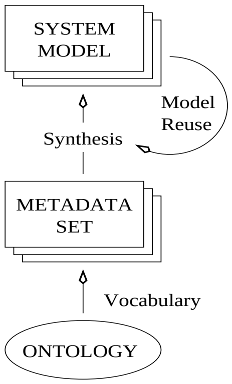


Figure 1.1: Model synthesis and reuse scenario.

This is the scenario within which knowledge reuse supported by ontologies is investigated in this thesis. We shall explore, in particular, roles that data ontologies can play in model synthesis as well as gains that might derive from that. In the process, three forms of knowledge reuse are addressed, which we enumerate below:

1. Reuse of existing structural models in the synthesis of new models given a new metadata set, where the reuse is enabled by the data ontology (Figure 1.1). If

we assume that the existing models have been designed backed by data, then to reuse one of them means to share its embedded modelling knowledge with the new model's data.

2. Reuse of domain-specific heuristic knowledge on connections between metadata and model structure. The knowledge is firstly formalised to connect metadata to model structure within an initial synthesis approach where existing models are *not* reused (this would correspond to Figure 1.1 without the Model Reuse arrow), and subsequently harnessed to, conversely, connect the structure of reused models to new metadata in the reuse-oriented synthesis approach.

3. Reuse of multiple existing ontologies in the construction of a domain-specific data ontology.

Such an ontology is built in the context and for the purpose of this research. We do so from the standpoint of ontology engineers trying to employ existing ontologies as well as existing technology as much as possible.

The research scenario is grounded in the ecological modelling domain:

- the synthesised models are structural system dynamics models, which represent flow of material through ecological systems;
- the data sets that inform model synthesis comprise field measurements of ecological variables and other preliminary information such as model objectives and assumptions; and
- the ontology contains concept definitions of ecological data properties.

1.4 Thesis Outline

The thesis is divided into four main parts as follows:

Part I Introduction and Background. In this chapter we have motivated the research and delineated a problem and a scenario for it. Chapter 2 surveys literature on ontology-driven knowledge reuse, on ontologies related to the ecological domain, and on AI techniques applied to ecological modelling. The chapter concludes with an introduction to the kind of models we tackle in our work on synthesis, namely, system dynamics models.

Part II Ontology Engineering. Chapter 3 presents Ecolingua, the ontology we built for description of ecological data. In Chapter 4 we report on the knowledge engineering effort involved in this ontology construction experiment.

Part III Model Synthesis and Reuse. This is the central part of the thesis. It starts with Chapter 5 showing by example how data is described through Ecolingua to yield metadata. In Chapter 6 we define the heuristic knowledge used to connect metadata and model structure in the form of synthesis constraints. Chapter 7 describes Synthesis-0, the first of the two working systems we developed which have solution of the constraints as part of the synthesis process. Chapter 8 introduces the approach of reusing existing models in the synthesis of new models and then describes our second system, Synthesis-$\mathcal{R}$, which implements the approach.

Part IV Evaluation and Conclusions. Chapter 9 presents an empirical comparative evaluation of Synthesis-0 and Synthesis-$\mathcal{R}$ on the run time efficiency criterion, showing a significantly better performance by the latter system. This empirical result is generalised in the end of the chapter. To conclude, Chapter 10 enumerates the contributions of the thesis and motivates further work.

Chapter 2

Literature Survey

This literature survey focuses on ontology-driven *knowledge reuse* and on *model synthesis*, the two grand themes in the thesis. Section 2.1 reviews work on reuse of ontologies and reuse of ontology-founded knowledge. As the thesis contains an ontology for the ecology domain, related ontology building efforts are surveyed in Section 2.2. Subsequently, we turn our attention in Section 2.3 to the development and use of artificial intelligence techniques in the task of ecological model construction. The last section is dedicated to a brief introduction of system dynamics, the modelling paradigm in which our synthesis techniques are grounded.

2.1 Ontologies and Knowledge Reuse

The primary form of ontology-driven knowledge reuse is where the ontologies themselves are reused. Regardless of whether or not an ontology is created with a specific application in mind, one hopes that it will be reused, perhaps after some modifications, either across different applications and/or in the construction of other ontologies (Uschold and Gruninger, 1996). Another, less explored, form of ontology-driven knowledge reuse is when not the ontology itself is reused, but knowledge that is founded in or derived from it.

2.1.1 The Ontolingua Server

The literature indicates that the most widely applied tool so far in reuse of ontologies themselves has been the web-based Ontolingua Server (Farquhar et al., 1996; Ontolingua Server, 1995), developed and hosted by the Knowledge Systems Laboratory, Stanford University. As its name denotes, the tool is built on Ontolingua, a system created to solve the portability problem for ontologies (Gruber, 1993). Ontolingua has been proposed as a standard, system-independent interlingua for ontologies specification that can be translated and ported into and out of system-specific representation languages (implementation languages included). Ontolingua's own underlying representation language is KIF, Knowledge Interchange Format, a prefix monotonic first-order logic language extended with set theory (Genesereth and Fikes, 1992). The front-end representation style of Ontolingua, or its meta-ontology, in the Server is based on the Frame-Ontology (Gruber, 1993). Ontologies are specified through the frame-system constructs (class, slot, relation, function, etc.), and when these do not suffice, axioms can be written in KIF.

One of the main resources the tool provides is an extensive library of shareable ontologies. Other useful facilities are the group sessions allowing different users to simultaneously work on the design of the same ontology, and a service of automatic translation of ontologies into several target languages, such as Clips, Cml, Loom, and the so-called Prolog Syntax language, of particular interest to us as we employ Prolog to implement our Ecolingua ontology. The translation into Prolog Syntax, however, does not produce Prolog runnable code but logical sentences in a KIF-like Prolog readable syntax. KIF, a first-order logic language, subsumes Horn clause logic, therefore only a subset of it is translatable into standard Prolog. A note on design decisions for translating Ontolingua to Prolog Syntax can be found in (Farquhar, 1995).

The translation service gives the options of translating an ontology in isolation, in which case the referenced definitions in other ontologies are not included, or translating the ontology including the referenced definitions through theory inclusion (Farquhar et al., 1996) — when an ontology (theory) A refers to a concept of another ontology (theory) B then all definitions in B are incorporated by A. Through this means, the resulting ontology is simply the union of the ontologies in full. As far as knowledge sharing is concerned, we are interested in translation mechanisms that provide for inclusion of referred definitions (Cohen et al., 1999) (but not necessarily inclusion of the complete theories). The motivation for defining a class C of a new ontology as a subclass of class D of an existing ontology, for example, resides on C inheriting "for free" the properties of D. Using the first translation option (the new ontology in isolation) would translate the definitions of C alone and the desired properties of D would be lost (unless there was some dynamic way of accessing the properties of D without including them in the translation output, but there is not).

In Chapter 4 we will see that while using the Server to design Ecolingua, concepts from five of its shareable ontologies were reused. Employing the translator, with Prolog Syntax as choice of target language, produced a specification including Ecolingua's own definitions, the five reused ontologies, and several other ontologies that had been directly or indirectly referred to by Ecolingua's own definitions or by the definitions of the reused ontologies. We show how we re-engineered this specification into a smaller and better Prolog specification.

2.1.2 Approaches and Experiences on Ontology-Driven Knowledge Reuse

(Pinto and Martins, 2000) discerns two kinds of reuse of ontologies in the construction of other ontologies: *ontologies merging*, where different ontologies on the same or similar subjects are merged into a single unified ontology; and, *ontologies integration*, where an ontology is built by aggregating or modifying (adapting, specialising, augmenting, etc.) other ontologies which become distinguishable parts of the integrated ontology.

Ontology reuse as in (Swartout et al., 1996) is an example of ontologies merging according to (Pinto and Martins, 2000). In this work a domain specific ontology for military air campaign planning is constructed through reuse of the SENSUS ontology (Knight and Luk, 1994), a natural-language-based ontology of broad coverage originally developed to support work on machine translation. Firstly, domain specific concepts are linked to the large hierarchy of SENSUS concepts. Then, to narrow down the concepts to those of relevance only, the hierarchy is heuristically pruned (using the same heuristic as we use in pruning Ecolingua's hierarchy — Section 4.2.3), followed by re-introduction of manually selected concepts.

Later developments on the air campaign planning ontology appear in (Valente et al., 1999). Again, existing ontologies, knowledge bases, and formulated theories are reused to extend the ontology. Two existing ontologies are merged to form the aircraft sub-ontology. One other ontology reused is an ontology of time from the Ontolingua Server. The translation facility provided by the Server was used to translate the ontology of time into Loom (MacGregor, 1991), the knowledge representation language of the air campaign planning ontology, but the result was not satisfactory. Extensive manual work was further required to adapt the translation output into a simpler specification that exploited properly Loom's representational and reasoning capabilities.

For the *integration* kind of ontology reuse, when constructing other ontologies, (Pinto and Martins, 2000, 2001) propose a first methodology that applies to specifications of ontologies at the knowledge (or conceptual) level only — issues related to reuse of implementation-level ontologies, such as translation between representation languages, are not considered. The methodology is supposed to be applied as part of a larger ontology building methodology, such as those in (Uschold and Gruninger, 1996; Fernández et al., 1997). It gives general guidelines on choosing, assessing and changing ontologies towards integration, for instance, that designers must ponder on the meaning of the hierarchical relation (e.g., is-a, part-of) between the concepts of an ontology to be integrated and on the consequences of changing it.

Two ontology-building experiences are reported where this integration methodology has been successfully applied, also reusing ontologies from the Ontolingua Server's library. The ontologies involved, including the resulting ontologies, are all specified within the same knowledge-representation system. One experience is on the reuse of the $(KA)^2$ ontology (Benjamins and Fensel, 1998), an ontology of the knowledge acquisition community (researchers, topics, products, etc.), to build the Reference ontology (Arpírez et al., 2000), an ontology of ontologies, known as the ontology yellow pages.

In the other reported experience the Monoatomic Ions ontology (to be part of an Environmental Pollutants ontology — see Section 2.2) is built reusing two other ontologies: the Chemical ontology (Fernández et al., 1999) and the Standard Units ontology which is part of EngMath (Gruber and Olsen, 1994), a family of ontologies created with a view towards enabling reuse and sharing of engineering models.

The process of reusing the Standard Units ontology is detailed in (Gómez-Pérez and Rojas-Amaya, 1999), where it illustrates an *ontological reengineering process* adapted from Chikofsky's *software reengineering process* (Chikofsky and Cross II, 1990). The method fits in the ontologies integration methodology aforementioned as its Integration Operations activity. It comprises three steps, taken in this order: a *reverse engineering* step, where a conceptual model of an existing ontology (being reused) is extracted from its code; a *restructuring* step, where the newly obtained conceptual model of the existing ontology is made more correct and complete with regard to the ontology or application that is going to reuse it; and finally a *forward engineering* step, where the new conceptual model is implemented. During the restructuring step the conceptual Standard Units ontology undergoes several modifications mainly to improve its clarity and extendibility.

We also reuse the EngMath ontologies, including Standard Units, in the construction of Ecolingua, our ontology for description of ecological data. We did not take the approach, though, of redesigning EngMath's conceptual model for reuse. We only selected and reused EngMath's definitions that were useful to our application, rewriting axioms where needed (Chapter 3 and Appendix A).

The EngMath ontologies have been well reused, specially in the construction of implemented ontologies (Borst et al., 1997; Uschold et al., 1998), partly due to their generality (Uschold et al., 1998) and high degree of formality (compared to most other ontologies available for reuse), which means (re)users can rely less on implicit assumptions (Grosso et al., 1998).

In (Uschold et al., 1998) EngMath is not reused to build another ontology; instead a part of it is incorporated into a deployed aircraft design application, developed using Specware, a system for formal specification and development of software. Broadly speaking, the knowledge engineering steps taken in this work included: finding the parts of EngMath, or "kernel", that were relevant to the application; translating the kernel from Ontolingua representation into Slang, the language of Specware, and augmenting the Slang specification with additional concepts not provided by the reused ontology; integrating the Slang specification into the application, refining it to the specific task at hand; and, translation into Lisp, which was automatically done by Specware. Note that in this experiment the translation from Ontolingua to Slang does not use the Ontolingua Server's translator; it is a manual translation from one knowledge-level representation to another. However, this did not stop the authors from encountering difficulties in the translation process, mainly bearing on semantic differences between KIF, which is based on set theory, and Slang, based in turn on categorical type theory. Another source of difficulty was the amount of infrastructure (or meta-ontological definitions), such as the whole of the frame ontology, that Ontolingua definitions carry with them.

From the above experiences and our own (Chapter 4) it is clear that reuse of formal ontologies rises better to the expectations of knowledge sharing when carried out within a single knowledge-representation system. When that is not the case one has to deal with the difficult, largely unresolved problem of reconciling mismatches between the (inevitably biased) representation and inference styles that the distinct representation systems commit to (Valente et al., 1999). Notice that this can also happen when translating ontologies from conceptual-level representation formalisms into implementation languages. As argued in (Uschold et al., 1999), "there is an inherent tradeoff between the need for different expressive capabilities for different purposes [provided by the different formalisms and languages], and the need to share information between systems", and in order to ease translation one ought to compromise on the expressive power of the representation languages.

Clearly, it is by far more common for a single ontology to be reused. In (Gómez-Pérez and Rojas-Amaya, 1999) two ontologies are reused but there is no overlap between them. It should be noted that reusing multiple, overlapping ontologies in one other ontology or application brings about additional challenges, such as handling possible conflicting definitions. The ontologies integration methodology in (Pinto and Martins, 2000, 2001) briefly touches upon the reuse of more than one ontology. It can seemingly be applied by integrating the ontologies one by one. Although a systematic approach, this would certainly lead to redundancy of engineering effort. Also, needless to say, the translation problems we mentioned above would be exacerbated in a scenario of reuse of multiple ontologies specified in distinct formalisms. A method for ontology mapping unrestricted to specific representation formalisms is currently under development (Kalfoglou and Schorlemmer, 2002). To achieve such generality the method grounds itself on channel theory and information flow theory.

The most straightforward way of assembling multiple interrelated ontologies in one integrated ontology is through theory inclusion, the strategy of the Ontolingua Server's translator. A different means for linking ontologies together has been proposed in the form of *ontology projections* (Borst et al., 1996, 1997). These are considered ontologies in their own right which contain formalisations of mappings between ontologies that express different viewpoints on a same domain. PhysSys is presented in this work as an example of such an ontology, created for the purpose of engineering physical systems modelling. The ontology projections in PhysSys are between ontologies that conceptualise views of physical systems in terms of *components* — a system is a configuration of components; *processes* — the physical processes that determine the system's behaviour; and *math* — the engineering mathematics that is used to describe the processes. In a nutshell, the projections formalise (through a set of axioms) the multifaceted notion that physical systems consist of "components [which] are carriers of physical processes that are described by mathematics". The EngMath ontology from the Ontolingua Server library is also reused here as the latter ontology. The component and process ontologies in turn are derived from abstract ontologies of mereology, topology and systems theory. All ontologies are specified in the Ontolingua formalism.

PhysSys provides the foundation for the design of a library of reusable model fragments, called Olmeco. The fragments are possible representations of components and their decomposition structures, of descriptions of physical processes, and of mathematical relations, all devised based on PhysSys axioms. The ontological representation of axioms becomes: conceptual database schema to represent components, decomposition structures, and the links between the various types of fragments; bond graphs to represent processes; and equations to represent mathematical relations.

The library is intended for modellers of physical systems who can browse the library, choose fragments and instantiate them to models of specific systems. Using the library enforces an evolutionary modelling style — from the components viewpoint, via the processes viewpoint, through to a mathematical specification of the model which can then be executed by way of numerical simulation. The library is reported to have been successfully used in several large-scale industrial applications, with gains in modelling time and model quality.

The model fragments above, in their design and usage, can be considered an example of reuse of knowledge that is founded in ontologies, where lies the main contribution of the present thesis. Here we have a wider perspective of ontologies as a specification technique that enables knowledge reuse (and sharing, consequently) in general (Neches et al., 1991) — the reuse of ontologies is only one form of this where the reused knowledge consists of the ontologies themselves. In our approach, however, the knowledge reuse process is automated and the foundational ontology plays a more direct role in it. Given ecological data described through the Ecolingua ontology, or metadata, an existing model structure is reused by matching with the metadata to give rise to new models, with ontological constraints being verified along the way. It is possible to reuse models because we have pinpointed a synthesis mechanism that establishes connections between metadata and model structure. So, knowing how to draw a model from metadata enables us to, conversely, find parts of the model that are reusable by trying to fit it against new metadata.

2.2 Environmental Ontologies

There is little research on the intersection between ontologies and ecology, or environmental sciences at large, despite the need for a standard terminology to reconcile conflicts of meaning amongst the multitude of fields — biology, geology, chemistry, law, computing science, etc.— that draw on environmental concepts (Gómez-Pérez and Rojas-Amaya, 1999). In contrast, the cousin discipline of biology, particularly molecular biology, has been enjoying significant interest and developments in this direction[1]. Notably, work has been done on exploring and enhancing current ontology technologies to suit needs characteristic of the domain.

An early molecular biology ontology is EcoCyc — Encyclopedia of E. coli Genes and Metabolism (Karp et al., 1996). It comprises a hierarchy of concepts on known genes of E. coli, enzymes and their reactions and metabolic pathways, represented in a standard frame-based system (without other explicit axioms). In the formal ontology of Experimental Molecular Biology (EMB) (Hafner and Fridman, 1996; Noy, 1997; Noy and Hafner, 2000) the frame formalism and other ontological conventions are extended to accommodate representational requirements of the field. For example, most ontologies that involve a concept of tangible objects divide them into either *decomposable objects*, which have distinct parts or components (e.g., cars, other artifacts and organisms), or *stuff* (e.g., water, air), which is distinct from decomposable objects in that every part taken from stuff is still the same stuff. The EMB ontology adds the concept of *mixture* to this classification. Mixtures, common in experimental biology, have properties of both *decomposable objects* and *stuff* — a mixture can be decomposed into the substances it is made of, and every sample from a mixture is still the same mixture. It is argued that features such as this devised as part of the EMB ontology should be relevant to experimental sciences in general (biology, chemistry, physics), and also to other domains like manufacturing, which involve complex substances that interact and undergo processes of transformation.

(Baker et al., 1999) takes the view of an ontology as a service provider in T.O., the ontology of the TAMBIS (Transparent Access to Multiple Biological Information Sources) system. T.O. builds on concepts from an existing ontology of medical terminology and

[1]The specialist events that have been taking place recently are an indication of such interest. For example, there has been five editions of the annual Bio-Ontologies Workshop in conjunction with the International Conference on Intelligent Systems for Molecular Biology (ISMB). A compilation of Bio-Ontologies events maintained by Dr. Robert Stevens at the University of Manchester can be found at http://www.cs.man.ac.uk/˜stevensr/events.html (last accessed on 28 Oct. 2002).

uses the Description Logics language originally developed to specify such ontology. T.O.'s role in the TAMBIS system is to give support to a mediation service: external applications can query for the classification of biological concepts in the ontology, and queries in terms of these concepts can be converted by TAMBIS into requests to access other sources round the world.

As for ecology, even though taxonomies have been in use in the field for a long time, the work started by B.N. Niven (Niven, 1982; Abel and Niven, 1990; Niven, 1992) is the earliest we are aware of on the definition of ecological concepts, specifically within theoretical animal and plant ecology, in the shape of what we call today a formal ontology (i.e., where concepts' interpretations are defined in logic). And her motives where similar to today's for developing a formal ontology: to have a precise definition of the theory with all its assumptions explicitly laid down, so that it can be communicated to others exactly, and so that the discipline is equipped with a framework within which specific models or theories can be constructed. Ecological concepts (e.g., environment, niche, community, etc.) were first specified in first-order logic and later in the z language[2] (Spivey, 1988) that allows for hierarchical encapsulation of concepts.

We can also find some attempts in structuring and precisely defining ecological knowledge as a means activity in the context of projects with wider purposes. Later in this chapter we will see a few examples: concept classes organised in hierarchies (Lorenz et al., 1989; Uhrmacher, 1995) in the context of object-oriented ecological modelling systems in Section 2.3.1, and an ecological modelling ontology in the context of a formal modelling language (Uschold, 1991) in Section 2.3.2.

The work in (Kashyap, 1999) is explicitly about building an ontology of environmental concepts, the EDEN ontology. Like the T.O. biology ontology above it gives support to an agent-brokering system, InfoSleuth (Bayardo et al., 1997), in the task of environmental information retrieval (and update) from distributed resources. It is meant to express at a semantic level concepts that are believed to be embedded in environmental databases — examples of concepts in the paper are 'site' and 'contaminant' in relation to a land contamination database. For this reason, it was not constructed through

[2] Concepts of animal ecology only are formalised in z (Abel and Niven, 1990).

traditional time-consuming knowledge acquisition methods involving domain experts. Rather, the approach was to build on knowledge "hard-coded" on readily available environmental information sources such as database schemas, user queries, data dictionaries and standardised vocabulary. Various techniques, such as reverse engineering of entity-relationship models from databases schemas, are applied in order to abstract an initial ontology from these sources. Domain experts are involved at this point to refine the initial ontology. The ontology is represented as a conceptual graph and does not contain axiomatic knowledge. The ultimate goal, it appears, is to in this way grow EDEN into a large-scale environmental ontology using a variety of data sources. We find it is implausible that disparate data sources could reliably give rise to a coherent and consistent ontology, even if restricted to the few ontological constructs (concept names, attributes/slots, relations) of the representational formalism employed.

There have been efforts in developing an Environmental Pollutants ontology (Gómez-Pérez and Rojas-Amaya, 1999; Pinto, 1999). When finalised it shall include concepts on pollutants of various media (water, air, soil, etc.), methods for detecting pollutants, and pollutants concentrations regulated by environmental legislation. A Monoatomic-Ions sub-ontology (Pinto, 1999) — ions are indicators of pollution — has already been built which reuses an ontology of chemical elements (Fernández et al., 1999) (see Section 2.1).

Ecolingua the ontology we developed, was not intended to encompass a grand theory of ecology. It gives a small contribution towards such a theory, mainly by formally characterising ecological quantities that are typically found in supporting data sets of system dynamics models.

2.3 AI Techniques in Design and Synthesis of Ecological Models

Ecological systems make complex objects of study. We only poorly understand the intricate, dynamic webs of dependencies between their innumerous aspects and components. In trying to follow the common "divide and conquer" strategy, one always risks oversimplifying or misrepresenting because, as opposed to artificial systems, key components and/or interrelations in ecological systems are likely to be unknown. Also, in studying the dynamics of such systems it is virtually impossible to bound all the causes and implications of change occurring in them. For reasons such as these purely analytical methods do not suffice in ecology. Traditional statistical methods, for example, fall short due to inherent difficulties in collecting sound ecological data. Therefore, as (Hilborn and Mangel, 1997) put, "we must rely on observation, inference, good thinking, and models to guide our understanding of the world around us".

Indeed, simulation models in particular are a powerful and widely used tool in environmental studies. Modelling enables ecologists to give shape to and to refine their understanding of complex systems by gradually filling knowledge gaps that are invariably discovered in the process. Once built, models can be used for simulation and prediction of environmental phenomena which can aid decision making in the definition and implementation of environmental policies.

2.3.1 Object-Oriented Modelling

Object orientation has been one of the computing paradigms applied to ecological modelling, probably motivated by the great popularity of this technology since the late 80's.

In (Lorenz et al., 1989) a tree growth model (CROWN) is designed as an object-oriented knowledge base, justified in the assumption that this kind of representation is close to the model conceptions of biological and ecological knowledge. Tree growth is modelled in three levels of abstraction: the *general* level, where generic properties of trees are represented; the *species* level, comprising species-specific properties; and the level of *individual trees*. The inheritance mechanism transfers information from one level of abstraction to the level immediately below and messages activate computation of procedures.

In the same line of work we find EMSY, an object-oriented ecological modelling system (Uhrmacher, 1995). Here it is argued that a modelling system for the environmental domain should support classification of knowledge because humans tend to structure their perception of the environment into classes and subclasses (Berlin, 1978). The emphasis is on representing natural phenomena of change of structure and strategy in ecological system and the new ecological processes that arise from such changes. An ecological system is represented in EMSY as a hierarchy of entity classes, where each entity has associated with it at a time *t*: a set of attributes; a set of rules, which defines changes the entity is subjected to; its composition (or sub-entities); its environment, i.e., the other entities that directly influence the entity or are influenced by it; and its coupling structure, which describes the specific influence relations with the other entities in the environment. The object-oriented inheritance mechanism proves useful, for example, to model environmental structural changes over time caused by (or inherited from) changes in individual entities.

The Object-Oriented (or frame-based) paradigm is present in part of our work, not in the representation of models as in (Lorenz et al., 1989) and (Uhrmacher, 1995) above but in the structuring of Ecolingua (Chapters 3 and Appendix A), an ontology we have engineered through which ecological data is described to support model synthesis. However, Ecolingua's frame-based structure was not a deliberate choice but imposed by the Ontolingua Server. In fact, in ontology design in general it is standard to structure ontological concepts as a class hierarchy.

2.3.2 Logic-Based Modelling

Our two model synthesisers, Synthesis-0 and Synthesis-$\mathcal{R}$ (Chapters 6, 7 and 8) are logic programs, succeeding extensive work on the exploration of logic programming to support the construction of ecological simulation models. Logic programming techniques have been proven useful to support, for example, guidance during model construction (Robertson et al., 1995; Castro, 1999), handling of qualitative information, comparative analysis of ecological knowledge bases, and assistance in model validation (e.g., justification of reasoning steps took by a decision support system to suggest a decision) (Robertson et al., 1995).

An initial motivation for this line of work was to use logic to improve both the rigour and accessibility of ecological modelling (Uschold et al., 1984; Muetzelfeldt et al., 1989). It is believed that ecologists, who usually have little or no programming background, have less difficulty with a declarative logic-based modelling style than with conventional programming-oriented modelling approaches where they are exposed to unwieldy and distracting implementation details (Castro, 1999).

Early work on improving accessibility of computer-based modelling to non-programmers is ECO, an intelligent front-end for ecological system dynamics modelling (Uschold et al., 1984). ECO allows a user to engage with the modelling system in free-form dialogues using ecological terminology. The system then integrates the modeller's input with a built-in knowledge base to produce a model in FORTRAN, which can be executed. The built-in knowledge base is composed of a library of ecological entities (lakes, trees, etc.), a library of ecological processes (respiration, evaporation, etc.), and a library of model modules, each module with associated inputs and outputs indicating its usage (e.g., an equation defining a respiration process, which would be an output, and its input variables such as temperature).

Follow-up works to the above are (Bundy and Uschold, 1989; Uschold, 1990, 1991) resulting in the development of the Elklogic modelling language based on typed lambda calculus. The language was designed to tackle two key hindering factors in model construction in general as argued in (Uschold, 1991), namely, the vast size of the modelling search space and a wide conceptual gap between available tools and the way that users think about their modelling problems.

The representational constructs in Elklogic derive from an 'ecological modelling knowledge ontology' that characterises various kinds of ecological information in four levels of abstraction: general, system-of-interest, user-modelling system dialogue, and runnable model levels. Ecolingua, our ecological metadata ontology, incorporates some of the concepts in the general and system-of-interest levels of abstraction. Elklogic's logical foundation provides the language with the versatility needed to represent knowledge belonging to all four levels of abstraction. In that resides one of the advantages of applying logic to ecological modelling. With a single representational language one can specify general modelling knowledge, information on the specific modelling problem, and the model itself (Robertson et al., 1991). Furthermore, such uniformity has positive repercussions on the resulting models and their construction. Applying Elklogic, for example, led to enhanced consistency between the models and their domain, and to better structuring of the modelling search space (Uschold, 1991).

Elklogic also has interesting inferential features such as induction of set attributes. For example, from the weight attribute of an individual animal it can be inferred that attributes like average, total or maximum weight apply to sets (groups) of the animal. We believe this kind of attribute induction would be useful in automating estimation of parameters of system dynamics models, in a modelling stage posterior to the design of model structure to which our synthesis systems provide automated support (see Section 10.2.4).

The overall assessment in (Uschold, 1991) of the Elklogic language and ELK, the system built on it, is that they mostly support structuring and pruning of the modelling search space but are weak in supporting heuristic guidance through the search space. The primary reason given for this being that the kind of heuristic knowledge needed to achieve so, used by model designers, is difficult to acquire and generalise.

Ecological modelling is a good example of a human activity where design decisions are made under uncertainty. Models are built based on assumptions and guess-estimations as much as they are on factual information. A consensual understanding about how a complex ecological system works is extremely rare. Thus an ecological model can be thought of as the formulation of an argument by the modeller about how they think the system works (Robertson et al., 1991). In such a scenario of incomplete, subjective knowledge, logic lends itself a convenient language for it can represent both whatever expertise is available as well as assumptions and forms of argument (Robertson et al., 1995). Instead of the traditional black-box models, the interpretation of the real-world that the model represents becomes accessible and the chain of reasoning followed to obtain simulation results can be reconstructed.

(Brilhante, 1996) and (Robertson and Agustí, 1999, Chapter 9) take advantage of this capability to make explicit to users sources of uncertainty embedded in simulation models, such as background information on the data that calibrate the model (when, from where and in what conditions the data were collected). Sources of uncertainty are tagged to the logic-based specification of the model, in a style that resembles endorsements in Cohen's theory of endorsements (Cohen, 1985). During simulation, as values for model variables are calculated from other variables, the sources of uncertainty are accordingly propagated, combined, and then presented to the model user. In this way the analysis of simulation results becomes more realistically informed — traditionally simulation results are given with no account of uncertainty in the model. The user can now adjust her level of trust in the model according to the sources of uncertainty present and take that into account in her decision-making.

To date (Robertson et al., 1991) is the most extensive reference in this topic containing a compilation of logic-based approaches and techniques applied to ecological modelling. Our model synthesis systems Synthesis-0 and Synthesis-$\mathcal{R}$ incorporate some of the techniques presented in this work, in particular, the use of abduction to bridge available knowledge (domain knowledge in their work, metadata in ours) and model structure, and meta-interpretation and efficiency-improving techniques (Sections 7.4 and 7.6).

All the references cited in this section lay emphasis on logic-based representation of *ecological modelling expertise* to support automated modelling. It had not yet been examined how representations of *ecological data* could contribute to that. Our work on model synthesis fills this gap. Based on common modelling practices we have designed synthesis heuristics that are able to populate the modelling search space with model components that can be justified in the properties of the data behind the model. This is realised in two synthesis systems which we describe in Chapters 6, 7 and 8.

2.3.3 Model Induction and Equation Discovery

Model Induction, part of the Machine Learning field, is an approach to derived models from data. Figure 2.1 summarises the Machine Learning task (Kubat et al., 1997) viewed from a modelling perspective. Data, together with essential background knowledge, are used as examples from which a general model can be induced (learnt). The models come in a variety of forms such as if-then rules, equations, decision and regression trees.

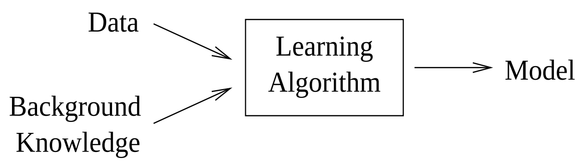


Figure 2.1: The Machine Learning task from a modelling perspective.

This style of knowledge induction is especially attractive for ill-defined or poorly understood problems that lack algorithmic solutions. In other words, given that we fail to formally state the problem and/or analytically solve it and all we have are factual examples, machine learning attempts to extract general concepts from the examples by finding patterns in them. As we mentioned in the beginning of this section, the environmental sciences teem with problems of this nature.

In the series (Kompare et al., 1994; Džeroski et al., 1997) machine learning techniques are used on automated modelling of ecological systems. The first paper discusses early results such as the RETIS system which induces regression trees from data related to algal growth in the Lagoon of Venice (Karalič, 1992). In the second paper in the series the induction algorithm CN2 (Clark and Niblett, 1989) learns if-then rules for classification of water quality of British and Slovenian rivers based on biological and chemical attributes of the water.

Of more interest to us are systems that discover behaviour patterns in ecological data and express them as equations, the so-called equation discovery systems, and in particular systems that induce differential equations as these are closely related to system dynamics models.

Lagrange (Džeroski and Todorovski, 1993, 1995) is one of such systems. It has also been applied to data of the Lagoon of Venice. Its performance was found to be good on synthetic data but poor on actual field data due to the problems of insufficient, inadequate or inaccurate data, which are typical of ecological data sets. The Goldhorn system (Križman et al., 1995) upgrades Lagrange by using numerical integration instead of derivation to reduce sensitivity to noisy data. Also, it induces a smaller number of equations compared to Lagrange. The Lagramge system (Todorovski and Džeroski, 1997) is yet another in this strand of equation discovery systems. The technique it introduces consists of restricting the equation hypothesis space by employing *declarative bias*, i.e., pieces of domain specific knowledge encoded in an easily modifiable form. Lagramge uses context-free grammars prescribing, for example, the composition of common forms of equation in the ecological domain from primitive mathematical operators and functions. The search for equations is then restricted to the space delineated by the grammar. The system outperformed its two predecessors in experiments using data of both artificial and real systems. The latter, though, were systems of limited complexity — the two-poles-on-cart system[3] was the most complex tackled. Moreover, the approach has the drawback of strong reliance on the adequacy of the declarative bias employed. For ecological systems of real-world complexity this is by and large difficult to attain. And even with the declarative bias, Lagramge does not dramatically reduces, in relation to Lagrange and Goldhorn, the large number of spurious or hard-to-interpret equations induced. A survey on this family of equation discovery system applied to ecology can be found in (Džeroski et al., 1999).

In summary, there exist induction algorithms and systems that are capable of extracting models from ecological data but in a smaller and simpler scale than is desirable for problems of real interest in this domain. Real-world ecological databases are inevitably partial, and noisier, larger and more complex than the usual smoothed, single-tabled databases tackled in published model induction experiments.

The considerations on future work in some of these publications point in the direction of further exploring techniques such as inductive logic programming to combine data, which is mostly quantitative, with symbolic, qualitative domain knowledge to provide more guidance to the induction process. Our work on model synthesis shares with this view on use of qualitative knowledge in that we do not use quantitative data as starting point for synthesis but described properties of the data.

[3] A system where the task is to balance two poles, one hinged on top of the other, which in turn are hinged on the top of a cart that moves along a horizontal track.

2.3.4 Compositional Modelling

Automating the construction of models in general is the ambition of the Automated Modelling field. Modelling processes of various disciplines are investigated and the modelling knowledge is explicitly represented, aiming at developing computer tools able to automatically follow modelling principles (Xia and Smith, 1996). Compositional modelling (Falkenhainer and Forbus, 1991) is the leading automated modelling approach. In a nutshell, provided a modelling scenario and a library of model fragments, these systems select fragments from the library to try and compose them into a model that fulfills given requirements. The fragments are often referred to as model 'building blocks',

In (Rickel, 1995; Rickel and Porter, 1997)[4] we find an application of compositional modelling to an ecology-related domain, namely plant physiology, chosen as representative of complex systems. The compositional modeller built, called TRIPEL, takes as input the variables of the physical system (e.g., variables involved in plant photosynthesis), the influences among the variables, domain knowledge from a multipurpose biology knowledge base, and a prediction question with driving condition(s) and variable(s) of interest (e.g., "How would decreasing soil moisture affect a plant's transpiration rate?" — 'decreasing soil moisture' is the driving condition and the 'plant's transpiration rate' is the variable of interest). Given that, the compositional algorithm gives the minimal qualitative model composed of those variables and their inter-influences that can adequately answer the prediction question. Model fragments are available as part of a model of the entire system-of-interest. The algorithm searches for the smallest sub-model that suffices to answer the prediction question posed. It is argued that the system would also be able to build numerical models comprising algebraic and ordinary differential equations.

(Keppens, 2002) presents a novel compositional modelling approach that has been motivated by challenging requirements of automated ecological modelling that existing techniques fail to fulfill. Namely, the non-monotonic nature of modelling reasoning,

[4]We will refer again to this work in Section 6.1.3.3 when looking at our use of influences between system-of-interest variables in relation to other approaches.

modellers' subjective and idiosyncratic preferences for design choices over others, the diversity of model representation formalisms, and model "disaggregation" — e.g., a population partitioned into sub-populations or individuals — which does not correspond to merely adjusting the model's level of detail (or granularity) as in the physical systems domain.

As givens the compositional ecological modeller has a modelling scenario (e.g., dynamics of predator and prey populations), a knowledge base of model fragments, and model fragment preferences. The modelling process starts with the knowledge base being exhaustively instantiated to the scenario, and in this way, a space of all possible partial models is generated. These partial models may then go through a process of disaggregation where variables and/or equations are replaced by corresponding sets of more detailed variables and/or equations. At this stage, the problem of selecting and combining a set of partial models (from the model space) that is consistent and satisfies the modelling assumptions is automatically translated into a dynamic constraint satisfaction problem (Mittal and Falkenhainer, 1990) of the activity constraint type (aDCSP) (Miguel, 2001). These are CSPs extended with special constraints, so called activity constraints, which are able to introduce or remove attributes and their constraints to/from the problem space. The advantage of adopting a CSP approach is that it allows the use of existing, well-studied constraint-satisfaction algorithms, avoiding the development of yet another *ad hoc* compositional modelling algorithm. Finally, user preferences for model fragments are incorporated to form a dynamic preference constraint satisfaction problem (DPCSP). Consequently, the model solutions yielded not only satisfy the standard modelling constraints defined but also take into account preferences of individual users.

As we have seen in the above systems, for compositional modellers to work appropriate model fragments must exist in the first place. But model fragment libraries are bound to be incomplete since it is not possible to anticipate all the fragments that will be needed. (Clark and Porter, 1997) presents a bold approach to tackle this problem of unpredictability of knowledge components that will be required to perform an intelligent task. They propose knowledge base components as abstract patterns of interacting concepts to act as abstract model fragments. On an 'as-needed' basis, domain-specific and/or task-specific knowledge components can be dynamically synthesised from applicable abstract patterns by binding their objects to objects of the domain/task.

Without fragments dynamically constructed, an *a priori* complete and coherent description of them is required. The good news is that because the fragments are composable, if they are at the right level of generality not too many of them are needed to form a wide range of models. But still, as is typical of knowledge modelling in general, acquiring an adequate library of fragments is considered to be the main limitation of compositional modelling (Keppens and Shen, 2001).

In contrast with our approach to model synthesis, we have ecological metadata as initial building blocks; these are mapped into model components — that can be broadly thought of as model fragments — which are assembled into models, again according to relations in the metadata. The knowledge acquisition effort involved here consists only of describing an existing ecological dataset through the terms of our domain-specific ontology, as we explain and exemplify in Chapter 5. Also, the synthesis algorithms do not require a pre-existing super-model, nor an exhaustive set of partial models to search from — models are synthesised incrementally through transformation of metadata evidence into model structure. The algorithms are shown in Chapters 7 and 8.

2.4 An Introduction to System Dynamics Modelling

This thesis concerns synthesis of ecological models of the system dynamics kind. This section provides some initial background on this modelling discipline, to which we will add according to need in Chapters 5 and 6.

System dynamics was first proposed in the early 60's by J. Forrester as a mathematical approach based on information-feedback theory to study the behaviour of industrial systems through modelling and simulation (Forrester, 1961). It was later applied to the modelling of socioeconomic systems as well but became best known for its applications to environmental studies (Haefner, 1996; Grant et al., 1997; Ford, 1999). Very influential to the popularity of system dynamics was the book *The Limits to Growth*

(Meadows et al., 1972) where simulations of a global system model showed that we were heading towards an unsustainable world, given the established rates of food supply and populational and industrial growth.

The defining characteristic of systems dynamics modelling is its emphasis on connections between system components. It considers these connections to be more influential to the dynamics of the system than the components themselves. The goal of model design is to represent such connections appropriately in both scale and scope.

A complete specification of a system dynamics model consists of a *conceptual model* and its corresponding *quantitative model*. The conceptual model is specified as a structural diagram (known as flow diagram or Forrester diagram), such as the one in Figure 2.2, with various types of elements and connections between them. This structure is underlaid by a mathematical specification consisting of a set of ordinary linear and non-linear differential equations. The equations model continuous change in the real world system with rates of change defined as functions of the current state of the system (Robertson et al., 1991). To run or simulate a model means in effect to calculate the changing values of the model's variables through the equations as simulation time progresses. The values of parameters, which are unchanging, and initial values for state variables comprise the settings to start up a simulation. Typically, simulated values of variables are plotted in graphs so that their behaviour, reflecting some dynamic aspect of the system-of-interest, can be observed for exploratory, descriptive or predictive purposes.

Elements and connections that compose system dynamics model diagrams can be of the following types[5]:

State variables are the elements that define the state of the model as it changes over time. Each state variable represents accumulated material in some part or entity of the ecological system.

Flows are both elements and connections. They represent processes that regulate

[5]System dynamics model components are known by a variety of terms other than the ones we use here. Some examples of terms are stock, compartment and level for state variable; rate for flow; auxiliary variable for intermediate variable; exogenous variable for driving variable; and constant for parameter.

transfer of material to and from state variables — an incoming flow into a state variable increases the quantity of material in it; conversely an outgoing flow decreases the quantity of material in the state variable. Flows can occur between state variables, from the outside (i.e., from somewhere in the system beyond the model's scope) to a state variable, or from a state variable to the outside.

Links are connections only, representing directed *information transfer* between pairs of elements in the model (as opposed to *material transfer* represented by flows). A link from an element A to an element B denotes that B is a function of, or is influenced by, A. A model element can be linked in both directions (i.e., can be influence and be influenced by) multiple other elements. The initial element of a link can be of any type but the terminal element can only be a flow or intermediate variable.

Intermediate variables, driving variables and parameters are the remaining types of model elements which are connected to state variables, flows and between themselves through links. For an element to be an intermediate variable it has to be at the end of at least one link, that is it has to be influenced by at least one other element in the model. Driving variables and parameters, on the other hand, are only sources of influence links. Parameters' values remain constant during simulation, while driving variables' values may change due to external factors such as simulation time.

Figure 2.2 shows an abbreviated version of a system dynamics model diagram[6] from (Grant et al., 1997). The model is intended to support management, with a view to increase profit, of a farm pond where fish are periodically stocked and later harvested to be sold. Profit is calculated based on the biomass of fish harvested , which in turn depends on the total biomass of fish accumulated in the pond. The accumulation, or growth, of fish depends on the amount of plants in the pond as fish feed on them, so the dynamics of aquatic plants biomass is also included in the model.

To further illustrate how dependencies between factors in the system-of-interest are represented in the model, let us look at weight of individual fish (intermediate vari-

[6]We will again make reference to the model in Figure 2.2 in several of the forthcoming chapters.

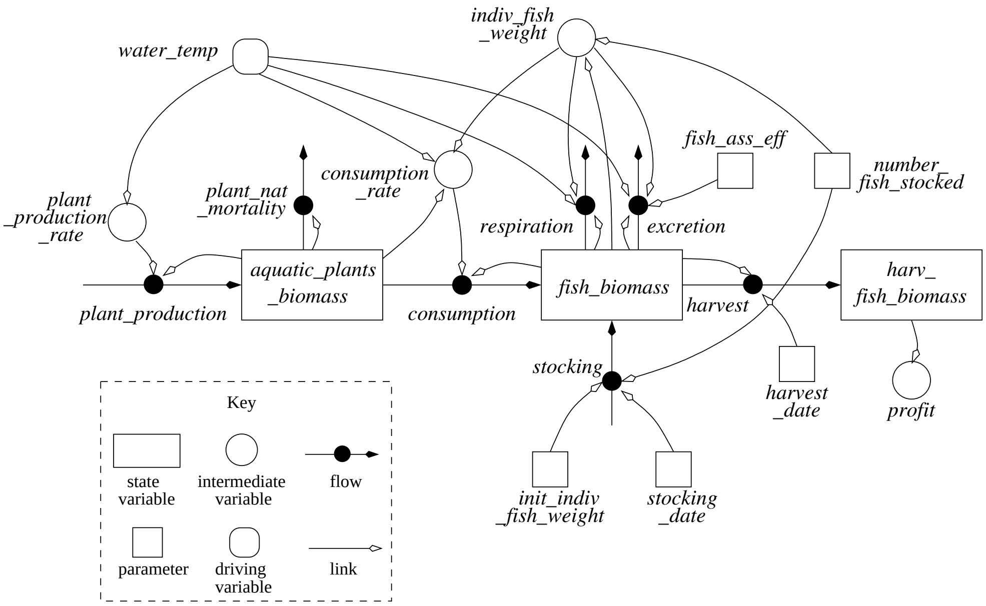


Figure 2.2: System dynamics diagrams exemplified: a fish pond management model.

able *indiv_fish_weight*) and its relationships with other components in the model. The value of this variable changes over time and is calculated as the amount of fish biomass (state variable *fish_biomass*) divided by the number of fish stocked (parameter *number_fish_stocked*). In turn, the weight of individual fish influences the processes of respiration and excretion (flows *respiration* and *excretion*) which decrease fish biomass in the pond. The weight of individual fish also influences the rate of the process of consumption of plants by fish (flow *consumption*) which increases fish biomass in the pond. These three flows plus *stocking* and *harvest* regulate the amount of fish biomass which feeds back into the calculation of the weight of individual fish.

The diagrammatic conceptual models exemplified above are what make system dynamics more appealing in comparison with purely equation-based models. More than just a graphical representation the diagrams provide a rich framework for the quantitative specification of the model. For example, state variable equations are always differential equations with respect to time defined as the sum of the incoming flows to the state variable minus the sum of its outgoing flows; an equation defining a flow (or intermediate variable) must involve all the variables connected to the flow through an incoming link.

This thesis gives techniques for the synthesis of system dynamics models in their conceptual form. In Section 10.2.4 we draw considerations on linking synthesised conceptual models to subsequent quantitative models.

Part II

Ontology Engineering

Chapter 3

The Ecolingua Ontology

In Chapter 1 we defined our task of synthesis of structural system dynamics models informed by properties of ecological data (ecological metadata). Because metadata is a higher-level, more compact representation of data, exploring metadata for synthesis eases the problems associated with inducing model structure from large volumes of data in the style of model induction.

However, a synthesis system of this kind can only be attractive if it is able to work without being restricted to metadata originating from specific data sets. Predefining a detailed knowledge base encompassing properties of every ecological data set is infeasible. Instead we can have a unifying representation language of wide application whose terms are adequate to describe ecological data sets in general.

A language of this kind is an ontology: a shared understanding of some domain of interest, specified in the form of definitions of representational vocabulary and axioms that constrain interpretations over this vocabulary (Uschold and Gruninger, 1996; Neches et al., 1991).

Ecolingua is a domain-specific prototypical ontology we have developed to serve as a vocabulary for property-level descriptions of ecological data from which structures of simulation models can be synthesised. It has been conceptualised within a view of the world as objects, properties of and relations between objects, and formalised in first-order logic. The core of Ecolingua consists of a classification of quantitative ecological data, the main objects in the domain, according to their physical dimension — a fundamental property of all physical quantities.

This generic vocabulary is then employed to describe (or is instantiated to) specific data sets. The description is not done automatically. Human judgement determines the ontological classes and relations that appropriately describe the objects in each data set. Upon a description of a data set, our systems — Synthesis-0 and Synthesis-$\mathcal{R}$ — synthesise structural models with the desirable feature of consistency with the properties of their supporting data.

Ecolingua is a composition of two ontologies: a main ontology of ecological data and a small ontology of ecological model requirements (Figure 3.1).

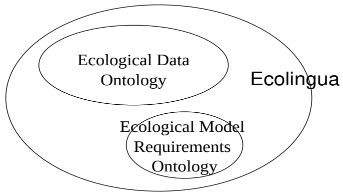


Figure 3.1: Ecolingua's composition: the Ecological Data Ontology and the Model Requirements Ontology.

The two ontologies are presented in Sections 3.2 and 3.3, after Section 3.1 where we comment on the textual and axiomatic forms of the Ecolingua vocabulary definitions. The lower-level relations in Ecolingua are presented in Appendix A with references to their conceptual foundation, the EngMath ontology.

Ecolingua was not itself an aim of this project, but a necessary means for developing and demonstrating the techniques of model synthesis based on metadata, and ontology-enabled reuse of model designs. It is intended as a prototypical, proof-of-principle ontology, as opposed to a finished, deployed ontology for ecological data description.

Reuse of ontologies on Ecolingua's engineering

One of the pursued goals of the knowledge sharing line of research is reusability of ontologies. Libraries of ontologies are already available for reuse on the World Wide Web — e.g., the Ontolingua Server library (Ontolingua Server, 1995); the DARPA Agent Markup Language library (DAML, 2002); the Protégé library (Protégé, 2000)

Motivated by this potential and also by the multidisciplinary nature of ecological data, we designed Ecolingua trying to build on existing ontologies, having chosen as the main reference the extensive ontology library of the Ontolingua Server.

The consideration of Ecolingua's intended usage was another important factor in its design. We share the view that, no different from knowledge engineering endeavours in general, an ontology is more likely to be effectively used when its design is guided not only by the domain but also by the tasks to be performed by the systems that incorporate the ontology (Uschold et al., 1998). An indication of this is general purpose ontologies not yet being practically and widely applied despite long-term efforts such as the Cyc project (Cyc, 1995). In Chapter 4 we report on the difficulties we encountered on tentatively engineering Ecolingua through practical reuse of a number of specific and general ontologies.

3.1 Forms of Concept Definitions

We consider Ecolingua a semi-formal ontology (Uschold and Gruninger, 1996). Most of its concepts have textual and axiomatic definitions, the latter intended to more precisely constrain the interpretation, and thus appropriate uses, of the concepts. With the axiomatic definitions automated proofs are possible. In Synthesis-0 and Synthesis-$\mathcal{R}$ data set descriptions are checked for compliance with Ecolingua. Nevertheless, Ecolingua does not qualify as fully formal in that logical properties such as soundness and completeness are not claimed or proved.

There are two reasons for the lack of axioms for some concepts. First, concepts such as *ecological entity* and *event* have a broad meaning and are inherently hard to constrain. This is why in general ontologies of commonsense knowledge concepts tend not to have explicit axiomatic definitions. The second reason is that Ecolingua is not a mature, polished ontology. Axiomatic definitions should be added and refined as the ontology evolves. For example, a *compatibility* relation is defined between *materials* and *ecological entities*. The relation's present axiom merely constrains the classes of its arguments, but domain knowledge should be formalised to better characterise the relation.

The current Ecolingua axioms are represented as FOL well-formed formulae of the form:

$$Cpt \rightarrow Ctt$$

That is, if Cpt holds then Ctt must hold, where:

- Cpt is an atomic sentence representing an Ecolingua concept.
- Ctt is a sentence (possibly with connectives) that constrains the interpretation of Cpt.

The sentences Cpt make up Ecolingua vocabulary, i.e., the ontology's descriptional terms. One describes an ecological data set by instantiating these sentences. In Chapter 5 we discuss through example the process of describing a data set in Ecolingua.

Most relations in Ctt, the consequents of the axioms, are "internal" in that they are not used as terms to describe data. We call the set of these relations low-level Ecolingua, which consists of a rendition of parts of the EngMath ontology. Low-level Ecolingua is specified in Appendix A.

The Ecolingua vocabulary is defined in the next two sections. In the axioms, variables not explicitly quantified are assumed universally quantified.

3.2 The Ecological Data Ontology

Figure 3.2 shows the class hierarchy of the Ecological Data Ontology, with its two main sections of Contextual Data classes and Quantity classes.

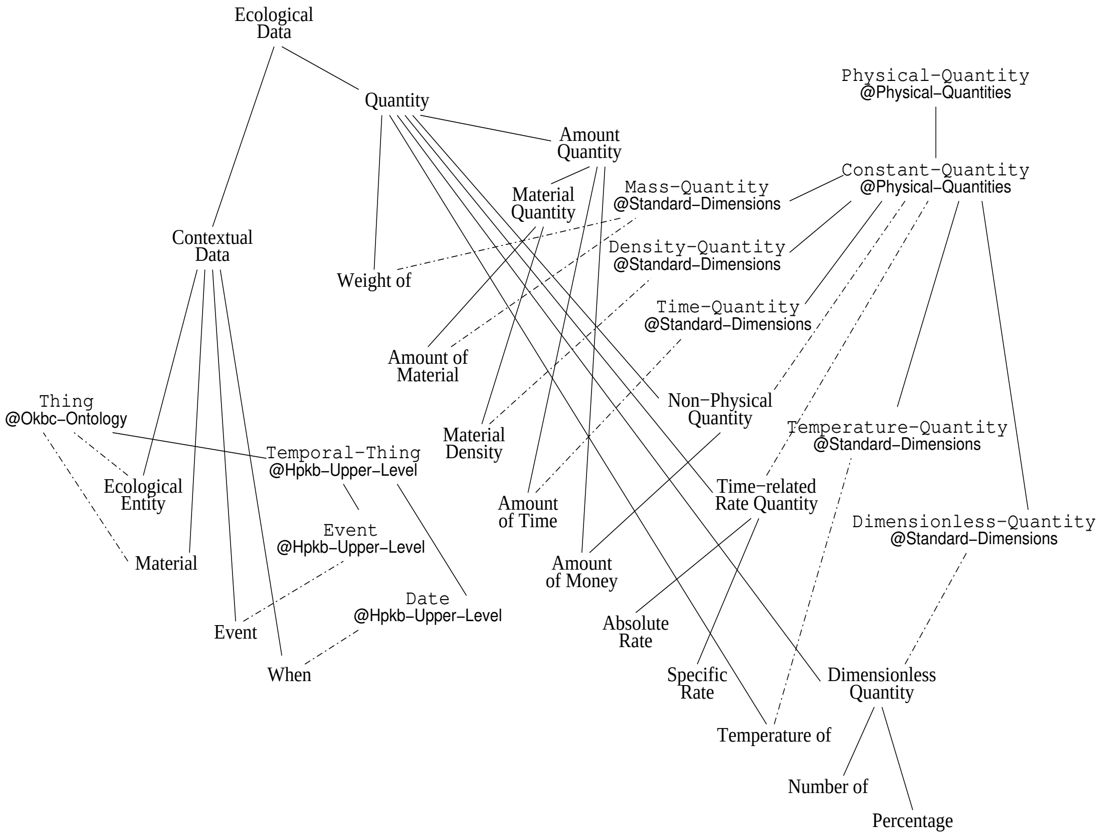


Figure 3.2: Class hierarchy of the Ecological Data Ontology.

The hierarchy comprises Ecolingua classes and external classes defined in other ontologies, namely, Physical-Quantities, Standard-Dimensions, Okbc-Ontology and Hpkb-Upper-Level, all of which are part of the Ontolingua Server's library of ontologies. External classes are denoted `Class@Ontology`, a notation we borrow from the Ontolingua Server.

The Physical-Quantities and Standard-Dimensions ontologies both belong to the Eng-Math family of ontologies (Gruber and Olsen, 1994) on which we comment at various points throughout the chapter. The Okbc-Ontology is the ontology of the generic frame protocol (Chaudhri et al., 1998b). The Hpkb-Upper-Level ontology is a public version of the upper level of the generic ontology Cyc (Lenat and Guha, 1990), containing about 3000 concepts of commonsense knowledge.

The arcs in the hierarchy represent subclass relations, bottom up, e.g., the 'Weight of' class is a subclass of the 'Quantity' class. Ecolingua leaf classes are mutually disjoint, that is, an object can only belong to one of them. We distinguish two different types of subclass relations, indicated by the bold and dashed arcs in Figure 3.2. Bold arcs correspond to full, formal subclass relations. Dashed arcs, on the other hand, correspond to relations between Ecolingua classes and external classes that do not hold beyond the conceptual level, i.e., definitions the (external) class involves are not incorporated by the (Ecolingua) subclass. We call these *referential* subclass relations. In the forthcoming definitions of the Ecolingua classes we refer to textual and axiomatic KIF[1] definitions of their external superclasses as they appear in the Ontolingua Server. In Chapter 4 we examine what led such subclass relations to hold at the conceptual level only.

As mentioned earlier, Ecolingua is not a definitive, complete ontology, therefore Figure 3.2 is far from representing an exhaustive classification of ecological data.

3.2.1 Quantity

The bulk of ecological data consist of numeric values representing measurements of attributes of entities and processes in ecological systems. An ecological data ontology, hence, must contain concepts that capture properties of this kind of data.

The most intrinsic property of a measurement value lies on the physical nature, or dimension, of what the value quantifies (Ellis, 1966). For example, a measure of weight is intrinsically different from a measure of distance because they belong to different physical dimensions, *mass*[2] and *length* respectively. A single physical dimension can characterise measurement values regardless of the measured objects (entities, processes) and units of measure used — grams of algae and gigatons of metal, say, are not intrinsically different. Their common physical dimension is implicit in 'grams' and 'gigatons', both units of the *mass* dimension.

The understanding of this fundamental relation between ecological measurements and

[1] Knowledge Interchange Format (Genesereth and Fikes, 1992).
[2] Or *force*, if rigorously interpreted (see Section 3.2.1.4).

physical dimensions drew our attention towards the EngMath family of ontologies, which provides a well-founded and well-defined conceptualisation of quantities and physical dimensions.

"A physical quantity is a measure of some quantifiable aspect of the modelled world [and differs from] a purely numeric entity like a real number [because it] is characterised by a physical dimension" (Gruber and Olsen, 1994). All defined properties in EngMath's conceptualisation of physical quantities are applicable to ecological measurements:

- every ecological measurement has an intrinsic physical dimension — e.g., vegetation biomass is of the *mass* dimension, the height of a tree is of the *length* dimension;
- the physical dimension of an ecological measurement can be a composition of other dimensions through multiplication and exponentiation to a real power — e.g., the amount of a fertiliser applied to soil every month has the composite dimension $mass/time$;
- ecological measurements can be dimensionless — e.g., number of individuals in a population; and can be non-physical — e.g., profit from a fishing harvest;
- comparisons and algebraic operations (including unit conversion) can be meaningfully applied to ecological measurements, provided that their dimensions are homogeneous — e.g., you could add or compare an amount of some chemical to an amount of biomass (both of of the *mass* dimension), but it would make no sense to add or compare, an amount of biomass to, say, an amount of time.

Also relevant to Ecolingua is the EngMath conceptualisation of units of measure. They are defined as physical quantities like all others, with the same properties. The only thing special about a unit of measure is that it is established by convention as an absolute amount of something to be used as a standard reference for quantities of the same dimension (we give an axiomatisation of units of measure in Appendix A). Therefore, one can identify the physical dimension of a quantity from the unit of measure in which it is expressed (Massey, 1986).

Ecolingua takes advantage of this. Each ecological quantity class is characterised by a physical dimension, which is (automatically) elicited from the unit of measure with which the quantities are specified. In this way, describing a data set in Ecolingua requires no additional effort since it is of course commonplace to have the units of measure of the data specified, whereas the physical dimension of the data is not part of the everyday vocabulary of ecologists.

The magnitude (value) of physical quantities is defined in EngMath as "a binary function that maps a quantity and unit of measure to a numeric value (a dimensionless quantity) ... [for example,] the magnitude of 50 kg in kilogrammes is 50 ... ." Ecolingua is not concerned with magnitudes. As we shall see shortly, numeric values do not have a role to play in Ecolingua's characterisation of quantities in terms of their physical dimensions and properties of the objects they measure. This also excludes unit conversion from Ecolingua's axiomatisation.

EngMath divides physical quantities into constant quantities and function quantities. Constant quantities are mappings of physical things to quantities, whereas function quantities are mappings, with any finite number of arguments, of constant quantities to other constant quantities (e.g., the altitude of a particle at particular times is a unary function quantity). The ontology also draws a distinction between scalars and higher-order tensors. Scalar quantities have size (or magnitude) but not direction, their magnitudes are real numbers and as such have a linear order. Higher-order tensors, such as vectors, cannot be fully characterised by ordered magnitude alone, they require an additional statement such as direction or orientation. The classes of ecological quantities defined in Ecolingua apply to constant, scalar, physical quantities.

In the axioms $Cpt \rightarrow Ctt$ (Section 3.1) that follow defining each type of ecological quantity, the first argument in the *Cpt* atomic sentences represents an identifier, a string constant, for each instance of that quantity type (see examples in Chapter 5 and Appendix B).

3.2.1.1 Amount quantity

Many quantities in ecology represent an amount of something contained (or how much of something there is) in a thing or place, for example, carbon content in leaves, water in a lake, energy stored in an animal's body. Clearly, not exclusive to ecology, amount of something appears to be a commonsense notion having had mention in works on qualitative modelling (Forbus, 1984) and commonsense reasoning (Davis, 1990, Chapter 4).

Material Quantity

Quantities that represent an amount of material things are of the *mass* dimension (intuitively a 'quantity of matter' (Massey, 1986)). For such quantities we define the amount of material class. It is a referential subclass of the `Mass-Quantity`@Standard-Dimensions class (Figure 3.2), defined in the Ontolingua Server as:

$$(\Leftrightarrow\ (\texttt{Quantity.Dimension ?X Mass-Dimension})\ (\texttt{Mass-Quantity?X}))$$

- The **Amount of Material** class — $amt\_of\_mat(A, Mt, E, U)$
  If A identifies a measure of amount of material Mt in E specified in U then Mt is a material, E is an entity which is compatible with Mt, and U is a unit of mass:

$$\begin{aligned}&amt\_of\_mat(A, Mt, E, U) \rightarrow\\&\quad material(Mt)\ \wedge\ eco\_entity(E)\ \wedge\ compatible(Mt, E)\ \wedge\\&\quad mass\_unit(U)\end{aligned}$$

Other quantities represent measurements of amount of material in relation to space, e.g., amount of biomass in a crop acre, or of timber harvested in a hectare of a managed forest. The dimension of such quantities is *mass* over some power of *length*, most commonly area (*length*^{2}) and volume (*length*^{3}). We define the material density class for these quantities. It is a referential subclass of the `Density-Quantity`@Standard-Dimensions class (Figure 3.2) defined in the Ontolingua Server as:

$$\begin{aligned}&(\Leftrightarrow\ (\texttt{Density-Quantity ?X0})\\&\quad(\texttt{Quantity.Dimension ?X0 Density-Dimension}))\end{aligned}$$

However, for our Material Density class the exponent of the length dimension is not restricted to three[3] (volume) as in the Ontolingua Server's definition of the density physical dimension:

$$(= \texttt{Density-Dimension}$$
$$(* \texttt{Mass-Dimension} (\texttt{Expt Length-Dimension} -3)))$$

- The **Material Density** class — $mat\_dens(A, Mt, E, U)$
  If A identifies a measure of density of Mt in E specified in U then Mt is a material, E is an entity which is compatible with Mt, and U is equivalent to an expression Um/Ul, where Um is a unit of mass and Ul is a unit of some power of length:

$$mat\_dens(A, Mt, E, U) \rightarrow$$
$$material(Mt) \wedge eco\_entity(E) \wedge compatible(Mt, E) \wedge$$
$$\exists\, Um, Ul \,.\, eqv\_expr(U, Um/Ul) \wedge mass\_unit(Um) \wedge length^n\_unit(Ul)$$

The *eqv_expr* relation used above and in upcoming axioms holds between expressions that are syntactically identical or between expressions that might be written differently but are algebraically equivalent, for example, the expressions $Um \times (1/Ul)$ or $Ul^{-1} \times Um$ are equivalent to Um/Ul.

Amount of time

Quantities can also represent amounts of immaterial things, time being a common example. The duration of a sampling campaign and the gestation period of females of a species are examples of ecological quantities of the amount of time class. The class is a referential subclass of the `Time-Quantity`@Standard-Dimensions class (Figure 3.2) defined in the Ontolingua Server as "conceptually, a time-quantity is an amount (duration) of time ...":

$$(\Leftrightarrow (\texttt{Quantity.Dimension ?X Time-Dimension}) (\texttt{Time-Quantity ?X}))$$

[3] See also Appendix A.

- The **Amount of Time** class — $amt\_of\_time(A,Ev,U)$
  If A identifies a measure of an amount of time of Ev specified in U then Ev is an event and U is a unit of time:

$$\begin{aligned} & amt\_of\_time(A,Ev,U) \rightarrow \\ & \quad event(Ev) \ \wedge \\ & \quad time\_unit(U) \end{aligned}$$

Non-Physical Quantity

Despite the name, the 'physical quantity' concept in EngMath allows for so-called non-physical quantities. These would be quantities of new or non-standard dimensions, which can be defined as long as all the properties of physical quantities, as already defined in the ontology, apply to them. That is, the quantities have an (new or non-standard) intrinsic dimension which makes them comparable and which is amenable to algebraic operations and combination with other dimensions. (Gruber and Olsen, 1994) gives a monetary dimension as an example of a valid non-physical dimension: "one can accumulate sums of money, do currency conversion, compare relative wealth" and one can combine amount of money with time, for example, to give a rate of inflation. Can you think of other examples of new dimensions?

The class of non-physical quantities is a referential subclass of `Constant-Quantity`@Physical-Quantities (Figure 3.2) defined in the Ontolingua Server. We quote here only an excerpt of the natural-language definition of the `Constant-Quantity` class in the Server (see Section 3.2.1): "A `Constant-Quantity` is a constant value of some `Physical-Quantity`, like 3 meters or 55 miles per hour. Constant quantities are distinguished from function quantities, which map some quantities to other quantities."

Ecological data often involve measurements of money concerning some economical aspect of the system-of-interest, e.g., profit given by a managed aquatic or forest system or balance of a cooperative's bank account. We define Amount of Money as a class of non-physical quantities for this kind of data.

- The **Amount of Money** class — $amt\_of\_money(A,E,U)$
  If A identifies a measure of amount of money in E specified in U then E is an entity and U is a unit of money:

$$amt\_of\_money(A,E,U) \rightarrow$$
$$eco\_entity(E) \ \wedge$$
$$money\_unit(U)$$

3.2.1.2 Time-related Rate Quantity

Generally, rates express a quantity in relation to another. In ecology, rates commonly refer to instantaneous measures of processes of movement or transformation of something occurring over time, for example, decay of vegetation biomass every year, consumption of food by an animal each day. Therefore, a class of time-related rate quantities is defined in Ecolingua. The class is also a referential subclass of the `Constant-Quantity`@Physical-Quantities class (Figure 3.2) defined in the Ontolingua Server.

The absolute rate class is for measures of processes where an amount of some material is processed over time. These quantities have a composite dimension of *mass*, $mass/length^n$ or *money* (the dimensions of amount quantities with the exception of time) over the *time* dimension. To the *mass* or $mass/length^n$ dimensions will correspond units of material which we define in Section A.1.

- The **Absolute Rate** class — $abs\_rate(R,Mt,E_{from},E_{to},U)$
  If R identifies a measure of the rate of processing Mt from E_{from} to E_{to} specified in U then Mt is a material, E_{from} and E_{to} are entities which are different from each other and compatible with Mt, and U is equivalent to an expression Ua/Ut, where Ua is a unit of material and Ut is a unit of time:

$$
\begin{aligned}
& abs\_rate(R, Mt, E_{from}, E_{to}, U) \rightarrow \\
& \quad material(Mt) \wedge eco\_entity(E_{from}) \wedge eco\_entity(E_{to}) \wedge \\
& \quad E_{from} \neq E_{to} \wedge compatible(Mt, E_{from}) \wedge compatible(Mt, E_{to}) \wedge \\
& \quad \exists\, Ua, Ut \,.\, eqv\_expr(U, Ua/Ut) \wedge \\
& \qquad\qquad (mat\_unit(Ua) \vee money\_unit(Ua)) \wedge time\_unit(Ut)
\end{aligned}
$$

Sometimes processes are measured in relation to an entity involved in the process. We call these measures specific rates. Let us take the example of food consumption by an animal. Suppose measures of how much food the animal consumes are given in g/day. That would be the absolute rate of consumption. Now, the rate of food consumption in relation to the animal's weight would be a specific rate. A measure given in, say, $g/g/day$ would mean how much food in grams per gram of the animal's weight is consumed per day.

Usually specific rates are expressed as a dimensionless quantity (a unitless number)[4] over time. This representation leaves implicit that the dimensionless quantity is in fact a ratio of two quantities of the same dimension whose units have been cancelled out. This is illustrative of representational issues that can benefit from ontologies. Ecolingua makes this hidden assumption explicit, and in this way ensures that a specific rate's dimension is correctly composed of a ratio of equal dimensions over the time dimension.

We define the specific rate class in relation to the absolute rate class as follows.

- The **Specific Rate** class — $spf\_rate(R, R_{abs}, Mt, U)$
  If R identifies a measure of a specific rate, related to R_{abs}, of processing Mt specified in U then: R_{abs} measures the absolute rate of processing Mt from E_{from} to E_{to} specified in U_{abs}, which is an expression equivalent to Ua/Ut where Ua is a unit of measure of material; and U is equivalent to an expression $Ub/Uc/Ut$ where both Ub and Uc are

[4] We will soon see that in fact EngMath defines the physical dimension of dimensionless quantities as the identity dimension with the identity unit as unit of measure.

units of measure of material and are of the same dimension D:

$$
\begin{aligned}
& spf\_rate(R, R_{abs}, Mt, U) \rightarrow \exists\, E_{from}, E_{to}, U_{abs}, Ua, Ut\ .\\
& \quad abs\_rate(R_{abs}, Mt, E_{from}, E_{to}, U_{abs}) \wedge eqv\_expr(U_{abs}, Ua/Ut) \wedge mat\_unit(Ua) \wedge\\
& \quad \exists\, Ub, Uc, D\ .\ eqv\_expr(U, Ub/Uc/Ut) \wedge mat\_unit(Ub) \wedge mat\_unit(Uc) \wedge\\
& \qquad\qquad unit\_dimension(Ub, D) \wedge unit\_dimension(Uc, D)
\end{aligned}
$$

Note that the axiom does not constrain the units of the absolute and specific rates, U and U_{abs}, to be of the same dimension. One can be a unit of $mass/time$ while the other is a unit of $mass/length^n/time$. However, the dimensions of the two units, Ub and Uc, part of the specific rate's composite unit of measure, must be the same. For example, the absolute rate could be specified in $mg/ha/day$ and the specific rate in $g/kg/day$.

Below is the definition of specific rates of processes measured in units of money per time.

If R identifies a measure of a specific rate, related to R_{abs}, of processing Mt specified in U then: R_{abs} measures the absolute rate of processing Mt from E_{from} to E_{to} specified in U_{abs}, which is an expression equivalent to Ua/Ut where Ua is a unit of measure of money; and U is equivalent to an expression $Ub/Uc/Ut$ where Ub and Uc are also units of measure of money:

$$
\begin{aligned}
& spf\_rate(R, R_{abs}, Mt, U) \rightarrow \exists\, E_{from}, E_{to}, U_{abs}, Ua, Ut\ .\\
& \quad abs\_rate(R_{abs}, Mt, E_{from}, E_{to}, U_{abs}) \wedge eqv\_expr(U_{abs}, Ua/Ut) \wedge money\_unit(Ua) \wedge\\
& \quad \exists\, Ub, Uc\ .\ eqv\_expr(U, Ub/Uc/Ut) \wedge money\_unit(Ub) \wedge money\_unit(Uc)
\end{aligned}
$$

3.2.1.3 Temperature Quantity

Another fundamental physical dimension is *temperature*, which has measurement scales rather than units (more on this in Section A.2.1). Temperature of water in a pond and environment temperature in a green house, are two examples of temperature quantities in ecological data sets. The referential superclass of the class below is

`Temperature-Quantity`@Standard-Dimensions (Figure 3.2), defined in the Ontolingua Server as:

(⇔ (`Temperature-Quantity ?X0`)
(`Quantity.Dimension ?X0 Thermodynamic-Temperature-Dimension`))

- The **Temperature of** class — $temperature\_of(T,E,S)$
If T identifies a measure of the temperature of E specified in S then E is an entity and S is a scale of temperature:

$$temperature\_of(T,E,S) \rightarrow \\ eco\_entity(E) \wedge \\ temperature\_scale(S)$$

3.2.1.4 Weight Quantity

Strictly speaking weight is a *force*, a composite physical dimension of the form $mass \times length \times time^{-2}$. But in ecology, as in many other contexts, people refer to 'weight' meaning a quantity of *mass* in fact. For example, the weight of an animal, the weight of a fishing harvest.

It is in this 'everyday' sense of weight that we define a class of weight quantities. It has `Mass-Quantity`@Standard-Dimensions as referential superclass (Figure 3.2) defined in the Ontolingua Server (see Amount of Material class).

- The **Weight of** class — $weight\_of(W,E,U)$
If W identifies a measure of the weight of E specified in U then E is an entity and U is a unit of mass:

$$weight\_of(W,E,U) \rightarrow \\ eco\_entity(E) \wedge \\ mass\_unit(U)$$

  Note that for quantities of both this class and the Amount of Material class the specified unit must be a unit of *mass*. But the intuition of a measure of weight does not bear a containment relationship between a material and an entity like the intuition of an amount of material does.

3.2.1.5 Dimensionless Quantity

Another paradoxically named concept in the EngMath ontology is that of dimensionless quantities. They do have a physical dimension but it is the identity dimension. Real numbers are an example.

The class of dimensionless quantities has a referential superclass of the same name, `Dimensionless-Quantity@`Physical-Quantities (Figure 3.2), defined in the Ontolingua Server as "although it sounds contradictory, a `Dimensionless-Quantity` is a quantity whose dimension is the `Identity-Dimension`:"

$$
\begin{array}{l}
(\Leftrightarrow \ (\texttt{Dimensionless-Quantity ?X}) \\
\quad (\texttt{And}\ (\texttt{Constant-Quantity ?X}) \\
\qquad (=\ (\texttt{Quantity.Dimension ?X})\ \texttt{Identity-Dimension})))
\end{array}
$$

This concept applies to quantities in ecology that represent counts of things, such as number of individuals in a population or age group. The class below is defined to this kind of quantities.

- The **Number of** class — $number\_of(N,E,U)$
  If N measures the number of E specified in U then E is an entity and N is a dimensionless quantity specified in U:

$$
\begin{array}{l}
number\_of(N,E,U) \rightarrow \\
\quad eco\_entity(E)\ \wedge \\
\quad dimensionless\_qtty(N,U)
\end{array}
$$

Percentages can also be defined as dimensionless quantities. Food assimilation efficiency of a population, mortality and birth rates are examples of ecological quantities expressed as percentages.

- The **Percentage** class — $percentage(P,E,U)$
  If P is a percentage that quantifies an attribute of E specified in U then E is an entity and P is a dimensionless quantity specified in U:

$$
\begin{aligned}
& \mathit{percentage}(P,E,U) \rightarrow \\
& \quad \mathit{eco\_entity}(E) \ \wedge \\
& \quad \mathit{dimensionless\_qtty}(P,U)
\end{aligned}
$$

3.2.1.6 Some other undefined quantity classes

The ecological quantity classes above are not exhaustive. They do not cover quantities of *space*, *energy* or *frequency*, amongst other possible dimensions. Also missing are other forms of density. To classify a population density quantity, for example, we would need a concept of density of objects that can be discretely counted (in contrast to material density).

An extension of Ecolingua should consider definitions of quantity classes such as these.

3.2.1.7 Influence relation and constancy of quantities

Besides quantities themselves, influences that hold (or are suspected to hold) between quantities are also part of ecological data specification. They are denoted in expressions such as ‘A is dependent on B’, ‘A is a function of B’, ‘A increases with B’, etc., and in data tables where values of a quantity A are shown as a function of some other quantity B.

The axiom below defines such relations between quantities.

- The **Influence** relation between quantities — $\mathit{influences}(Q,Q',\mathit{Sign})$
  If Q influences Q′ where the influence is qualified with Sign then Q and Q′ are different quantities of defined classes, the sign Sign of the influence is positive, negative or undetermined, and Q′ is not a constant quantity:

$$
\begin{aligned}
& \mathit{influences}(Q,Q',\mathit{Sign}) \rightarrow \\
& \quad \mathit{eco\_qtty}(Q) \ \wedge \ \mathit{eco\_qtty}(Q') \ \wedge \ Q \neq Q' \ \wedge \ \mathit{Sign} \in \{+,-,?\} \ \wedge \\
& \quad \neg\, \mathit{constant}(Q')
\end{aligned}
$$

A positive influence sign means that the two quantities change in the same direction, e.g., an influence between an absolute rate *plant production*, the influencer quantity, and an amount of material *plants biomass*, the influenced quantity: the more plant production the more biomass, the less plant production the less biomass. The influence is negative when the two quantities change in opposite directions, e.g., influencer *plant mortality* (also an absolute rate) and influenced *plants biomass*. A '?' sign means the direction of change is unknown or unclear, like when there may be threshold values of the quantities where the direction may reverse, e.g., influencer *water temperature* (a temperature quantity) and influenced *plant production*.

The property of constancy of quantities is defined as a contrapositive of the influence relation. If a quantity is constant then it does not suffer influence of any other.

- The **Constancy** property of quantities — $constant(Q)$
  If Q is constant then Q is a quantity of a defined class and there is no other quantity Q' that influences Q:

$$
\begin{aligned}
& constant(Q) \rightarrow \\
& \quad eco\_qtty(Q) \ \wedge \\
& \quad \neg \ \exists \ Q', Sign \ . \ Q' \neq Q \ \wedge \ influences(Q', Q, Sign)
\end{aligned}
$$

3.2.2 Contextual Data

An ecological quantity alone does not tell much if it is not contextualised, if no reference is given on what it quantifies, for example. We define a few, very general concepts of contextual data around the ecological quantities which form the greater part of the ontology. The concepts are the following:

- The **Ecological Entity** class — $eco\_entity(E)$
  Referential subclass of the `Thing`@Okbc-Ontology class (Figure 3.2) of the Ontolingua Server. The class is not axiomatically defined in the Server. We quote here an excerpt of the natural-language definition given: "`Thing` is the class of everything in the universe of discourse that can be in a class. This includes ...

all other objects defined in user ontologies. `Thing` is ... the practical root of all ontologies.” (Also denoted Entity and $\top$ (Sowa, 2000).)

An ecological entity E is any distinguishable thing, natural or artificial, with attributes of interest in an ecological system (e.g., vegetation, water, an animal, a group of individuals or population, machinery), the system itself (e.g., a forest, a lake), or its boundaries (e.g., atmosphere).

Ecological quantities usually consist of measurements of attributes of entities (e.g., carbon content of vegetation, temperature of an animal’s body, birth rate of a population, volume of water in a lake).

- The **Material** class — $material(Mt)$
  Also a referential subclass of the `Thing`@Hpkb-Upper-Level class (Figure 3.2).

  A material Mt is anything that has mass and can be contained in an ecological entity (e.g., biomass, chemicals, timber).

- The **Compatibility** relation between materials and entities — $compatible(Mt,E)$
  A material and entity are compatible if it occurs in nature that the entity contains the material. For example, biomass is only thought of in relation to living entities (plants and animals), not in relation to inorganic things.

  If Mt and E are compatible then Mt is a material and E is an entity:

$$
\begin{aligned}
&compatible(Mt,E) \rightarrow \\
&\qquad material(Mt) \ \wedge \ eco\_entity(E)
\end{aligned}
$$

- The **Event** class — $event(Ev)$
  Referential subclass of the `Event`@Hpkb-upper-level class (Figure 3.2) defined in the Ontolingua Server, also only in natural language, as “[`Event`] is one important subset of `Temporal-Thing` (the collection of all things which have a particular temporal extent, things about which one might sensibly ask ‘When?’). The elements of `Event` are events or actions, things that we say are ‘happening’, changes in the state of the world . . . ”

  An event Ev is any happening of ecological interest with a time duration (e.g., seasons, sampling campaigns, logging or harvest events).

- The **When** class — $when(T)$
  Referential subclass of the `Date`@Hpkb-upper-level class (Figure 3.2) defined in the Ontolingua Server, again only in natural language, as "a `Date` is any `Time-Interval` which can be defined purely by its location on the calendar. Thus a `Date` could be [a particular calendar minute], a particular calendar day, ..., a particular calendar month, a particular decade, etc."

  A when T is any description that identifies a moment or period in time (e.g., 7.55 pm on the 5th Aug. 2002, 1990–2000 decade, the breeding season).

- The **Time of Event** relation — $time\_of\_event(Ev,T)$
  Every event occurs at some moment or period in time (e.g., months of the rainy season, harvest date).

  If a time of event Ev is T then Ev is an event and T is a when description:

$$time\_of\_event(Ev,T) \rightarrow$$
$$event(Ev) \wedge when(T)$$

3.3 The Ecological Model Requirements Ontology

The Ecological Model Requirements Ontology is a set of only three concepts in Ecolingua which are essential for the model synthesis task (Figure 3.3). The concepts' axioms given here link model requirements with the Ecological Data Ontology.

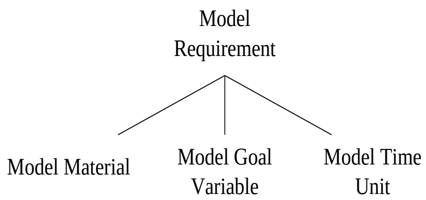


Figure 3.3: Class hierarchy of the Model Requirements Ontology.

Our model synthesis task is concerned with the class of system dynamics models that represent flow of material through ecological systems (e.g., flow of water in a lake basin, flow of a nutrient through a tropical forest, flow of a pollutant through the environment). Materials determine models' modularisation. Models that represent flow of multiple materials are divided into submodels, one to each material (Grant et al., 1997; Haefner, 1996).

- The **Model Material** class — $model\_mat(X)$
  If X is a model material then X is a material:

$$model\_mat(X) \ \rightarrow \ material(X)$$

Certain model variables are of special interest — those whose behaviour over time the model is expect to describe or predict. Such variables can be of any Ecolingua quantity class except Amount of Time for it would be meaningless to simulate the behaviour of an amount of time over time.

- The **Model Goal Variable** class — $model\_goal\_var(Q_{term})$
  If Q_{term} specifies a model goal variable then Q_{term} denotes a quantity of an Ecolingua *class suitable for simulation:*

$$model\_goal\_var(Q_{term}) \ \rightarrow \ sim\_qtty\_class(Q_{term})$$

Finally, every model has a specified unit of time for simulations. The unit should be of a resolution appropriate to the processes that occur in the system-of-interest. A model of forest regeneration, for example, requires a time unit of the order of years since significant changes only occur in this temporal scale.

- The **Model Time Unit** class — $model\_time\_unit(U)$
  If U is a model time unit then U is a unit of time:

$$model\_time\_unit(U) \ \rightarrow \ time\_unit(U)$$

Chapter 4

On the Engineering of Ecolingua

In this chapter we report on the knowledge engineering effort that ultimately led to the Ecolingua ontology presented in the previous chapter and in Appendix A.

Ecolingua was engineered with reuse as the central tenet. In principle, given the task of engineering a new ontology, the easiest way is to reuse existing ontologies whose concepts overlap with those of the new ontology. We had difficulty in trying to adhere to this approach, however, and most of the ontology engineering effort we put in ended up not directly contributing to the Ecolingua needed for metadata-based model synthesis (shown in Chapter 3). Still we believe reporting the experience is worthwhile as it has been a realistic attempt of ontology construction through reuse of multiple ontologies using currently available technology. We are not aware of other reported experience on this particular style of reuse on the same scale.

An informal methodology for ontology construction proposed in (Uschold and Gruninger, 1996) was followed in the experiment. The methodology is summarised below.

1. Definition of purpose and scope;
2. Capture:
    (a) Identification of key concepts and relationships in the domain of interest;
    (b) Production of precise unambiguous (as much as possible) text definitions for the concepts;

   (c) Definition of the meta-ontology — the underlying ontology of representational primitive terms to be used to express the ontology under construction;

   (d) Agreement on the above;

3. Coding;

4. Integration with existing ontologies;

5. Evaluation;

6. Documentation.

For identification of concepts and relations in the domain we took as initial case study a data set generated from a tropical forest logging experiment in the Amazon, Brazil (Biot, 1995). This data set was chosen due to its diversity and large scale which should provide a good coverage of concepts, and also because it has been applied in modelling. It underpinned the construction of a large simulation model used to support decision-making on sustainable logging strategies for forests in that part of the world (Biot et al., 1996). The concepts identified through the data set fell into three grand groups of structural (e.g., data types and composition), spatial (e.g., sampling areas and location), and temporal (e.g., time, duration and frequency of sampling) data properties.

At this stage we started using the Ontolingua Server (Farquhar et al., 1996; Ontolingua Server, 1995). Its extensive library of shareable ontologies was particularly attractive to us in that ecological data has a multidisciplinary nature and we were keen to experiment with the reuse of existing ontologies of the various disciplines concerned.

Chapter outline and usage of fonts

We start with briefly commenting on the conceptualisation of Ecolingua using the Ontolingua Server in Section 4.1. Most of the chapter's content is in Section 4.2 where we explain in detail the steps taken to transform this conceptual specification of Ecolingua into an executable set of Horn clauses. Finally, a discussion of the experiment is given in Section 4.3.

Various fonts are used throughout the chapter to highlight ontologies and ontological definitions. Names of ontologies are written in sans serif (e.g., Ecolingua, Hpkb-

Upper-Level). Classes, relations and their definitions quoted from the Ontolingua Server appear in typewriter font (e.g., `Class`, (`Positive-Integer ?N`)). Prolog syntax expressions translated from KIF by the Server (Section 4.2.1) also appear in typewriter font with atoms that start with a capital letter enclosed in single quotes (e.g., `'Subclass-Of'('Plot','Area')`). Our own logical definitions including those generated by our Horn clause translator (Section 4.2.5) appear in italic (e.g., $unit\_of\_measure(U)$).

4.1 Conceptual Ecolingua in the Ontolingua Server

Figure 4.1 presents an extract from an early conceptualisation of Ecolingua, illustrating the spatial and temporal aspects considered at that stage. The notation is that of conceptual graphs. Boxes represent classes and ellipses represent relations between classes. The classes constrain the type of objects involved in the relations. The directed arcs show the direction of the relation, from domain classes to range classes. For instance, a common sampling strategy in ecological experiments is to replicate different treatments (interventions) on the environment, where each replicate is partitioned into plots; the 'Location' relation locates a quantity 'Sample' (domain class) by giving the 'Replicate', 'Treatment' and 'Plot' (range classes) from where it was sampled. The passing of Ecolingua modelled as a conceptual graph to the Ontolingua Server's frame-based specification style was straightforward.

To give an idea of the size of Ecolingua specified in the Server, excluding reused definitions from other ontologies, it contained 37 classes (including subclasses), 24 relations and 19 functions, at the time when the translation process described below was initiated.

4.2 Translating Conceptual into Executable

The deployment of an ontology through the Server in a form that can be used by some automated reasoning system is a two-fold process. First, a conceptual ontology

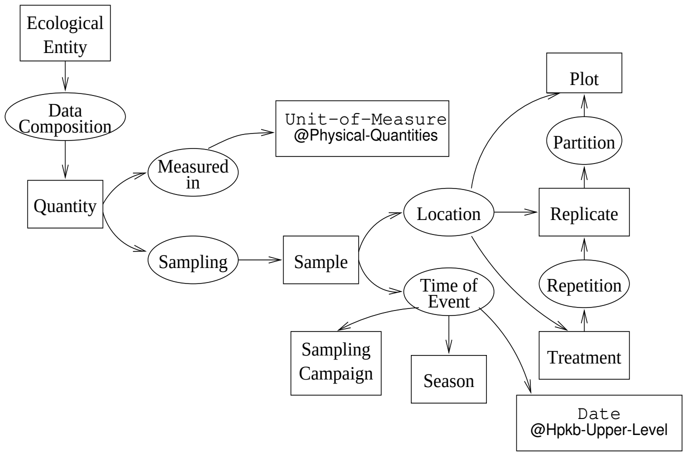


Figure 4.1: Extract from early conceptual Ecolingua.

is created by specifying definitions through the `Class`, `Slot`, `Relation`, `Function`, `Individual` and `Axiom` constructs (which come from the Frame-Ontology). Second, the conceptual ontology is translated into some target implementation language considered adequate to the ontology's intended usage. Having conceptualised Ecolingua in the Ontolingua Server, the next step was then to translate it into Prolog, our choice of implementation language.

Ideally the translation should yield an executable Ecolingua containing:

- Ecolingua-specific definitions, conceptualised for description of ecological data.

- definitions inherited from external ontologies through references made to them in the Ecolingua-specific definitions. Not only the definitions D that Ecolingua directly refers to should be inherited but also the definitions D' that D refers to, and the definitions that D' refers to and so on. If this systematic inclusion of definitions does not take place the translated theory will not be self-contained.

- definitions related to the meta-ontological frame-based primitive terms (`Class`, `Relation`, etc.) imposed by the Server, since these concepts and associated mechanisms are not known to our target language. Explicit clauses are needed through which it can be determined, for example, which are the subclasses of a class, or the properties they acquired through inheritance.

The translation of Ecolingua by the Ontolingua Server into its Prolog syntax target language produced a large specification, far from executable, comprising the union of Ecolingua with the whole of all ontologies it directly and indirectly referred to. For example, the Server's meta-ontology, the Frame-Ontology, includes other four ontologies: Kif-Extensions, Kif-Meta, Kif-Relations. Ecolingua does not explicitly refer to definitions in Frame-Ontology, but because it is the meta-ontology it is added to the translation output together with all its included ontologies.

To re-engineer this output from the Ontolingua Server's translator into a more manageable specification we have implemented additional tools in the form of Prolog programs. They perform three major tasks: clean up of extraneous definitions (e.g. duplicates), pruning of the class hierarchy and related definitions accordingly, and transformation of logical sentences into Horn clauses. The tools can be applied to the translation output as a whole as well as to selected definitions, in which case the latter functionality is of greater relevance.

We now describe a stepwise detailed account of the entire translation process.

4.2.1 Translation by the Ontolingua Server into Prolog Syntax

Conceptual Ecolingua, at the time when this translation work was carried out, referred to selected definitions of five other ontologies: Hpkb-Upper-Level (Cohen et al., 1998), Kif-Numbers, Kif-Extensions (Genesereth and Fikes, 1992), Simple-Time (Ontolingua Server, 1995) and Physical-Quantities (Gruber and Olsen, 1994). The translation of Ecolingua alone into Prolog Syntax produced an 85Kb file, whereas requesting its translation along with the ontologies it referred to produced a 5.3Mb file. Through the steps that follow we re-engineered the 5.3Mb file (hereafter called workfile) in order to turn it into a smaller, better structured, and more manageable set of axioms.

A last remark is that the Ontolingua Server translator did not translate KIF sequence variables, inserting commented lines where they appeared in the translation output. Sequence variables, nonexistent in Prolog, are variables that bind with a finite sequence of objects (e.g., 1 2 3). We did not tackle this in our re-translation either and left the commented lines as they were.

4.2.2 Cleaning up Extraneous Clauses

We call extraneous clauses: over general facts, definitions of self subclasses and duplicated clauses. These clauses were identified and removed from the workfile.

Over General Facts

A fact is defined as over general if all its arguments are uninstantiated variables. These facts were identified by:

$$\begin{aligned} \mathit{overgeneral}(F) \;\leftarrow\; & \mathit{is\_ontolingua\_predicate}(F) \;\wedge\; F \;\wedge\; \mathit{arguments}(F,\, \mathit{Args}) \;\wedge \\ & \forall X \,.\, \mathit{var}(X) \;\leftarrow\; \mathit{member}(X,\mathit{Args}) \end{aligned}$$

There were 59 occurrences of over general facts in the workfile (e.g., $'\mathtt{List}'(\mathtt{X})$, $'\mathtt{Class}'(\mathtt{X})$, $'\mathtt{Relation}'(\mathtt{X})$, $'\mathtt{Set}'(\mathtt{X})$, $'\mathtt{Positive\text{-}Integer}'(\mathtt{X})$ and $'\mathtt{Natural}'(\mathtt{X})$). To illustrate this issue, let us consider in detail a particular clause defining the relation $'\mathtt{Positive\text{-}Integer}'(\mathtt{X})$, which comes from the axioms defining the `Nth-Domain` relation in the Frame-Ontology.

Figure 4.2 shows both the original KIF axioms and their Ontolingua Prolog syntax translation. The axioms refer to domain restrictions generalised to n-ary relations; for instance, the construct (`Nth-Domain Relation 3 Type`) says that the third element of each tuple in the relation `Relation` is an instance of class `Type`.

```
(⇒ (Nth-Domain ?Relation ?N ?Type)
    (⇒ (Member ?Tuple ?Relation)
        (And (≥ (Length ?Tuple) ?N)
            (Instance-Of (Nth ?Tuple ?N) ?Type))))

(Positive-Integer ?N)
(Relation ?Relation)
(Nth-Domain Nth-Domain 3 Class)
(Nth-Domain Nth-Domain 2 Positive-Integer)
(Nth-Domain Nth-Domain 1 Relation)
```

↦

```
⇒ 'Nth-Domain'(Relation,N,Type),
    ⇒ ('Member'(Tuple,Relation),
        & (≥ ('Length'(Tuple),N),
            'Instance-Of'('Nth'(Tuple,N),Type)))).

'Positive-Integer'(N).
'Relation'(Relation).
'Nth-Domain'('Nth-Domain',1,'Relation').
'Nth-Domain'('Nth-Domain',2,'Positive-Integer').
'Nth-Domain'('Nth-Domain',3,'Class').
```

Figure 4.2: KIF axioms (left) and their Ontolingua Prolog syntax translation (right).

The axioms (`Positive-Integer ?N`) and (`Relation ?Relation`) define the classes of objects that instantiate variables `?N` and `?Relation`. There is a notion of frame scope for the variables here. An instantiation to `?N` or `?Relation` will be shared across the axioms in the frame. This is what causes the over general facts in the Prolog syntax version. In Prolog, the scope of a variable does not go beyond the clause where it appears. Thus the variable `N` in the implication axiom and the other variable `N` in the `'Positive-Integer'(N)` axiom do not share. Any term can match `N` in the Prolog axiom `'Positive-Integer'(N)`, leading to unwanted, yet logically sound instantiations.

Self Subclasses

The Ontolingua Server does not allow users to define circular subclasses/superclasses. On attempting to define such a relation, an error message "Cannot have a circular superclass/subclass graph." is displayed. However, the query `'Subclass-Of'(X,X)` over the workfile succeeds for twelve instances of `X`, all of them classes of the Hpkb-Upper-Level ontology (e.g., `'Subclass-Of'('Locomotion-Process','Locomotion-Process')`).

Duplicated Clauses

Duplicated clauses amounted to 57Kb of the workfile.

4.2.3 Pruning the Class Hierarchy

Every time a class is created in the Server it must be defined as a subclass of some other existent class through the `Subclass-Of` relation. The `Subclass-Of` relation is defined in the Okbc-Ontology (Chaudhri et al., 1998a) as "a class C is a subclass of parent class P if and only if every instance of C is also an instance of P", with the corresponding KIF axiom:

```
(⇔ (Subclass-Of ?Child-Class ?Parent-Class)
   (Forall (?Instance)
          (⇒ (Instance-Of ?Instance ?Child-Class)
             (Instance-Of ?Instance ?Parent-Class))))
```

The parent class may be chosen among the classes of the end-ontology being built or from the classes of any of the ontologies in the Server's library (henceforth called external classes). There were eighteen classes in Ecolingua with external parent classes. For most of them we were able to find conceptually appropriate parent classes, e.g., the parent class `Area` of the Hpkb-Upper-Level ontology for the Ecolingua class `Plot`. For the remaining classes we used the catchall class `Thing` of the Okbc-Ontology.

The chosen parent classes in turn will also have been defined as a subclass of some other class, except for the root classes (i.e., parent-less classes). An end-ontology class cannot be a root class since when created they must be defined as a subclass of some other class. Ultimately the hierarchy of classes is completed with the end-ontology classes as leaf nodes and external classes as root nodes. The resulting class hierarchy of Ecolingua was an acyclic graph structure (or forest) containing 1758 classes altogether, 1721 of which were external.

All classes can be found as the set of all nodes that take part in the `Subclass-Of` relation:

$$S_{Nodes} = \{Node \mid \exists\ Child, Parent\ .\ subclass\_of(Child, Node) \vee subclass\_of(Node, Parent)\}$$

And the roots in the forest are the nodes that have no parent:

$$S_{Roots} = \{Node \mid \exists\ Child\ .\ subclass\_of(Child, Node) \wedge \neg\ \exists\ Parent\ .\ subclass\_of(Node, Parent)\}$$

S_{Roots} contained eight elements: `Thing` of Okbc-Ontology; `Term`, `Expression` and `Operator` of KIF-Meta; `Set` and `Bounded` of KIF-Sets; `List` of KIF-Lists; and `Number` of KIF-Numbers. The largest tree by far is the tree under `Thing` containing 1681 classes. The `Thing`, `Set`, `Bounded` and `Number` trees were connected encompassing all but 47 of the classes, which were nodes in the four other disjoint trees with roots `Term`, `Expression`, `Operator` and `List`. In sum, the topology of the class hierarchy was a forest of five disjoint trees, one of them containing approximately 90% of the classes.

Figure 4.3 shows an extract of the class hierarchy. It illustrates the myriad of classes included in the hierarchy, most of them of unlikely practical relevance. The relation $'\texttt{Subclass-Of}'('\texttt{Plot}','\texttt{Area}')$, for example, being `Plot` an Ecolingua class and `Area`

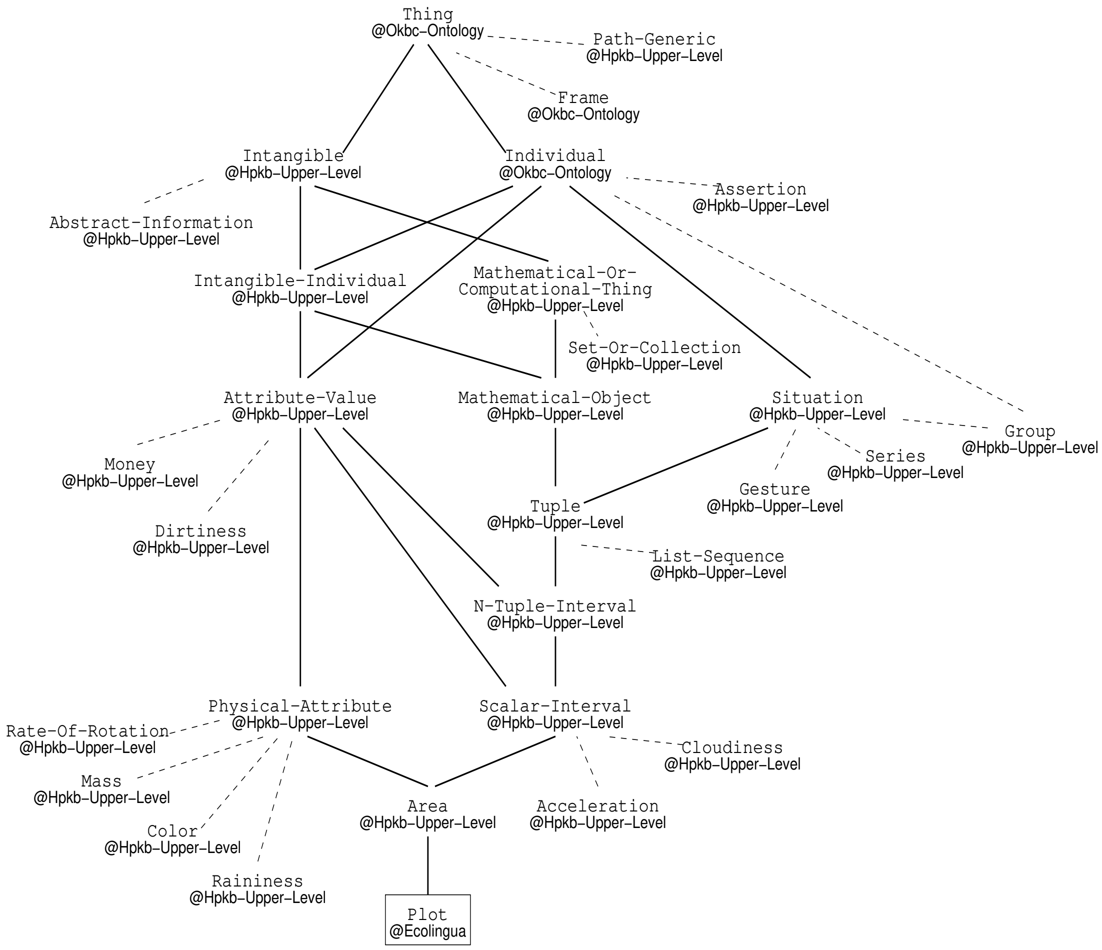


Figure 4.3: Extract of Ecolingua's class hierarchy in the Ontolingua Server and pruning paths.

a Hpkb-Upper-Level class, causes all Hpkb-Upper-Level classes to be included in the hierarchy.

We pruned the hierarchy by traversing it bottom-up, i.e. from the leaf nodes, which are Ecolingua classes, up to the root nodes.

S_P is the set of pruned classes, the ones we keep, if S_{Eco} is the set of the end-ontology classes and S_P is connected to S_{Eco}:

$$pruned\_classes(S_P) \leftarrow S_{Eco} = \{Class \mid ecolingua\_class(Class)\} \wedge connected(S_P, S_{Eco})$$

S_P is connected to S_{Eco} if it is the set of classes that are in the paths from the end-ontology classes to the forest roots:

$$\begin{aligned}&connected(S_P, S_{Eco}) \leftarrow \\ &\quad S_P = \{Class \mid \exists\ Eco\_class\ .\ Eco\_class \in S_{Eco}\ \wedge\ in\_path(Eco\_class, Class)\}\end{aligned}$$

A class is in its path to a root:

$$in\_path(Class, Class)$$

And so is any ancestor of the class:

$$in\_path(Class, Aclass) \leftarrow subclass\_of(Class, Pclass) \wedge in\_path(Pclass, Aclass)$$

This pruning method reduced the class hierarchy from 1758 classes to only 76.

The bold arcs in Figure 4.3 represent the paths from `Plot`, an end-ontology class, to `Thing`, a root class. All classes in these paths were kept. The classes with dashed arcs to their parent class exemplify classes that were pruned off. They are not in the paths from `Plot` to `Thing` (and are not in the paths from any of the end-ontology classes to any of the root classes).

Despite reducing the hierarchy size dramatically, the method does not guarantee that all and only classes of interest (conceptually speaking) are kept. It is reasonable to expect that some specifications, such as slots and axioms, in the definitions of classes will be useful for a reasoning system using the resulting ontology. For instance, through the path `Plot` — `Area` — `Scalar-Interval` Ecolingua inherits the slot `Absolute-Value-Fn`, an Hpkb-Upper-Level unary function that returns the absolute value of its argument with value type `Scalar-Interval`. The function can be used to ensure that the area of a plot is given as a positive value. However, as we move higher along the paths, away from `Plot`, the class definitions become more and more remotely related to the concept of plot, rendering unlikely their practical reuse.

At the same time the method may end up pruning off classes of interest. This is the case of some subclasses of `Physical-Attribute`, such as `Raininess`, `Color` and `Mass`, which are desirable as part of Ecolingua's universe of discourse.

Only an inspection of the class hierarchy by the ontology designer could guarantee that all and only classes of interest were maintained, but this was obviously impractical given the size of the hierarchy.

4.2.4 Pruning Clauses

The objective of this step was to prune the workfile, keeping only clauses that were relevant. The relevance criteria used is empirical, based on our familiarity with Ontolingua relations. The method applied reduced the workfile down to 1.4Mb. The method starts by asserting an initial set of relevant classes, relations and functions and proceeds by selecting clauses that are related to this initial set.

The initial set of relevant classes was defined as

- all Ecolingua classes and their ancestor classes, i.e., the classes in the set, S_P, of pruned classes (Section 4.2.3); and
- the classes `Class` (Okbc-Ontology), `Relation`, `Binary-Relation` and `Function` (Kif-Relations), as these are the meta-ontology classes of which the ontological constructs are instances.

As for the initial set of relevant relations and functions, it consisted of all Ecolingua relations and functions.

The method checks every clause in the workfile for relevance, with clauses of certain forms treated specifically. These specific checks first select the clauses that specify relevant instances of the `Class`, `Relation`, `Individual` and `Axiom` constructs (checks 1 to 5 below), and then the clauses that specify `Subclass-Of` and `Instance-Of` relations involving them (checks 6 and 7). Clauses of other forms are relevant if they involve relevant classes or relations (check 8). Below we explain the relevance checks for each of these specific forms of clauses as well as the general case.

1. **Clauses** $'\texttt{Class}'(\texttt{C})$
   A clause $'\texttt{Class}'(\texttt{C})$ is relevant if `C` is one of the initial relevant classes:

$$relevant(class(C)) \leftarrow C \in S_P \vee C \in \{class, relation, binary\_relation, function\}$$

Or if there is a relevant class, relation or function, T, which is an instance of class C:

$$relevant(class(C)) \leftarrow$$
$$instance\_of(T,C) \wedge$$
$$(relevant(class(T)) \vee relevant(relation(T)) \vee relevant(function(T)))$$

$'\texttt{Instance-Of}'(\texttt{T},\texttt{C})$ is an Okbc-Ontology relation that holds for every ontological term defined. Every class, relation or function is an instance of some class[1]. For example, the Hpkb-Upper-Level relation `Adjacent-To` is an instance of the classes `Relation` (Kif-Relations) and `Spatial-Predicate` (Hpkb-Upper-Level).

2. **Clauses** $'\texttt{Relation}'(\texttt{R})$ **and** $'\texttt{Function}'(\texttt{R})$
A clause $'\texttt{Relation}'(\texttt{R})$ or $'\texttt{Function}'(\texttt{R})$ is relevant if `R` is one of the initial relevant relations or functions, i.e., if `R` is an Ecolingua relation or function:

$$relevant(\mathcal{P}(R)) \leftarrow \mathcal{P} \in \{function, relation\} \wedge ecolingua\_relation(R)$$

In addition to the clauses above were also kept $'\texttt{Relation}'(\texttt{R})$ and $'\texttt{Function}'(\texttt{R})$ clauses where `R` is a relation involving relevant external classes. (Hereafter we use the generic term ‘relation’ to refer to both relations and functions.)

Relations in Ontolingua are used to describe relationships between n terms, with $n \geq 2$. $'\texttt{Domain}'(\texttt{R},\texttt{C})$ — a domain of relation `R` is class `C` — a relation of Okbc-Ontology, specifies the classes to which each of the $n-1$ terms of a relation belong. Each of the $n-1$ terms of a relation must be an instance of its domain class. For instance, the Ecolingua function $'\texttt{Domain}'('\texttt{Number-Of-Replicates}','\texttt{Treatment}')$ specifies that $'\texttt{Number-Of-Replicates}'$ applies to objects of the class $'\texttt{Treatment}'$ (also in Ecolingua). The counter-relation to $'\texttt{Domain}'(\texttt{R},\texttt{C})$ is $'\texttt{Range}'(\texttt{R},\texttt{C})$ which specifies the class of the relation’s n^{th} term.

Continuing with the pruning method, we also kept $'\texttt{Relation}'(\texttt{R})$ and $'\texttt{Function}'(\texttt{R})$ clauses of relations `R` that had at least one relevant domain class.

[1] ‘Instance’ here should not be confused with the Ontolingua construct `Individual`. In Ontolingua, a class can be an instance (of the class `Class`), but a class cannot be an individual.

Relevant range classes, on the other hand, did not count for relevant relations. The intuition behind this heuristic is that a relation with a relevant domain class has a stronger bond with the hierarchy of relevant classes. Relations that have relevant range classes only are considered outsider relations in that they just *end* in a relevant class. Thus:

$$relevant(\mathcal{P}(R)) \leftarrow \mathcal{P} \in \{function, relation\} \wedge domain(R,C) \wedge relevant(class(C))$$

3. **Clauses** $'\texttt{Inverse-Of}'(\texttt{R1},\texttt{R2})$

   $'\texttt{Inverse-Of}'(\texttt{R1},\texttt{R2})$ is a Kif-Relations function that applies over binary relations which are equivalent when their arguments are swapped. For instance, the Simple-Time relations $'\texttt{After}'(\texttt{T1},\texttt{T2})$ and $'\texttt{Before}'(\texttt{T2},\texttt{T1})$, where T1 and T2 are time ranges, are equivalent — $'\texttt{Inverse-Of}'('\texttt{After}','\texttt{Before}')$ holds.

   These clauses were kept when involving a relevant relation, for the sake of retaining in the set of clauses the knowledge of the inverse relation of relevant relations. Thus, a clause $'\texttt{Inverse-Of}'(\texttt{R1},\texttt{R2})$ is relevant if any of R1 or R2 is a relevant relation:

$$relevant(inverse\_of(R1,R2)) \leftarrow relevant(relation(R1)) \vee relevant(relation(R2))$$

4. **Clauses specifying Individuals**

   Clauses specifying individuals appear in the workfile as ground unary predicates $\mathcal{C}(\texttt{I})$, where I is an individual and $\mathcal{C}$ is the class of which I is a member. For instance, $'\texttt{Forest-Logging-Disturbance-Class}'('\texttt{Clearing-Edge}')$ (Ecolingua) denotes that $'\texttt{Clearing-Edge}'$ is an individual of the class $'\texttt{Forest-Logging-Disturbance-Class}'$.

   The pruning method keeps the clauses that specify individuals of relevant classes excluding the meta-ontology classes Class, Relation, Binary-Relation and Function. The exclusion is necessary because there exists a clause of the form $'\texttt{Class}'(\texttt{T})$, $'\texttt{Relation}'(\texttt{T})$, $'\texttt{Binary-Relation}'(\texttt{T})$ or $'\texttt{Function}'(\texttt{T})$ for every class, relation or function T. Thus:

$$relevant(\mathcal{C}(I)) \leftarrow \mathcal{C} \notin \{class, relation, binary\_relation, function\} \wedge relevant(class(\mathcal{C}))$$

5. **Clauses specifying Logical Sentences**

   The Server's translator into Prolog syntax writes axioms as clauses with the predicates: $'\mathtt{Exists}'$, representing the existential quantifier; $'\mathtt{Forall}'$, for the universal quantifier; $\Rightarrow$, for implication; $\Leftarrow$, for reverse implication; $\Leftrightarrow$, for equivalence; and $-$, for negation.

   All logical sentences that involved relevant classes, relations or functions were kept. Names of classes, relations and functions appear in the clauses as functors of the arguments of the logical operator predicates, for instance, $\Leftrightarrow('\mathtt{Individual}'(\mathtt{A}), -('\mathtt{Set}'(\mathtt{A})))$.

   Thus, a logical sentence clause is relevant if any of its arguments contains a functor which is a relevant class, relation or function:

   $$\begin{aligned} & relevant(Lclause) \leftarrow \\ & \quad predicate\_name(Lclause, P) \wedge P \in \{exists, forall, \Rightarrow, \Leftarrow, \Leftrightarrow, -\} \wedge \\ & \quad arguments(Lclause, Args) \wedge contains\_functor(Args, F) \wedge \\ & \quad (relevant(class(F)) \vee relevant(relation(F)) \vee relevant(function(F))) \end{aligned}$$

6. **Clauses** $'\mathtt{Subclass\text{-}Of}'(\mathtt{C1}, \mathtt{C2})$

   The $'\mathtt{Subclass\text{-}Of}'(\mathtt{C1}, \mathtt{C2})$ clauses give shape to the class hierarchy. All these clauses where at least one of the classes was relevant were kept to allow for inheritance to occur.

   A clause $'\mathtt{Subclass\text{-}Of}'(\mathtt{C1}, \mathtt{C2})$ is relevant if any of $\mathtt{C1}$ or $\mathtt{C2}$ is a relevant class:

   $$relevant(subclass\_of(C1, C2)) \leftarrow relevant(class(C1)) \vee relevant(class(C2))$$

7. **Clauses** $'\mathtt{Instance\text{-}Of}'(\mathtt{T}, \mathtt{C})$

   A clause $'\mathtt{Instance\text{-}Of}'(\mathtt{T}, \mathtt{C})$ is relevant if the term $\mathtt{T}$ is a relevant class, relation or function:

   $$\begin{aligned} & relevant(instance\_of(T, C)) \leftarrow \\ & \quad relevant(class(T)) \vee relevant(relation(T)) \vee relevant(function(T)) \end{aligned}$$

   Note that checking relevance of these clauses on the grounds that $\mathtt{C}$ is a relevant class would have no pruning effect. The meta-ontology classes `Class`, `Relation`, `Binary-Relation` and `Function` are relevant classes, and to every term `T` which is a class exists a clause $'\texttt{Instance-Of}'(\texttt{T},\texttt{Class})$, to every term `T` which is a relation exists a clause $'\texttt{Instance-Of}'(\texttt{T},\texttt{Relation})$, and so on.

8. **Other clauses**

   This relevance check applies to any clause that do not fall into the cases above, i.e., clauses with predicates outside the set:

$$S_{Pred} = \left\{ \begin{array}{l} class, relation, function, subclass\_of, instance\_of, inverse, \\ exists, forall, \Rightarrow, \Leftarrow, \Leftrightarrow, - \end{array} \right\}$$

   From checks 1 and 2 above, the $'\texttt{Class}'(\texttt{C})$ clause of domain classes `C` of relevant relations are kept in the pruned set of clauses, and subsequent checks guarantee that other clauses that involve these domain classes are kept as well. Complementary to that, clauses that contain as a subterm the range class of a relevant relation should also be kept.

   Thus, a clause whose predicate is not in S_{Pred} is relevant if it has a subterm which is the range class of a relevant relation or function:

$$\begin{array}{l} relevant(Clause) \leftarrow \\ \quad predicate\_name(Clause, P) \wedge P \notin S_{Pred} \wedge \\ \quad (relevant(relation(R)) \vee relevant(function(R))) \wedge \\ \quad range(R, Class) \wedge subterm(Clause, Class) \end{array}$$

   And finally the most general case. A clause whose predicate is not in S_{Pred} is relevant if it has a subterm which is a relevant class, relation or function:

$$\begin{array}{l} relevant(Clause) \leftarrow \\ \quad predicate\_name(Clause, P) \wedge P \notin S_{Pred} \wedge subterm(Clause, T) \wedge \\ \quad (relevant(class(T)) \vee relevant(relation(T)) \vee relevant(function(T))) \end{array}$$

4.2.5 Transforming Logical Sentences into Horn Clauses

At this stage we have a much smaller set of clauses (5.3Mb down to 1.4Mb) but these still are not in the form required for the computations we wish to perform in Prolog. The good news is that since KIF and Prolog share a common logical parentage, standard truth-preserving transformations from first order predicate calculus to Horn clauses (see for example (Lloyd, 1993)) can be used to convert many of the expressions in the set to exactly the form required. To illustrate this, let us take one of the axioms from the definition of `Unit-Of-Measure`, a class of the Physical-Quantities ontology. The axiom is:

$$(\Rightarrow\ (\texttt{And}\ (\texttt{Unit-of-Measure ?U})\ (\texttt{Real-Number ?N}))\ (\texttt{Unit-of-Measure}\ (\texttt{Expt ?U ?N})))$$

After applying the Ontolingua Server's translator followed by our Horn clauses translator, which implements the standard transformations we mentioned, we obtain:

$$unit\_of\_measure(expt(U,N)) \leftarrow unit\_of\_measure(U) \wedge real\_number(N)$$

This is an example of automated translation where the outcome can directly be incorporated into the target language specification. The expression above corresponds to one of the clauses of our $unit\_of\_measure/1$ predicate specification in low-level Ecolingua (see Section A.2).

The bad news is that not all KIF expressions are translated into an elegant computational form by this means. An example is the expression $(\Leftrightarrow (\texttt{A ?X})\ (\texttt{Or}\ (\texttt{B ?X})\ (\texttt{C ?X})))$. As a first stage translation we can obtain the equivalent set of expressions $\{a(X) \leftarrow (b(X) \vee c(X)), a(X) \rightarrow (b(X) \vee c(X))\}$. The first of these is acceptable as a Horn clause but the second is not. There is then an issue about the form into which we should translate the second expression. We observe that in our domain of application (and we suspect in many others) circumstances like this one occur when the KIF expression's most likely computational use is in testing the consistency of the precondition of the implication. In this example we want to show that $b(X)$ or $c(X)$ is true when $a(X)$ is true. Since we are testing X, rather than generating an instance of it, we can use the equivalence between $P \rightarrow Q$ and $\neg(P \wedge \neg Q)$ to translate to a Prolog goal. This is, for the example, the consistency constraint $\neg(a(X) \wedge \neg(b(X) \vee c(X)))$.

In addition to the above simplified example, let us show an actual axiom from the Ontolingua Server, again one from the definition of the `Unit-Of-Measure` class:

```
(⇒ (Unit-of-Measure ?U)
   (Forall (?Other-Unit)
           (⇒ (And (Unit-of-Measure ?Other-Unit)
                   (Compatible-Quantities ?U ?Other-Unit))
              (And (Real-Number (Magnitude ?U ?Other-Unit))
                   (Positive (Magnitude ?U ?Other-Unit))))))
```

Which is translated into the consistency test:

$$
\begin{array}{l}
consistency\_test(\ \neg\ (\ unit\_of\_measure(U), \\
\qquad\qquad \neg\ (\ \neg\ (\ (\ unit\_of\_measure(Other\_unit), \\
\qquad\qquad\qquad compatible\_quantities(U, Other\_unit)\ ), \\
\qquad\qquad\qquad \neg\ (\ real\_number(magnitude(U, Other\_unit)), \\
\qquad\qquad\qquad\quad positive(magnitude(U, Other\_unit))\ )\ )\ )\ )\ )
\end{array}
$$

In this case, the translation outcome is not as handy. The sentence is unwieldy compared to what one could achieve by manually rewriting a Prolog specification of the KIF axiom. Also, the sentence is not relevant to the purposes of Ecolingua in the knowledge it represents. It involves the magnitude concept, which is alien to an ontology not concerned with data values (numeric or non-numeric) but with higher-level properties of the data.

In summary, all KIF-like logical sentences in the pruned set of clauses were translated to Prolog, either as program definitions or as consistency constraints. The resulting specification was small enough (220Kb) to allow us to read it and select definitions for reuse.

4.3 Experiment Discussion

We now highlight and summarise issues raised by employing, in particular, the Ontolingua Server in the ontology construction experiment, and conclude with a more general discussion on ontology reuse.

4.3.1 Employing the Ontolingua Server

We turned to the Ontolingua Server, with its varied ontologies library, hoping that it would assist us in quickly constructing an easy-to-apply, well-engineered ontology. It has been helpful in the design of Ecolingua in its conceptual form (Section 4.1), which would certainly have taken us longer had we not referred to the Server's rich pool of concepts. Selecting concepts of interest for reuse can be tedious, nonetheless, by browsing the ontologies or through word-matching search. Some other meaning-based search facility would have been helpful.

The frame system constructs offered by the Server's interface have been fairly convenient for representation of our concepts, promoting a somewhat structured formal specification. We also found it discouraging to have to write KIF axioms to add to this specification (Uschold and Gruninger, 1996). Most ontologies in the Server's library do not contain KIF axioms because, we suspect, it is unnatural to switch between representation formalisms and most users are not familiar with KIF.

Beyond the conceptual design, many practical issues arose in taking Ecolingua to its implementation form. Ecolingua's translation into Prolog by the Ontolingua Server was insufficient, leading us to devise our own complementary translation methods and tools. The problems started with getting as outcome from the Server's translation the union of the full content of all ontologies Ecolingua directly and indirectly referred to. This happens because of the Server's theory inclusion strategy (Farquhar et al., 1996). This is the easiest, standard way to obtain the union of both theories but at the risk of raising inconsistencies. To get round this problem we applied our ontology pruning method, which is heuristic in nature. This method reduces the ontology by making informed choices, but is not capable of producing a self-contained, consistent logical theory.

The Horn clauses translation method, in turn, compromises the expressiveness of KIF, the original underlying logical language. That would be the case in translating Ontolingua ontologies into any implementation language, since reasoners that support the full expressiveness of KIF, a first-order logic, cannot be built. The common logical parentage between KIF and Prolog allowed our compromises to be relatively modest.

In the end, after pruning and translation, only a small subset of the resulting clauses contributed to the Ecolingua specification that we actually employ for model synthesis (presented in Chapter 3 and Appendix A). Even assisted by our Horn clauses translator, we often had to manually fine-tune the axioms in this subset to the needs of the application. The cost of all this manual work of selecting and rewriting definitions certainly exceeded the benefit of the few definitions we were able to directly, or nearly directly, reuse.

4.3.2 Ontology Reuse

Over the past few years ontologies have been abundantly produced, many of them seemingly intended for some form of reuse (Uschold and Gruninger, 1996). Libraries of ontologies and tools to support ontology construction and sharing are available on the World Wide Web, e.g., (Ontolingua Server, 1995; Protégé, 2000; DAML, 2002). However, the number of reports on how and for what ontologies are used or reused once they are designed still does not grow in the same rate (Uschold, 1998). Convincing successes on application and reuse of ontologies in a realistic scale are scarce.

Ecolingua's construction pushes further the ontology reuse potential and currently available machinery — the Ontolingua Server — by attempting to reuse multiple ontologies, in response to the domain's diversity. Moreover, Ecolingua, as opposed to being an end in itself, serves a composite purpose: to allow ecological data properties to be expressed and, thereby, support model synthesis by providing substantiating information and enabling reuse of models.

Most of the literature either addresses ontology reuse from a theoretical perspective or is limited to reuse of conceptual-level ontologies. In reports of experiences on the latter, the "reused" ontologies are mainly looked up for reference, rather like glossaries, and from there knowledge engineers manually write their own ontology or other application, possibly adapting selected definitions of the referred ontologies, in their preferred representational language.

We have shown in this chapter that bringing ontology reuse to the implementational level is a process fraught with difficulties, only poorly supported by currently technology, demanding an impractical amount of time and effort. This corroborates with the assessment of similar experiments reported in (Swartout et al., 1996; Uschold et al., 1998; Valente et al., 1999; da Silva et al., 2002)[2].

Referring to definitions found in multiple ontologies during Ecolingua's conceptualisation led to an oversized automatically translated union of ontologies with no guarantee of consistency, despite the included ontologies all sharing Ontolingua's representational framework. A more challenging, yet plausible, scenario would be to reuse during conceptualisation definitions from multiple ontologies that are only available represented in distinct formalisms. Reuse and integration of multiple ontologies supported by fully automatic translators is still far from achievable, hindered by the old problem, larger than ontologies, of translation between logical theories represented in semantically heterogeneous, inherently biased formalisms (Uschold et al., 1998; Valente et al., 1999). Methods are lacking to reliably reduce scope and content of imported ontologies. We have shown one heuristic method. It would be useful for ontology sharing tools to make such methods available, informing users of possible drawbacks where heuristics are involved.

Still, our and other experiments suggest that in ontology reuse even if, idealistically, conceptually suitable definitions are reused, and semantically sound translations and reliable reduction methods are applied, the resulting specification can be cumbersome, requiring manual rewrites that take closely into account the assumptions and intended usage of the new ontology (Grosso et al., 1998).

Unfortunately, we do not have a more objective cost-effectiveness metric for this experiment. Neither could we use an off-the-shelf one for practical metrics on ontologies reuse are yet to be deployed; (Cohen et al., 1999) is the pioneering work on that. Our subjective overall assessment is that if we were to engineer another domain-specific ontology, we would again use the Ontolingua Server or a similar tool for browsing existing, potentially useful, ontologies, but would prefer to write the specification of

[2](da Silva et al., 2002) includes an earlier version of the report we give here of Ecolingua's construction experiment.

the ontology manually, selectively rewriting reusable definitions from other ontologies, being in complete control of every ontological commitment made.

In summary, one can contemplate an ontologies sharing scenario — this has been achieved in the experiment to a degree — however, current support technology did not prove mature enough to sustain such scenario. Methods and tools for cost-effective reuse of ontologies in ontology construction are still deficient, linked to the largely unsolved problem of translation between representation formalisms. Ontology reuse is fundamentally difficult because of the unboundedness and eclecticism of concept formation by humans. But, once the ontology is in place it can indeed empower knowledge reuse with the benefit of efficiency as this thesis shows (Chapters 8 and 9).

Part III

Model Synthesis and Reuse

Chapter 5

Metadata: Sources and Ecolingua Descriptions

As we know from Chapter 3, Ecolingua is intended for description of ecological data and from these descriptions, which we call metadata, we want to synthesise models. With our choices of vocabulary in place in Ecolingua, we can now encode descriptions of specific instances of the metadata-based synthesis problem.

This chapter bridges the chapter on Ecolingua and the next three, which discuss synthesis heuristics, methods and mechanisms by showing through example the kind of information metadata stems from and how it is specified in the ontology.

To specify metadata is to instantiate Ecolingua

Input to one problem instance is two-fold: a data set, and preliminary information about the model to be synthesised such as its objectives and assumptions. Ecolingua's constructs work as templates for description of this input. Writing metadata terms consists of applying grounding substitutions to Ecolingua predicates. This is a non-deterministic knowledge representation exercise to which we do not have an automated tool nor a precise procedure to follow. It is done by a metadata "mark-up person" who looks at the data and preliminary model information to identify objects and relations of interest, and finds appropriate concepts in Ecolingua to instantiate. Knowledge of

Ecolingua is required in order to understand how the problem (data and preliminary model information) can be mapped into the ontological concepts.

The subjective manner in which metadata is described makes it best shown by example. In the context of the sections on metadata sources that follow, we will be using a pond management example found in (Grant et al., 1997, Chapter 9). It is an example of a common scenario in Ecology where some ecological system is economically exploited and better management strategies are sought for. Having a model of the system of interest allows for simulation and evaluation of alternative strategies. In particular, the way in which the example is presented, with detailed annotations on development of the model, makes it interesting material for us. We can think of it as a best case setting for exercising metadata specification and subsequently synthesising models, where data, model objectives and assumptions are all explicit and available. Clearly, the amount and quality of information available at this stage will have an effect on synthesis success.

However, we do not present here the whole metadata set of the pond management example. This can be found in Appendix B which contains the Prolog specification of the full set, accompanied by quotes of the referent metadata sources. In this chapter we show selected descriptions that are representative of the most commonly used terms in Ecolingua vocabulary, presented in full in Sections 3.2 and 3.3. The reader may also consult that chapter for other details on the Ecolingua constructs used in the examples that follow.

A last remark is that descriptions are shown (here and in Appendix B) as if they are produced in an one-off manner. In fact, each description may be refined along the way, as the metadata mark-up person works through all the available information, until a final form is reached as part of *a* round metadata set. More than one set of descriptions may suit the same input information as it can be interpreted and marked up differently.

5.1 Metadata from Field Data

All models, regardless of modelling paradigms and practices, are just approximate representations of reality. In that sense they are all wrong. This is so because, to a greater or lesser degree, the representation will inevitably be prejudiced by the model designer's own interpretation of the real system. In this context, data from direct observation or experimentation with the real system is the least biased kind of information and thus invaluable in substantiating the model (Grant et al., 1997). Furthermore, having models rooted in data allows for well-grounded analysis and interpretation of simulation results.

The problem is that bridging concrete data to abstract conceptual models is not straightforward. A compromise is to initially build models without much regard to the data and later fit them to the data in retrospect. For data to support model conceptualisation, it is necessary to raise the data's level of abstraction to make it level with and thus more easily associated with models. This is the effect we achieve by describing the data through Ecolingua, transforming it into metadata. Associations with models become possible in this way because Ecolingua prescribes the properties and relations of the data that translate into conceptual model representations.

Field Data Metadata by Example

Table 5.1 appears in (Grant et al., 1997) as part of the information provided for construction of a pond management model. It is typical of ecological data in format and content — a table filled with measurement values of a variable of interest taken under certain conditions. Also, it consists, most likely, of processed data, resulting of application of some statistical method to extensive raw data as collected in sampling campaigns. We assume processed data is given for metadata specification and subsequent model synthesis.

At the metadata level, Table 5.1 contains information about two quantities, namely net production rate of plants and water temperature, and a functional relationship between them. The two quantities map into the Ecolingua quantity classes specific-

(a) Net Production Rate of Plants (g Produced/kg Plant Biomass-Day)

	Net Production				
Water Temperature (°C)	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5
10	0.225	0.422	0.072	0.358	0.002
15	0.639	0.601	0.578	0.143	0.231
20	0.595	0.353	1.180	0.377	1.555

Table 5.1: Data on processes occurring within the pond system including (a) the net production rate of plants as a function of water temperature (extract from (Grant et al., 1997)).

rate and temperature-of respectively, and thus instantiate the Ecolingua predicates $spf\_rate(R, R_{abs}, Mt, U)$ and $temperature\_of(T, E, U)$. Generally, each metadatum, or metadata term, is represented by the unary predicate *data* with a ground Ecolingua term as argument. For descriptions of model requirements specific predicates are used.

Hence, the descriptions of the two quantities above are represented as:

$$data(spf\_rate(plant\_production\_rate, plant\_production, biomass, g/kg/day))$$
$$data(temperature\_of(water\_temp, water, celsius))$$

These are examples of the main descriptive sentence required for every quantity — a ground term as argument to the *data* predicate, with its functor being a quantity class (Section 3.2), its first argument the quantity's name, and its last argument the quantity's unit of measure.

Units of measure are crucial due to the information they hold on the physical dimension of the quantities (Section 3.2.1). Modellers are advised to get the units right from the beginning and to rethink the model structure if mismatching units occur along the design process (Ford, 1999; Grant et al., 1997). Analogously, the synthesis mechanism uses the unit of measure of a quantity to check whether it is of a physical dimension that can be associated with certain types of model components (Section 7.4.1).

The sentences are also illustrative of non-primitive sentences with constants that require further descriptions, such as *plant_production*, *biomass* and *water*. Additional data and/or preliminary model information should furnish the complementary descriptions (see Appendix B).

Lastly, the functional relationship of undetermined sign between the two quantities is represented by instantiating the $influences(Q, Q', Sign)$ predicate, with *water_temp* as influencer quantity and *plant_production_rate* as influenced quantity, giving:

$$data(influences(water\_temp, plant\_production\_rate, ?))$$

Being able to describe influences of undetermined sign helps to maximise the use of metadata evidence. An influence is still an influence — a valuable information for model structure synthesis — regardless of its sign being unknown.

5.2 Metadata from Preliminary Modelling Information

Most often, it is not data alone that support model conceptualisation. Usually there are aspects of the system being modelled for which no data is available. In such cases, modellers resort either to the literature, for theoretical or generally applicable empirical information, or to expert opinion (Grant et al., 1997).

Much of this kind of non-data information, that we generally call 'preliminary modelling information', can be found in statements of model's objectives and assumptions. Complementary to field data, we also use these statements as sources for metadata specification.

5.2.1 Model Objectives

Ecological modelling authors place great importance on model objectives being clearly stated (Haefner, 1996; Grant et al., 1997; Ford, 1999). This is essential, they argue, because model objectives provide focus for model development as well as context and standards for interpretation and evaluation of simulation results.

Model objectives are usually formulated in general terms at first. For the pond management example, for instance, the general model objective is stated as “to determine if any of several alternative stocking and harvesting schemes will yield higher than current profits”. For a predator-prey system this could be “to simulate interactions between one prey population and one predator population to identify circumstances under which predators affect prey population dynamics and vice versa” (Grant et al., 1997).

Because they are so broad, general model objectives like these are of little use in informing model conceptualisation (either human-made or via automated synthesis). For this purpose they must be distilled into more specific objectives. In his ‘Principles of Qualitative Formulation’ (Haefner, 1996) argues that to discover the simplest description of a system that will satisfy the model’s general objectives, one should start by precisely identifying *the questions to be answered* (by the model) and *the quantities and their units needed to answer the questions*. It is from this level of detail of model objectives that we can produce metadata that is useful for informing model synthesis.

Model Objectives Metadata by Example

Back to the pond management example, one of the specific questions to be answered by the model is:

> The current scheme is stocking 75 0.227-kg fish on April 15 and harvesting them on November 15. What profit is associated with this harvest?

The sentence about the current scheme, albeit short, on close inspection gives rise to a wealth of modelling information. It denotes two ecological processes (*stocking* and *harvesting*), the dates when they take place, the number of fish stocked (75), and the weight of individual fish (0.227 kg).

System dynamics models represent transfer of material through a system by processes, such as stocking and harvest in the case of this system of interest. Material transfer processes (Section 2.4) like these map into Ecolingua quantity class absolute-rate with nonground term $abs\_rate(R_{abs}, Mt, E_{from}, E_{to}, U)$. Instantiating this term with the two processes yields the descriptional sentences:

$$data(abs\_rate(stocking, biomass, outside, fish, kg/ha/day))$$
$$data(abs\_rate(harvest, biomass, fish, harv\_fish, kg/ha/day))$$

The dummy constant *outside* is used to instantiate origins (E_{from}) or destinations (E_{to}) of the material that the process carries that are unspecified or external to the system of interest. The unit $kg/ha/day$ comes from additional information later given (an example of description refinement mentioned earlier).

The other concerned data are described as:

$$data(number\_of(number\_fish\_stocked, fish\_stocked, 1))$$
$$data(weight\_of(indiv\_fish\_weight, indiv\_fish, kg))$$
$$data(time\_of\_event(stocking, stocking\_date))$$
$$data(time\_of\_event(harvest, harvest\_date))$$

And the influence relations between them and the processes as:

$$data(influences(number\_fish\_stocked, stocking, ?))$$
$$data(influences(init\_indiv\_fish\_weight, stocking, ?))$$
$$data(influences(stocking\_date, stocking, ?))$$

$$data(influences(harvest\_date, harvest, ?))$$

Now the question itself “What profit is associated with this harvest?” indicates that the *profit* made on the pond system is a quantity whose behaviour the model is expected to simulate — a model goal variable.

$$model\_goal\_var(amt\_of\_money(profit, pond\_system, \$))$$

5.2.2 Model Assumptions

Another aspect of the insufficiency of data for modelling is that modellers’ knowledge about the system of interest usually transcends what can practically be documented by data. Thus, commonly, a great deal of modelling decisions may be made on the basis of assumptions derived from this knowledge.

Like model objectives, model assumptions also indicate relevant ecological quantities as well as their properties and relations. In particular, they are rich in expressions such as ‘A depends on B’, ‘A is limited by B’, ‘A increases with B’, etc., which suggest interdependencies between variables.

Model Assumptions Metadata by Example

We shall again use an extract from the information for construction of the pond management model to illustrate model assumptions and their expression in Ecolingua:

> Processes that affect biomass dynamics of aquatic plants include net primary production [(measured in kg/ha/day)] ... [which] is a function of biomass of aquatic plants and water temperature.

Net primary plant production, like stocking and harvesting, is also a material transfer process, giving the descriptional sentence:

$$data(abs\_rate(plant\_production, biomass, outside, aquatic\_plants, kg/ha/day))$$

And the process’ functional relationships become:

$$data(influences(aquatic\_plants\_biomass, plant\_production, ?))$$
$$data(influences(water\_temp, plant\_production, ?))$$

It is apparent from the data and preliminary modelling information quoted thus far that *biomass* must have a key role in the model. In fact, biomass is the model material, i.e., the material whose flow throughout the system of interest the model shall represent. It is essential for model structuring guidance to ascertain such materials as they determine the division of models into submodels.

The following quotation establishes *biomass* as the pond management model material:

> Profit is calculated based in part on the biomass of fish accumulated on the harvest date, which depends on the net accumulation of fish biomass in the pond since stocking. Therefore, we need to represent fish *biomass* dynamics. Because fish are herbivorous, their growth depends in part on the amount of plant biomass accumulated in the pond at any given time, so we also need to represent plant *biomass* dynamics.

Which is represented as:

$$model\_mat(biomass)$$

Lastly, another important piece of information for model synthesis is the model's time unit. It may be explicitly stated as a model assumption, or implied, as in this example. Note that *day* is the time unit of the net production rate of plants, in Table 5.1, and also appears in references to other process rates. Hence, *day* is taken for the model's time unit, described as:

$$model\_time\_unit(day)$$

One can expect reasonably adequate model time units to appear in data like in Table 5.1 and also within preliminary model information, since these are elaborated having in mind time lengths that make sense to the problem at hand. This is obviously a simplification. The questions to be answered by the model may require temporal changes in processes to be represented in a finer or coarser time resolution. Or, distinct model time units may appear in different metadata sources, which would require conversion to a single unit. We shall assume however that these considerations have been taken into account on provision of the metadata sources — references in them to the rate time units are homogeneous and adequate.

Chapter 6

Heuristic Knowledge for Synthesis of Model Components

Recall from Chapter 1 that our task is synthesis of system dynamics model structures based on ontology-described data, or metadata for short. The task is first tackled with the metadata-only approach (so called for reasons that shall become clear in Chapter 8). We start presenting the approach in this chapter by discussing its formal definitions of model components — the compositional units of the model structures.

A metadata set, like the one in Chapter 5, is the starting point of the model synthesis process. Now, we need formally represented knowledge that can connect the metadata terms to model structure.

The first point of reference for formulating such knowledge is domain knowledge: ways in which system dynamics modellers connect data to model structure. This, to a great extent, however, is a tacit kind of knowledge, not readily available from textbooks or experts (the old knowledge acquisition problem). No precise characterisation of how data and model mesh has been proposed as yet. We identified commonplace modelling practices and based on that developed a characterisation of associations between data and model structure that is rooted in physical dimensions of ecological quantities.

We make no claim that this characterisation is cognitively plausible. The synthesis process we have automated is not like human modelling. Modellers use a great deal of tacit knowledge about the real-world system-of-interest. We only use information about the system-of-interest that is made explicit as data and model assumptions. In sum, the knowledge we formalise as part of our synthesis systems is not comparable to the extensive expertise and techniques applied by ecological modellers.

Also, modellers apply much ecological modelling expertise and techniques other than what we formalise in our synthesis system. We formalise the corner of the modelling process to do with linking data and their properties to model structure, guided by model objectives and assumptions. This is done to support model conceptualisation, the first stage in the modelling life-cycle (see Chapter 1). Also in this respect, our approach is unlike conventional modelling practices, where, usually, data is only directly used in the later stages of quantitative specification of the model. It is difficult in practice for modellers to link concrete data to conceptual models. They do subjectively consider what data is there to support the model, but what mainly prevails, maybe for the sake of conceptual freedom, is the modeller's understanding of the system-of-interest, the variables and their interdependencies deemed relevant to the modelling problem at hand. Then, after the conceptual model is designed, it is confronted with and fitted to the available data, mostly through parameter estimation.

The twist in our synthesis approach that enables the mapping of data into conceptual models is that, in describing the data in Ecolingua, we make explicit the characteristics of the data that connect to models. Data becomes metadata and from then on, a synthesis process is carried out that hypothesises conceptual model alternatives that have the advantage of being consistent with the available data. These are meant to be prototypical models, to which modellers can apply further knowledge of the system-of-interest and other expertise to produce refined final models.

The synthesis process consists, basically, of a reasoning procedure performed over domain-specific knowledge formalised as Horn clauses. Such knowledge is bound to be heuristic, given the inherent heuristic nature of the conceptual modelling task itself, as well as the inherent imperfect nature of ecological data which grounds the task. As a representation of heuristic knowledge, the set of domain-specific modelling rules we have cannot be claimed either complete or infallible. Devising heuristic rules is an empirical problem. One can only know how good heuristics are by putting them to the test on problem instances (Luger and Stubblefield, 1993). In the forthcoming evaluation chapter (Chapter 9) we show that the set of modelling heuristic rules we show here is general enough to support the synthesis of a wide range of models.

The space of solutions of the synthesis problem is large. The number of models that a single data set can potentially support is uncountable. Proportionally, a metadata set can give rise to many model structure solutions. In representing modelling knowledge, the heuristics must be able to rule out choices, narrowing them down to those that are best justified in the metadata. In that sense, the modelling heuristics are constraints that restrict the space of synthesis solutions. Constraint satisfaction lends itself a convenient framework within which we shall discuss our synthesis problem-solving methods.[1]

The goal of this chapter is to present all the pieces of static knowledge, the synthesis constraints, that define each type of model component. Component is a blanket term we use to refer to both model elements and model connections. Elements are the model's nodes — in analogy to a graph structure — namely, state variables, flows, intermediate variables, parameters and driving variables. Each element represents an ecological quantity, has a unit of measure adequate to its role in the model, and has an equation defining its behaviour. Connections are the model's arcs, namely, flows and links, that connect pairs of model elements. Note that flows are both element and connection. (See Section 2.4 for a model example.)

There are, largely, two kinds of constraints for synthesis of model components: metadata constraints and integrity constraints. Under metadata constraints we place all constraints that involve descriptions from the metadata set as evidence for synthesis of model components. These range from underlying ontology constraints to classification of model elements according to supporting metadata evidence. While metadata constraints concern single model components, integrity constraints prevent conflicts

[1]Let us note, however, that we do not take up off-the-shelf resources from the Constraint Logic Programming (CLP) machinery. The gain over the conventional logic programming techniques we apply would not be significant. To our synthesis task it suffices to find solution instances. It is not necessary, as normally done in CLP, to find an initial set of solutions, constrain it until it is optimum, and then take solution instances from this set.

between components from occurring within a model.

The chapter is organised round the constraint kinds. It has three sections: the first two address metadata and integrity constraints, and the third shows the integration of these two kinds of constraints into rules that define each type of model component. The next chapter then focuses on the reasoning techniques and algorithms in the Synthesis-0 system that assemble the components into full conceptual models.

6.1 Metadata Constraints

Metadata constraints define the bridges between metadata and the synthesis mechanism. The solving of these constraints is carried out in a hypothesis-formation fashion, by exploring plausible relationships between metadata and model components.

6.1.1 Underlying Ontology Constraints

We should start with the most fundamental of the constraints, the ontology constraints. They are the bedrock of the synthesis system. Their definitions consist of Ecolingua axioms (Chapter 3). Every metadata that feeds into the synthesis mechanism must conform to Ecolingua, i.e., must satisfy the applicable ontological constraints.

The constraints are called upon in an as-needed basis. Every time a metadata term, or description, is required by the modelling heuristic rules (Section 6.3) in action, it is, if existent, retrieved from the metadata set, and attempted to be proven over the corresponding Ecolingua axiom. If proved the description is carried forward into the proceeding synthesis process. Otherwise, if the description does not comply with the ontology, it is considered invalid and is rejected. By allowing only descriptions that conform to the ontology to get through we also restrict the space of solutions. The ontology compliance checks are implemented by way of meta-interpretation, as we shall discuss in Chapter 7.

In conventional human-made model design, the decisions regarding data that back model structure, as well as the interpretations data is given, are highly subjective and thus virtually inaccessible to others but the designers themselves. In our approach, by contrast, having the synthesis based on ontology-compliant metadata only, renders the meaning of synthesis input information defined and accessible — an asset for model analysis, evaluation, and sharing. In fact, in Chapter 8 we present a second synthesis approach that demonstrates ontology-enabled model reuse.

6.1.2 Library of Metadata↔Model Association Rules

At the basis of the synthesis process we have a library of Metadata↔Model association rules. Strictly speaking, these should be called ‘quantity type-model components’ association rules, as the metadata involved are only the quantity type terms in Ecolingua (see Chapter 3 for definitions of each quantity type), each being associated with a model component term or a conjunction of them.

The association rules bridge metadata and model structure, by heuristically defining what types of model components each of the quantity types suggest. Domain knowledge dictates that these must not be one-to-one associations. A single quantity type may suggest multiple alternatives of model component types. To allow for this, while preserving the direct logical interpretation of the rules, the associations are established through abduction in the metadata-only approach. We delay a more detailed discussion on such use of abduction until Chapter 7, where the reasoning techniques in the Synthesis-0 system are addressed.

For devising the rules we, firstly, tried and matched quantity types with model component types. In a fundamental level, the matching is between physical dimensions, that characterise quantity types (as defined in Ecolingua— Section 3.2), with the roles that the different types of components play in the model. Secondly, these intuitions were empirically checked up on a wide range of model examples found in the literature.

The library is shown in Table 6.1. The model component types, presented in our introduction to system dynamics in Section 2.4, are represented by the terms on the right hand side of the implications, with predicates *sv*, *iv*, etc. Later on, in Sections 6.3 and 8.6, we present heuristic rules defining each of the model component types.

Rules `1` **and** `2` The *mass*, or 'quantity of matter' (Massey, 1986), physical dimension characterises the *amount of material* quantity type in Ecolingua. Such metadata concept of *amount of material* naturally associates with the role of *state variables* in the models, that of representing accumulated material in parts of the system (Ford, 1999; Forrester, 1961).

Other amounts appear in the models representing calculations over amounts in state variables, for example, total amount of carbon in a forest vegetation adding up amounts of carbon in roots, leaves, branches and stems. This kind of aggregated amounts are represented as intermediate variables, since they are influenced by other variables in the model (the state variables).

Rules `3` **and** `4` The same applies to quantities of type *material density*, which represent amount of material in relation to space ($mass/length$ dimension).

Rules `5` **and** `6` Similarly, amounts of non-physical things, such as money, are associated with state variables and intermediate variables. Note that differently from the *amount of material* and *material density* types that generically applies to any physical material, in this case the quantity type itself determines the material argument (the *money* constant) in the state variable term.

Rule `7` Amounts per se are static quantities. By contrast, activity or processes that occur over time are represented by dynamic quantities we call *absolute rates*. Their physical dimension is either $mass/time$, or $mass/length^n/time$, or $money/time$. In the class of system dynamics models we deal with, processes are material transfer processes, e.g., in a pond system, consumption of plants by fish is a process that over time transfers biomass from plants to fish; harvest of fish is a process that transfers biomass

1	$assoc\_rule((amt\_of\_mat(A,Mt,E,U) \leftarrow sv(A,Mt,E,U)))$
2	$assoc\_rule((amt\_of\_mat(A,Mt,E,U) \leftarrow iv(A,U)))$
3	$assoc\_rule((mat\_dens(A,Mt,E,U) \leftarrow sv(A,Mt,E,U)))$
4	$assoc\_rule((mat\_dens(A,Mt,E,U) \leftarrow iv(A,U)))$
5	$assoc\_rule((amt\_of\_money(A,E,U) \leftarrow sv(A,money,E,U)))$
6	$assoc\_rule((amt\_of\_money(A,E,U) \leftarrow iv(A,U)))$
7	$assoc\_rule((abs\_rate(R,Mt,E_{from},E_{to},U) \leftarrow mattrans(R,Mt,E_{from},E_{to},U)))$
8	$assoc\_rule((spf\_rate(R,R_{abs},Mt,U) \leftarrow iv(R,U) \wedge link(R,R_{abs})))$
9	$assoc\_rule((spf\_rate(R,R_{abs},Mt,U) \leftarrow param(R,U) \wedge link(R,R_{abs})))$
10	$assoc\_rule((spf\_rate(R,R_{abs},Mt,U) \leftarrow dv(R,U) \wedge link(R,R_{abs})))$
11	$assoc\_rule((temperature\_of(T,E,S) \leftarrow iv(T,S)))$
12	$assoc\_rule((temperature\_of(T,E,S) \leftarrow param(T,S)))$
13	$assoc\_rule((temperature\_of(T,E,S) \leftarrow dv(T,S)))$
14	$assoc\_rule((weight\_of(W,E,U) \leftarrow iv(W,U)))$
15	$assoc\_rule((weight\_of(W,E,U) \leftarrow param(W,U)))$
16	$assoc\_rule((weight\_of(W,E,U) \leftarrow dv(W,U)))$
17	$assoc\_rule((number\_of(N,E,U) \leftarrow iv(N,U)))$
18	$assoc\_rule((number\_of(N,E,U) \leftarrow param(N,U)))$
19	$assoc\_rule((number\_of(N,E,U) \leftarrow dv(N,U)))$
20	$assoc\_rule((percentage(P,E,U) \leftarrow iv(P,U)))$
21	$assoc\_rule((percentage(P,E,U) \leftarrow param(P,U)))$
22	$assoc\_rule((percentage(P,E,U) \leftarrow dv(P,U)))$
23	$assoc\_rule((amt\_of\_time(T,Ev,U) \leftarrow param(T,U)))$
24	$assoc\_rule((amt\_of\_time(T,Ev,U) \leftarrow dv(T,U)))$

Table 6.1: Library of Metadata$\leftrightarrow$Model association rules.

to outside the pond. Therefore, *absolute rates* are associated to *material transfer* model elements.

Material transfers are an intermediate structure between *absolute rates* in the metadata set and *flows* in the model. They may be worked out into flow connections in the model, provided that the concerned synthesis constraints are satisfied (Section 6.3.3.1). The association cannot be made directly because the origin of the material transfered by a process (quantified as an absolute rate) as well as its destination are different in nature from origins and destinations of flows. The distinction takes place because metadata and models belong to different levels of abstraction. Rates are metadata, where origins and destinations of material are entities (e.g., a consumption process transfers biomass from *plants* into *fish*). Accordingly, Ecolingua prescribes that the arguments E_{from} and E_{to} of the *absolute rate* term are entities (Chapter 3). On the other hand, flows are model connections. Their origins and destinations are either sources and sinks, representing the outside of the system, or, more importantly, state variables. A state variable does not represent just an entity, but the content of material within an entity (e.g., a consumption flow transfers biomass from *plants biomass* into *fish biomass*).

Rules `8`**,** `9` **and** `10` Conjoined with absolute rates, there are *specific rates*. They also measure material transfer over time but in relation to the amount of material in entities involved. For example, for the process of consumption of plants by fish a *specific rate* would be a rate of biomass transfer per day in relation to the amount of biomass in fish.

This relation between absolute and specific rates is represented in system dynamics models as influence links with the model element that holds the specific rate regulating the model element that holds the absolute one. Thus, a description of a specific rate is rich enough to suggest both a model element holding it and a link from it to an element holding the absolute rate.

In system dynamics models specific rates appear modelled as either *intermediate variables*, *parameters* or *driving variables*. Hence these association rules — a *specific rate* may become an *intermediate variable*, a *parameter*, or a *driving variable* in the model, and be *linked* to its respective *absolute rate* model element.

Rules `11` **to** `22` The quantity types in the association rules thus far, namely, amounts (*amount of material*, *material density* and *amount of money*), and rates (*absolute* and *specific*) are distinctive in that they all involve a material *Mt*. Accordingly, quantities of these types are mostly associated, directly and indirectly (in the case of specific rates), with equally distinctive elements in the model, i.e.*state variables* and *flows* that represent the pathway over which the material of interest flows.

Remaining quantity types in Ecolingua, namely, *temperature_of* (*temperature* dimension), *weight_of* (*mass* dimension), *number_of* and *percentage* (*identity* dimension), not having the distinctive material feature, are associated , by exclusion, with modelling elements other than *state variables* and *flows*, i.e., *intermediate variables*, *driving variables* and *parameters*. We ratified this heuristic by finding, to each of the association rules, at least one example of model in the literature where the association rule holds.

Rules `23` **and** `24` In associating the quantity types in rules `11` to `22` with *intermediate variables*, it is implicit that these quantities/variables are liable to being influenced by other quantities/model elements. In the case of quantities of type *time*, of the *time* dimension, we exclude the association with *intermediate variables*, since, by definition, variables representing time cannot depend on other variables in the model. They are represented either as *parameters* — a model input that does not change during simulation — or as *driving variables* — a model input that may change during simulation, but not caused by other model element.

Note that almost every quantity type is associated with an *intermediate variable* in the library of association rules. This is in accordance with *intermediate variables* being, generally, the most versatile, and thus most relatively abundant element type in system dynamics models.

The set of rules presented support synthesis of a wide range of models within the class of models tackled, as shown in Chapter 9. We must stress, however, that the library is not complete, neither has it been intended to be so — because it is a representation of heuristic knowledge, completeness is inherently unattainable. The idea is that the library should be augmented, or altered even, ameliorating the associations between metadata and model structure and widening the range of synthesisable models. Also, the synthesis systems are engineered to allow for the library to be modified without causing side-effects or requiring other modifications elsewhere. The library's simple form is meant to make updates easy. Recall that Ecolingua is also extensible. Addition of new quantity types to the ontology, for instance, would trigger addition of new Metadata↔Model association rules to the library.

6.1.3 Influence Constraints

Work in modelling of complex systems shows that some knowledge of influences, causal relationships, or functional relationships (to mention but a few terms for this concept) between objects in the domain is essential for automated and non-automated approaches. It is important for our synthesis systems too. Because it is such a widely used concept, we have this section specifically on influence constraints, wherein we discuss our use of influences in perspective, relating it to three other relevant approaches.

6.1.3.1 Influences in System Dynamics Causal Loop Diagrams

In system dynamics modelling itself, we find Causal Loop Diagrams (Ford, 1999). While flow diagrams (like the one in Section 2.4) are designed for simulation (the ultimate purpose of system dynamics models), loop diagrams are a tool for understanding and communication of models. Types of model elements (or variables) — state variables, flows, intermediate variables, parameters, driving variables — are not specified in loop diagrams. All variables are treated the same and denoted by their names only. Likewise, types of connections — flows, links — are not distinguished either. Uniform arrows denote causal connections (or influences) between variables, regardless of whether they may be a flow (material transfer) or a link (information transfer) in the corresponding flow diagram. The arrows are labelled either $+$ or $-$. The positive sign stands for a causal connection where the two variables change in the same direction (both increase or both decrease). Otherwise, if the connected variables change in opposite directions, the arrow receives a negative sign. In that way, by abstracting types of variables and connections, a loop diagram gives an overall picture of cause-and-effect relationships and information feedback in the modelled system, as opposed to flow diagrams where the different roles of model components are emphasised.

6.1.3.2 Influences in Qualitative Process Theory

Influences are a key modelling concept in Qualitative Process Theory — QPT (Forbus, 1984), the foundational theory of Qualitative Reasoning (Kuipers, 1994). Influences in QPT specify relations between system processes and quantities that can inflict change in quantities. Change is assumed to be caused, directly or indirectly, only by processes (the sole mechanism assumption). Changes inflicted by processes on quantities are modelled as direct influences. For example, Q is a quantity representing the amount of some 'stuff', and n is the flow rate of some process that directly influences Q. This would be written as: $I+(Q,n)$, if the influence is positive (increasing monotonic); $I-(Q,n)$, if the influence is negative (decreasing monotonic); or $I\pm(Q,n)$, if the sign is unspecified. Indirect influences denote changes inflicted on a quantity by some other quantity, which in turn changes due to direct or indirect influences. For example, an indirect influence from a process on a quantity Q_1 exists, if the process directly influences a quantity Q_2 which indirectly influences Q_1.

6.1.3.3 Influences in TRIPEL — a system for automated compositional modelling of complex systems

In (Rickel and Porter, 1997; Rickel, 1995) influences are also used to compose qualitative models. Influences among system variables are the building blocks from which a compositional modelling algorithm, within a system called TRIPEL, constructs the simplest model that can answer a given prediction (what if) question. Additional domain knowledge on plant physiology specified by instantiating a large multipurpose biology knowledge base is also used. Assisted by this domain knowledge, and given the complete network of influences, the algorithm searches for the smallest portion of the network that answers the prediction question.

A similar notion to QPT's classification of influences into direct and indirect takes place in TRIPEL. QPT's direct influences are equivalent to so called differential influences. Differential influences on a variable are combined to form a differential equation that defines the variable. QPT's indirect influences are equivalent to functional influences. These, in turn, are combined to form algebraic equations defining influenced variables. The notation used consists of arrows from influencer into influenced variables. Each influence is labelled with the sign of its partial derivative (+ for increase, – for decrease). These are like the Causal Loop Diagrams mentioned above, except that there differential and functional influences are not distinguished.

6.1.3.4 Influences in Metadata-supported Model Synthesis

How influences are described

The Ecolingua term $\mathit{influences}(Q, Q', Sign)$ (Chapter 3) is used to specify influence relations between quantities. The influence sign is positive $(+)$ when the two quantities change in the same direction, negative $(-)$ when the two quantities change in opposite directions, or is undetermined (?). Like all metadata, influence descriptions come from field data and other preliminary modelling information (Chapter 5).

What influence descriptions represent

Our specifications of influences belong to the metadata level of abstraction, differently from QPT and TRIPEL where they denote relationships at the system level of abstraction. In QPT and TRIPEL the influences are between system variables. Our influences are between quantities in the data. Therefore, for specifying influences, we do not have a prominent notion of system process like in QPT, that determine *direct* and *indirect* influences. Metadata sources may refer to processes, but, as metadata, they are described as rate quantities. Similarly, *differential* and *functional* influences as in TRIPEL are not particularly distinguishable in metadata sources.

Causal Loop Diagrams relate to our influence descriptions more closely. Still, in loop diagrams there are influencers and influenced model variables as opposed to data quantities. Another difference is that we may have the sign of the influences unspecified (like in QPT). The similarity between the approaches is that in both each influence represents a direct causal relationship from influencer to influenced element, regardless of what type of model element they *are* — in the case of loop diagrams — or *may become* — in the case or our synthesis systems — in a corresponding system dynamics flow model. Clearly, indirect influences derive from the direct ones (e.g., suppose A, B and C are variables or quantities; if A directly influences B and B directly influences C, then A indirectly influences C). In our synthesis systems, from the set of individual descriptions of direct influences between quantities may stem a network of model connections (those whose synthesis constraints are satisfied) that, like in the loop diagrams, can be diagrammatically represented.

How influence descriptions are explored for synthesis

Influences are not as central to our synthesis systems as they are to QPT and TRIPEL. The qualitative models TRIPEL produces, in particular, are a subset of the system description, the given specification of the system of interest in terms of variables and influences between them. So, the success of the compositional modelling algorithm, in finding the smallest portion of the system description that answers the prediction question, depends on how good the given system description is in the first place. For success, it must contain all relevant variables and influences.

We could not have this sort of requirement, since the metadata sources from where the influence descriptions come (field data and textual statements of model objectives and assumptions), are inherently incomplete and possibly ambiguous. The system needs to be flexible. The more influence descriptions the metadata sources expel, so much the better — that means more metadata evidence for synthesis, but completeness is not a requisite.

Influence descriptions are not the only kind of metadata we use for establishing model connections. As we shall see in Section 6.3, an influence description is required in some of the rules for establishing flows and links. They also determine the evidence class of some types of model elements, but not all. Generally, the importance of metadata (an influence description is just one of their kinds) is relative to the type of model component under synthesis. Also, individual descriptions do not go a long way without others. For synthesis to work on its best a metadata *set* with a variety of interrelated metadata terms is needed.

The models constructed are conceptual structural models. The signs in the influence descriptions are important to that because, for flow connections (material transfers) established from influence descriptions, it is the sign that determines the direction of the flow in relation to its state variable(s). Establishing a flow connection F into a established state variable S would require a description of the form *influences*$(F,S,+)$ (together with other constraints — see Section 6.3.3.1). The positive influence determines that F must be a flow *into* S, i.e., F causes S to increase. Conversely, a flow F *from* S, would require a negative influence of F on S, causing the latter to decrease.

As for links (information transfers), the influence sign is irrelevant for structural specification. The direction of links is always *from* the model element established from the influencer quantity *to* the model element established from the influenced quantity, regardless of the sign ($+$, $-$, or ?). The sign shall be important, however, for future work on synthesis of algebraic equations defining influenced model elements.

6.1.4 Evidence Classes of Model Elements

In our approach, as we know, model synthesis is substantiated by metadata evidence. This is done, in part, by finding descriptions of quantities in the metadata set that instantiate the quantity type terms in the association rules library (Section 6.1.2), and applying abduction over the rules to connect the quantities to model element types.

To establish a model element by that only, we need what we call *minimal evidence* or minimal constraint: a description of a quantity. But that only would be too soft a constraint, that would ultimately lead to a large number of weakly metadata-supported model solutions. Besides descriptions of quantities, we also use as *additional evidence* descriptions of properties of quantities (e.g., constancy), descriptions of relations between quantities (e.g., influence relations), and descriptions of model requirements (e.g., model material).

Additional evidence descriptions cannot be a hard requirement, though. The metadata sources may not cater to them. We make the system flexible by allowing model elements to be established either way, whether additional evidence holds or not. We define two evidence classes, *strong* and *weak*, and assign model elements to them accordingly: if full (minimal plus additional) evidence holds to that model element, it is said to be *strongly* suggested, otherwise, if only minimal evidence holds, it is said to be *weakly* suggested (for short, we also refer to the model elements as *strong* or *weak* elements).

Different metadata make additional evidence to different model element types. The *model material* requirement is additional evidence for establishing state variables; the *constancy* property for establishing parameters; and the *influence* relation for establishing intermediate variables. For driving variables, it is the absence of descriptions of the constancy and influence properties upon a quantity that constitutes additional evidence.

Assigning each model element's evidence class is integral to the heuristic synthesis rules (in Section 6.3). In establishing a model element through one of these rules, its evidence class is determined, according to availability or not of additional evidence. We then take advantage of this evidence classification of model elements to formulate integrity constraints that resolve possible conflicts between model element alternatives. Integrity constraints are introduced in the next section (Section 6.2).

Also, the evidence classes can be useful for users in evaluating model alternatives delivered by the synthesis system. For example, most likely, a user will prefer a synthesised model where a key quantity is, say, *strongly* suggested as an intermediate variable to one where it is *weakly* suggested as a state variable. Or, he may well take the second alternative, but he would do so knowing that the data set does not provide as strong an evidence for a state variable element type to hold that quantity, compared to an intermediate variable element type.

Partial plans, weak elements. Whole plans, strong elements.

We find a similar approach to our model elements evidence classes in the AI Planning literature. (Ferguson and Allen, 1994) proposes a model of plans based on defeasible argumentation, where plans that do not take all preconditions into account, so called *partial plans*, can be represented and reasoned upon. For every action with conjunctive preconditions there is a set of defeasible rules organised in a lattice of specificity — from the most specific where all preconditions appear, to the least specific with no preconditions. This renders degrees of support for actions (and plans, consequently) according to the preconditions' degree of specificity.

Our approach is a specialisation of this, where metadata evidence (preconditions) is always required (no evidence cases are not considered), with a bipartite lattice: within the set of constraints defined, full (most specific) metadata evidence suggests *strong* model elements, and minimal (less specific) metadata evidence suggests *weak* model elements.

Simplistic evidence class approach

Our dual evidence class approach is clearly very simple. A more sophisticated evidence ranking scheme (classification, valuation, etc.) could be elaborated, not restricted to just two evidence classes. Another strand for further elaboration would be on combination and propagation of evidence ranks. The assignment of evidence classes is restricted to atomic model substructures, the model elements. Evidence classes of connected elements could be somehow combined to assign an evidence class to the *flow* or *link* connecting them. Moreover, a framework for evidence ranking propagation throughout the connections in the model could be developed. Time did not allow our current research to explore this direction.

6.2 Integrity Constraints

As mentioned earlier, system dynamics models synthesised from metadata should not be uniquely determined. This motivates the use of abduction in the solving of metadata constraints because it allows for the synthesis of multiple model structure alternatives. Characteristic problems of abductive reasoning arise (Kakas et al., 1998), however: possible conflicts between explanations (the general term to what we call model substructures), and a potential large number of explanations. A common solution is to use *integrity constraints* to reduce the number of explanations and to handle those that may lead to inconsistencies. In our model synthesis context, the risk of inconsistency lies in conflicting synthesisable model substructures (mainly model elements) finding their way into the same model. Moreover, multiple possible model substructures may lead to a combinatorial explosion of model alternatives.

In general, we apply integrity constraints at the level of synthesis of individual model components, and at the level of synthesis of the overall model structure:

1. Based on domain knowledge, preference is given to certain types of model elements over others, depending on the roles the model element types involved play in the model, and also on the evidence classes the elements belong to. For example, if both a strong state variable and a strong intermediate variable are suggested by the same quantity, preference is given to the synthesis of the state variable, since this type of element is more central to the model than intermediate variables. More so, a strong state variable is also given preference over a weak intermediate variable. Generally, strong model elements are favoured over weak ones suggested by the same quantity. In other words, we take the practical stance of preferring model elements with full metadata evidence to the ones with partial evidence.

2. For cases of multiple substructure alternatives suggested by the same quantity that do not fall on the above, i.e., when one is not preferred to the other, the constraint is then to generate separate models for the alternatives. The preferences above keep the number of alternatives manageable. Clearly, this constraint cannot be defined at the level of individual model substructures, but at the level of assemblage of overall model alternatives. This goes in Chapter 7.

At this point, we will leave the discussion on integrity constraints at this general level. More detail on integrity constraints specific to each type of model element and connection is given throughout Section 6.3. Resolution of conflicts between the several model element types are summarised in Section 6.3.2.

6.3 Synthesis Constraints of Model Components

We will now see how metadata and integrity constraints are put together to define the synthesis constraints for each type of model component.

6.3.1 Constraints of Model Elements

The synthesis constraints of model elements are now discussed. Elements are the nodes in the model. They can be of types *state variable*, *intermediate variable*, *parameter*, *driving variable*, and *flow*, which is a hybrid element-connection type.

6.3.1.1 State Variables

State variables (also known as stocks, compartments or levels) represent stations of accumulation of material in the system-of-interest. For example, carbon in roots, leaves, branches and stems, in a model of carbon flow in a tropical forest; water in a reservoir, in a model of water storage; DDT in soil, rivers, ocean and air, in a model of accumulation of DDT in the environment; oil in reserves, in storage, in transit, in refinery stocks, etc., in a model of the world oil industry (Ford, 1999); profit in bay and ocean fisheries, in an economic model of fisheries management (Grant et al., 1997).

These examples make clear that a state variable represents a relation of containment between a material and distinguishable entities, or entity, that hold it (e.g., material carbon in entities roots, leaves, branches and stems).

We add name and unit of measure to material and entity to form the relation that represents state variables in their synthesis constraints. The relation is denoted $sv(\mathcal{E}(S), Mt, E, U)$, where

- $\mathcal{E}(S)$, with $\mathcal{E} \in \{stg,\ weak\}$, is the state variable's name S, qualified by its evidence class $\mathcal{E}$;
- Mt refers to a material;
- E refers to an ecological entity; and
- U refers to a unit of measure.

The constraints for state variables with strong metadata support are defined as follows.

> *S is a strongly suggested state variable, representing material Mt in entity E, with unit of measure U, if there exists a data term or model goal variable, Q_{term}, that can be associated, by abduction, with a $sv(S,Mt,E,U)$ term; if Mt is the model material; and if the integrity constraint for strong state variables is satisfied for S and U:*

$$
\begin{aligned}
& sv(stg(S),Mt,E,U) \leftarrow \\
& \quad \left.\begin{aligned}
& \exists Q_{term}\ .\ (data(Q_{term}) \vee model\_goal\_var(Q_{term})) \wedge \\
& \qquad\qquad association_{abd}(Q_{term}, sv(S,Mt,E,U)) \wedge \\
& model\_mat(Mt) \wedge
\end{aligned}\right\}\ \text{metadata constraints} \\
& \quad sv\_integrity(stg(S),U)
\end{aligned}
\tag{6.1}
$$

Recall that descriptions of quantities are to be found in a metadata set that contains descriptions of field data and/or model preliminary information (Chapter 5). Also recall that *data* is a generic descriptional predicate, used for any metadatum, except specific model requirements. $data(Q_{term})$ in Rule 6.1 thus means that Q_{term} can be broadly any of these generically described metadata.

$model\_goal\_var(Q_{term})$, on the other hand, has a more particular meaning in Rule 6.1. Model goal variables are identified from model objectives (Chapter 5). The rule encodes that in modelling practice the choice of state variables is often driven by model objectives. State variables usually are the variables whose behaviour the model is meant to simulate, by calculating the variables' values over simulation time. The diagrammatic structure of system dynamics models reflect the central role of this type of model element. All parameters, intermediate and driving variables in a model directly or indirectly regulate the flow variables, which, in turn, regulate the state variables. So, all the other variables in an model ultimately exist to determine the behaviour of the state variables.

The next condition in the rule is that an association between the described Q_{term} and a state variable term must take place. This is done by applying abduction over the association rules in Section 6.1.2 that relate descriptions of quantities to model components.

Next, that *Mt* must be a model material is the additional metadata evidence that assigns the *strong* evidence class to *S*. Our class of system dynamics models, as we know, represent flow of material through a system, with state variables representing storage stations of the material. Therefore, adding to the previous conditions, if there is a described quantity that can be associated with a state variable term concerning the material *Mt* known to be a model material (see Section 5.2.2) , then this *strongly* suggests a state variable.

This concludes the metadata constraints for strong state variables. Only the integrity constraint is now left. A note on notation, most other forthcoming model components rules[2] have this same structure: all except the last conjunct in the rule's premise make up the metadata constraints, with the last conjunct being the integrity constraint.

The integrity constraint for state variables is defined as:

> *The integrity of a state variable S of any evidence class (strong* or *weak) with unit of measure U is verified, if a strongly suggested parameter S with unit U does not hold:*

$$sv\_integrity(\mathcal{E}(S), U) \leftarrow \neg\, param(stg(S), Mt, U) \tag{6.2}$$

This is to say that a strong parameter is preferred to a state variable that originates from the same quantity. A single quantity, within a single model, cannot be mapped into multiple model elements of different kinds (Section 6.2). The parameter is preferred because, as we will see in Section 6.3.1.3, a quantity to become a strong parameter needs to be described as a constant. This is incompatible with a state variable behaviour, whose value is expected to fluctuate in simulation.

[2]The exception is Rule 6.8 which does not comprise an integrity constraint.

The unit of measure argument, U, is important in integrity verification. There only is a conflict between model elements when they originate from the same quantity, and a quantity's unit of measure is its most distinguishing attribute since it identifies the quantity's physical dimension (Section 3.2).

We should note that for the association rules library in Section 6.1.2, this integrity constraint is always satisfied, since there is no quantity type that can be associated with both a state variable and a parameter model element. Yet the integrity constraint is included for generality. We want to allow the library to be extended (possibly to include rules that would make possible the constraint not to hold) without changes needed in the integrity constraints.

A second rule defines weak state variables. The metadata sources may be short of information for identification of model material(s), in which case the additional evidence required for a strong state variable would be absent from the metadata set. This does not have to stop the state variable from being inferred, but this time, with less evidence, it will belong to the *weak* evidence class.

> *S is a weakly suggested state variable, representing some material Mt in entity E, with unit of measure U; if no model material is described, if there exists a data term or model goal variable, Q_{term}, that can be associated, by abduction, with a $sv(S,Mt,E,U)$ term; and if the integrity constraint for weak state variables is satisfied for S and U:*

$$
\begin{aligned}
& sv(weak(S), Mt, E, U) \leftarrow \\
& \quad \neg\ model\_mat(Mt')\ \wedge \\
& \quad \exists Q_{term}\ .\ (data(Q_{term})\ \vee\ model\_goal\_var(Q_{term}))\ \wedge \\
& \qquad\qquad association_{abd}(Q_{term}, sv(S, Mt, E, U))\ \wedge \\
& \quad sv\_integrity(weak(S), U)
\end{aligned}
\tag{6.3}
$$

6.3.1.2 Intermediate Variables

Commonly, *intermediate* (or *auxiliary*) variables are the most abundant element type in models. They are the model elements in between the variables representing inputs to the model — those outer most variables that affect but are not affected by other model variables — and the model's core, the chain of flows and state variables. State variables and flows aside, a model element is an intermediate variable if it is influenced by at least one other model element (i.e., there must be at least one *link* into it). This is their most distinctive structural characteristic. They usually are abundant in models because a complex network of these influences may be needed to represent the linkage between the model inputs and its core.

In the constraints rules, intermediate variables are represented by a relation of the form $iv(\mathcal{E}(I), Mt, U)$, similar to that of state variables. The rule for inference of strongly suggested intermediate variables is as follows.

> *I is a strongly suggested intermediate variable, in a model of flow of material Mt, with unit of measure U, if there exists a data term or model goal variable* Q_{term} *that can be associated, by abduction, with an* $iv(I,U)$ *term; if I is described as an influenced quantity; and if the integrity constraint for strongly suggested intermediate variables is satisfied for I, Mt and U:*

$$
\begin{aligned}
& iv(stg(I), Mt, U) \leftarrow \\
& \quad \exists Q_{term} \,.\, (data(Q_{term}) \vee model\_goal\_var(Q_{term})) \wedge \\
& \qquad\qquad association_{abd}(Q_{term}, iv(I,U)) \wedge \\
& \quad data(influences(Q, I, Sign)) \wedge \\
& \quad iv\_integrity(stg(I), Mt, U)
\end{aligned}
\tag{6.4}
$$

For state and intermediate variables only, Q_{term} can also be described with the *model_goal_var* predicate, i.e., quantities described as model goal quantities may become either state variables or intermediate variables in the models. Like state variables, intermediate variables "may represent an end product of calculations that is of particular interest to us" (Grant et al., 1997). In simulation, one is interested in finding out how things change over time due to processes and other relationships between entities in the real system. In the model, these entities and processes become variables and the changes of interest can be observed as fluctuations in variables' values over simulation time. Because intermediate variables are, by definition, influenced by other model elements, they can quantify some of these changes of interest — as the values of the influencers change during simulation so do the values of the intermediate variables. By contrast, we do not have model goal quantities leading to parameters, for example (Section 6.3.1.3). We are interested in quantities that change and parameters are constants.

Proceeding with the constraints, an association must hold between a data term Q_{term} in the metadata set and an intermediate variable term with name I and unit of measure U (via some rule in the association rules library).

Moreover, $data(influences(Q,I,Sign))$ is the additional piece of evidence that assigns I to the *strong* evidence class. Once more, evidence is to be found in the metadata set, where influences between quantities are described by the relation $influences(Influencer,Influenced,Sign)$. It can only be established that I is an influenced model element if the quantity it originates from is described as influenced — I is unified with the name of the quantity described in Q_{term} when proving $association_{abd}(Q_{term},iv(I,U))$ (see Section 7.4.2). This is symbolic of transferring the 'influenced' property from the quantity in the metadata to the model element I.

The integrity constraint for a strongly suggested intermediate variable is that it must not conflict with a strongly suggested state variable.

> *The integrity of a strongly suggested intermediate variable I, in a model of flow of material Mt, with unit of measure U, is verified, if a strongly suggested state variable I of Mt with unit of measure U does not hold:*

$$iv\_integrity(stg(I),Mt,U) \leftarrow \neg\, sv(stg(I),Mt,E,U) \tag{6.5}$$

That is, between establishing with strong evidence, within a model of flow of some material, a state variable and an intermediate variable with same name and unit of measure, we choose the former. The intuition is that state variables are more central to the model and should be given preference. Each single model is determined by the material argument *Mt* — each model represents flow of a single material through a system of interest (Section 5.2.2). This is the purpose of the *Mt* argument in the integrity constraint rule above. The material is not as intrinsic to intermediate variables as it is to state variables (which represent amount of material in an entity), but it is necessary for integrity verification.

As mentioned before, the distinctive structural characteristic of intermediate variables is that they must be influenced by at least one other model element. If there is no description in the metadata set providing evidence that the quantity the intermediate variable will hold is influenced, then the intermediate variable can only be weakly suggested.

> *I is a weakly suggested intermediate variable, in a model of flow of material Mt, with unit of measure U , if there exists a data term or model goal variable Q_{term} that can, by abduction, be associated with an $iv(I,U)$ term, but there is no metadata evidence that I is an influenced quantity, and if the integrity constraint for weakly suggested intermediate variables is satisfied for I, Mt and U*:

$$
\begin{aligned}
& iv(weak(I), Mt, U) \leftarrow \\
& \quad \exists Q_{term} \, . \, (data(Q_{term}) \vee model\_goal\_var(Q_{term})) \wedge \\
& \qquad\qquad association_{abd}(Q_{term}, iv(I,U)) \wedge \\
& \quad \neg\, data(influences(Q, I, Sign)) \wedge \\
& \quad iv\_integrity(weak(I), Mt, U)
\end{aligned} \tag{6.6}
$$

A weakly suggested intermediate variable must not conflict with a strongly suggested state variable nor with a strongly suggested parameter.

> *The integrity of a weakly suggested intermediate variable I, in a model of flow of material Mt, with unit of measure U, is verified, if neither holds a strongly suggested state variable nor a strongly suggested parameter with arguments I, Mt and U:*

$$
\begin{aligned}
& iv\_integrity(weak(I), Mt, U) \leftarrow \\
& \quad \neg\, (sv(stg(I), Mt, E, U) \vee param(stg(I), Mt, U))
\end{aligned} \tag{6.7}
$$

The integrity constraint encodes that strong evidence (of a state variable or parameter) is given preference over weak evidence (of an intermediate variable). Also, still applies the same reason for preferring strongly suggested state variables to strongly suggested intermediate variables — that the former are more central to the model,

In Rule 6.5 it was not necessary to check for parameters as in Rule 6.7 because the ontology already guarantees that influenced quantities do not become parameters (see Section 6.3.2).

6.3.1.3 Parameters

Parameters, or constants, represent characteristics of the system of interest that are considered unchanging under all conditions simulated by the model (Grant et al., 1997). Contrary to intermediate variables, they are not influenced by other model elements. For this reason they are also known as model inputs or exogenous variables (Ford, 1999).

Two concepts in Ecolingua directly relate to these characteristics of parameter model elements: the *constancy* property and the *influences* relation. The ontology defines that constants and influenced quantities are mutual exclusive (see Section 3.2.1.7). Metadata that violates this definition is rejected for model synthesis (Section 7.4.1).

In the synthesis constraints, parameters are represented by the relation $param(\mathcal{E}(P), Mt, U)$, in the same style of state and intermediate variables. The constraints for *strong* parameters are like this:

> *P is a strongly suggested parameter, in a model of flow of material Mt, with unit of measure U, if there exists a data term Q_{term} that can, by abduction, be associated with a $param(P,U)$ term, and if it is described as constant*:

$$
\begin{aligned}
& param(stg(P), Mt, U) \leftarrow \\
& \quad \exists Q_{term} \,.\, data(Q_{term}) \ \wedge \\
& \qquad\qquad association_{abd}(Q_{term}, param(P,U)) \ \wedge \\
& \quad data(constant(P))
\end{aligned} \tag{6.8}
$$

The description of P as constant is what makes for its strong class of evidence. Again, like the ‘influenced’ property for intermediate variables, this is to say that by proving

$association_{abd}(Q_{term}, param(P,U))$ the constancy property of the quantity described in Q_{term} is transferred to the model element P.

Note the absence of an integrity constraint. Integrity constraint 6.2 already rules out a conflict with a strong state variable. A conflict with a strong intermediate variable, in turn, is guaranteed not to take place by Ecolingua constraints. And finally, Rule 6.11 below that establishes strong driving variables makes them mutual exclusive in relation to strong parameters, in that it requires the originating quantity *not* to be described as constant (see Section 6.3.2 for a summary of conflict resolution between model elements).

If, however, a described quantity exists that can be associated with a $param(P,U)$ term but the evidence of constancy is missing, P is only suggested as a *weak* parameter.

> *P is a weakly suggested parameter, in a model of flow of material Mt, with unit of measure U, if there exists a data term Q_{term} that can, by abduction, be associated with a $param(P,U)$ term, if P is not described as constant, and if the integrity constraint for weakly suggested parameters is satisfied for P, Mt and U:*

$$\begin{aligned}
& param(weak(P), Mt, U) \leftarrow \\
& \quad \exists Q_{term} \, . \, data(Q_{term}) \; \wedge \\
& \qquad\qquad association_{abd}(Q_{term}, param(P,U)) \; \wedge \\
& \quad \neg \, data(constant(P)) \; \wedge \\
& \quad param_integrity(weak(P), Mt, U)
\end{aligned} \tag{6.9}$$

An integrity constraint is needed this time. Preference is given over weak parameters to state and intermediate variables supported by stronger metadata evidence. State variables are preferred because they are more central to the model, and intermediate variables because the description of an influence upon the originating quantity overrules the absence of a description of the quantity as constant.

> *The integrity of a weakly suggested parameter P, in a model of flow of material Mt, with unit of measure U, is verified, if it holds that P is neither a strong state variable nor an intermediate variable, also with arguments Mt and U:*

$$
\begin{aligned}
&\mathit{param\_integrity}(\mathit{weak}(P), Mt, U) \leftarrow \\
&\qquad \neg\, (sv(stg(P), Mt, E, U) \;\vee\; iv(stg(P), Mt, U))
\end{aligned} \tag{6.10}
$$

6.3.1.4 Driving Variables

Driving variables are similar to parameters in that they are also inputs to the model and are not influenced by other model elements. But they are not constants. They may change value during simulation due to factors other than other model elements, such as simulation time (Grant et al., 1997).

This renders a different use of metadata that relate to these characteristics of driving variables. Unlike intermediate variables and parameters, what counts for establishing a driving variable, is the absence of descriptions of the constancy and 'influenced' properties applied to quantities that suggest it. In other words, the lack of evidence is the evidence.

Similarly to the other model elements thus far, driving variables are represented by a relation of the form $dv(\mathcal{E}(D), Mt, U)$ in the synthesis constraints rules, which are formulated as follows.

> *D is a strongly suggested driving variable, in a model of flow of material Mt, with unit of measure U, if there exists a data term Q_{term} that can, by abduction, be associated with a $dv(D,U)$ term; if there is no description of D as constant, nor as influenced quantity; and if the driving variables integrity constraint is satisfied for D, Mt and U:*

$$
\begin{aligned}
&dv(stg(D), Mt, U) \leftarrow \\
&\qquad \exists Q_{term} \,.\, data(Q_{term}) \;\wedge \\
&\qquad\qquad association_{abd}(Q_{term}, dv(D,U)) \;\wedge \\
&\qquad \neg\, (data(constant(D)) \;\vee\; data(influences(Q, D, Sign))) \;\wedge \\
&\qquad dv\_integrity(stg(D), Mt, U)
\end{aligned} \tag{6.11}
$$

Driving variables can only be strongly suggested. To suggest a *weak* counterpart, we would have to rely on the existence of $data(constant(D))$ or $data(influences(Q,D,Sign))$. But these already suggest strong parameters and intermediate variables, respectively, which, upon conflict, would be given preference over a weak driving variable.

Driving variables may however conflict with strong state variables. The following integrity constraint gives preference to the latter.

> *The integrity of a strongly suggested state variable D, in a model of flow of material Mt, with unit of measure U, is verified, if a strongly suggested state variable S of Mt with unit U does not hold:*

$$dv\_integrity(stg(D),Mt,U) \leftarrow \neg\ sv(stg(D),Mt,E,U) \tag{6.12}$$

Like integrity constraint 6.2 (which resolves conflicts between state variables and strong parameters), this driving variables integrity constraint is bound to succeed given the current association rules library; yet it is included for generality.

6.3.2 Summary of (Non-)Conflicts between Model Elements

At various points throughout the previous sections on state variables, intermediate variables, parameters and driving variables, we have discussed conflicts between model elements. For better clarity, in Tables 6.2 to 6.5 below we summarise all possible conflicts and non-conflicts between pairs of model elements, each characterised by type (sv, iv, param, dv) and evidence class (stg, weak). Hereafter we will refer to them as just model elements. The tables show:

- **Conflicts resolved by preference defined by integrity constraints — $\Leftarrow$ and $\Uparrow$ cells.** Conflicting pairs of model elements occur in relation to a single metadata set and a single material. They may arise from descriptions of a single quantity (provided the metadata constraints are satisfied) but are not allowed to be both established. Rather, one is preferred (and the other rejected) as determined by an applicable integrity constraint. The arrows in the table point to the preferred element should a conflict occur.

- **Non-conflicts where the model elements are co-established — $\surd$ cells.** These are elements that arise from a single quantity, like before, but have no preference defined. Both can be established but in separate models. Synthesis of multiple model alternatives is discussed in Chapter 7.
- **Non-conflicts determined by underlying ontology constraints or other metadata constraints — $\times$ cells.** These are elements that dispense an integrity constraint, since either an ontological or other metadata constraint already prevents them both from being established.

		iv	
		stg	weak
sv	stg	$\Leftarrow$ (integ.)	$\Leftarrow$(integ.)
	weak	$\surd$	$\surd$

Table 6.2: State variables vs intermediate variables. Strong state variables are preferred to strong and weak intermediate variables (Section 6.3.1.2). Weak state variables can be co-established with strong and weak intermediate variables.

		param	
		stg	weak
sv	stg	$\Uparrow$ (integ.)	$\Leftarrow$ (integ.)
	weak	$\Uparrow$(integ.)	$\surd$

Table 6.3: State variables vs parameters. Strong parameters are preferred to strong and weak state variables (Section 6.3.1.1). Strong state variables are preferred to weak parameters (Section 6.3.1.3). Weak parameters and state variables can be co-established.

		param: stg	param: weak
iv	stg	× (onto.)	⇐(integ.)
	weak	⇑(integ.)	√

Table 6.4: Intermediate variables vs parameters. The Ecolingua constraints in Section 3.2.1.7 prevent strong intermediate variables and parameters from conflicting. Strong intermediate variables are preferred to weak parameters (Section 6.3.1.3) and strong parameters to weak intermediate variables (Section 6.3.1.2). Weak intermediate variables and parameters can be co-established.

		sv		iv		param	
		stg	weak	stg	weak	stg	weak
dv	stg	⇑(integ.)	√	×(md)	√	×(md)	√

Table 6.5: Driving variables vs state variables, intermediate variables and parameters. Strong state variables are preferred to strong driving variables (Section 6.3.1.4). Strong driving variables can be co-established with weak state variables, intermediate variables and parameters. Metadata constraints prevent conflicts between strong driving variables and strong intermediate variables and parameters (Section 6.3.1.4).

In the tables we can see that less pairs of elements can be co-established (7 in total) than cannot (11 in total). This has the desirable overall effect of restraining a combinatorial explosion of model alternatives. Remarkably, the metadata and integrity constraints have not been devised with this specific purpose in mind — they were primarily intended to just capture domain knowledge. The control over the number of model alternatives emerges from domain knowledge.

Also note that driving variables are more permissive on being co-established with weak model elements of other types. This is not surprising. Since strong driving variables and weak state variables, intermediate variables and parameters are all supported by the lack of certain metadata specifications, they are less exclusive model elements, and should be allowed co-establishment.

6.3.3 Constraints of Model Connections

Connections are the model's arcs, namely, flows and links, connecting model elements as nodes. Connections constraints are harder, in the sense that, besides their own, they also include the constraints of the model elements in the two ends. That is, for a flow or link to be established the constraints of the two model elements it connects must be satisfied.

In the synthesis rules, the premises for establishing connections comprise a conjunction of constraints, with, as indicated in Rule 6.13, the metadata constraints followed by the constraint(s) for establishing the initial model element and/or the terminal model element, followed, where appropriate, by the integrity constraint.

6.3.3.1 Flow Element-Connections

The storage of material, represented by the state variables in the models, is regulated by processes that transfer material throughout the system-of-interest. These processes are represented by *flows*. In the model diagrams, a flow pointing into a state variable represents a process that increases the amount of material in the state variable, whereas, a flow pointing away from a state variable represents a process that decreases its amount of material. A flow in between two state variables, hence, represents a process that has the effect of transferring material, causing a decrease of material in the state variable it leaves from and an increase in the state variable where it ends.

Flows are both model element and model connection. They are elements because like state variables, intermediate variables, driving variables and parameters, each has an equation to quantify it, giving the instantaneous flow rate over time. They are connections because they connect state variables forming the pathway over which the material of interest flows.

There are three kinds of flows: from a source into a state variable (an *in-flow*), from a state variable into a sink (an *out-flow*), or from a state variable into another. Sources and sinks are the boundaries of the 'flows-and-state variables' chain. They can also be thought of as state variables (although they are not modelled as such) that are outside of the model's scope (Ford, 1999).

We encapsulate all these features that constitute flows in a relation of the form $\mathit{flow}(F, Mt, From, To, U/U_t)$, where

- F is the flow's name;
- Mt is the material that the flow transfers;
- $From$ is the flow's origin, either a state variable or a source;
- To is the flow's destination, either a state variable or a sink; and
- U/U_t is the flow's unit of measure. Since a flow quantity is a rate of flow of some material over time, its unit of measure must be a composite of some unit U, measuring material, over a unit U_t, measuring time.

In-flows

F is a flow of material Mt from an entity E_{from} into a state variable $\mathcal{E}(S)$ with unit of measure U, and is measured in the unit of measure U/U_t, if there exists a data term Q_{term} that can, by abduction, be associated with a term representing transfer of Mt from entity E_{from} into entity E_{to}, also measured in U/U_t; if U_t is the model time unit; and if a state variable with unit of measure U cannot be established representing material Mt in entity E_{from}, but a state variable $\mathcal{E}(S)$ with the same unit can be established representing material Mt in entity E_{to}:

$$
\begin{aligned}
&\mathit{flow}(F, Mt, E_{from}, sv(\mathcal{E}(S), U), U/U_t) \leftarrow \\
&\quad \left.\begin{aligned}
&\exists Q_{term}\ .\ \mathit{data}(Q_{term}) \wedge \\
&\qquad\qquad \mathit{association}_{abd}(Q_{term}, \mathit{mattrans}(F, Mt, E_{from}, E_{to}, U/U_t)) \wedge \\
&\mathit{model\_time\_unit}(U_t) \wedge
\end{aligned}\right\} \text{metadata constraints} \\
&\quad \neg\ sv(S', Mt, E_{from}, U)\ \wedge \qquad\qquad \mathit{From}\ \text{constraint} \\
&\quad sv(\mathcal{E}(S), Mt, E_{to}, U) \qquad\qquad \mathit{To}\ \text{constraint}
\end{aligned}
\tag{6.13}
$$

This rule defines flows from a source into a state variable taking as starting ground a description of a quantity that can be associated with a matching material transfer term, through a rule such as Rule 7 in the association rules library (Section 6.1.2).

In this rule, an entity, E_{from}, rather than a state variable, is established as the flow's origin, because a state variable representing material *Mt* in E_{from} cannot be established. Since in system dynamics sources (and sinks) are not model elements as such, we just preserve the entity described at the metadata level as the origin of the process rate quantity that gives rise to the flow.

The units of measure are very important in this rule. In a system dynamics model the unit of flows, which is the same to all of them, consists of the unit of state variables, also unique, over the model's required time unit. The rule requires that the units U and U_t, that compose the unit of measure in the material transfer term, share with the state variables' unit and the model's time unit, respectively, giving the flow's unit U/U_t.

The underlying principle here is to preserve the same physical dimension. Strictly speaking, we could allow, for example, $mattrans(F, Mt, E_{from}, E_{to}, U/U_t))$, and $model\_time\_unit(U_t')$, with $U_t \neq U_t'$, as long as both units, U_t and U_t', were of the time dimension. Nevertheless, allowing units not to be the same but just ensuring that they share the same physical dimension would leave conversion between units of the same physical dimension — which is always possible — to the quantitative specification of the model. Since we address synthesis of conceptual models only, we make this simplification: the physical dimensions of units are guaranteed to be the same by way of requiring the units to be the same. That suffices as a proof of principle.

Besides descriptions of quantities, in-flows can also be established based on descriptions of influences between quantities. This is formalised in the next rule.

> *F is a flow of material Mt from the outside of the model's scope into a state variable $\mathcal{E}(S)$ with unit of measure U, and is measured in the unit U/U_t, if data exists of a positive influence of a quantity F into a quantity S; if U_t is the model time unit; if a state variable $\mathcal{E}(S)$ with unit of measure U can be established representing material Mt in some entity; and if the integrity constraint for flows from the outside is satisfied for F, Mt, $\mathcal{E}(S)$ and U:*

$$\begin{aligned}&\mathit{flow}(F,Mt,\mathit{outside},sv(\mathcal{E}(S),U),U/U_t) \leftarrow\\&\quad \mathit{data}(\mathit{influences}(F,S,+)) \ \wedge\\&\quad \mathit{model\_time\_unit}(U_t) \ \wedge\\&\quad sv(\mathcal{E}(S),Mt,E,U) \ \wedge\\&\quad \mathit{flow_{in}integrity}(F,Mt,\mathcal{E}(S),U/U_t)\end{aligned} \tag{6.14}$$

All the influence data says is that a quantity F positively influences a quantity S. In system dynamics, only flows can influence state variables. So, if a quantity F is described to influence a quantity S, and S proves to become a state variable in the model, then F is bound to be a flow. And since the sign of the influence is positive, S must be the flow's destination, i.e., F carries material into S.

On the other hand, the influence data does not hold any information that can be related to the origin of the flow. The constant *outside* is then used, provided that the flow F does not hold having a state variable as origin. This is part of the integrity constraint for flows to be established based on a positive influence data description. Compared to the previous in-flows rule (6.13), the metadata evidence used here is poorer and calls for checks in relation to conflicting model elements that may hold. The integrity constraint is formulated as follows.

> *The integrity of an in-flow F of material Mt into a state variable S, with unit of measure U/U_t, is verified if it does not hold that: F is a state variable of Mt with unit U; or F is a flow of Mt with unit U/U_t from a state variable with unit U; or F is a flow of Mt with unit U/U_t, from an entity E_{from} that is not 'outside' into a state variable S with unit U:*

$$\begin{aligned}&\mathit{flow_{in}integrity}(F,Mt,\mathcal{E}(S),U/U_t) \leftarrow\\&\quad \neg\left(\begin{array}{l} sv(\mathcal{E}(F),Mt,E,U) \vee\\ \mathit{flow}(F,Mt,sv(S',U),To,U/U_t) \vee\\ (\mathit{flow}(F,Mt,E_{\mathit{from}},sv(\mathcal{E}(S),U),U/U_t) \ \wedge \ \neg\, E_{\mathit{from}} = \mathit{outside})\end{array}\right)\end{aligned} \tag{6.15}$$

Model assumptions (Chapter 5) can give rise to descriptions of influences between two quantities which may both become state variables in the model. This happens when the process that causes the influence is implicit in the description. For example, in the assumptions for a pond management model, it could be said that the amount of harvested fish biomass increases with the biomass of fish accumulated in the pond. This could be described as $data(influences(fish\_biomass, harv\_fish\_biomass, +))$. In system dynamics models, however, influences between state variables are not directly represented. The *harvest* process would become explicit in the model as a flow connecting the two state variables.

The first condition in the integrity constraint above avoids an inconsistency that could arise from this kind of situation — for the example given, it avoids *fish_biomass* being established as a flow, when it can be more appropriately established as a state variable.

The other two conditions concern the flow's origin. Recall that the influence description holds no information that can be related to a flow's origin. But this lack of information does not rule out that the flow starts in a state variable. The constant *outside* will only be established as the flow's origin if a state variable cannot be established for it instead. Neither will *outside* hold, if some other more informative entity can be established instead. Although quantitatively it makes no difference if a source or sink is specified as *outside* or as a more specific entity such as *atmosphere*, the latter would be preferred for it adds qualitative information to the model.

Out-flows

We now show, more briefly, the constraints rules for 'state variable-to-sink' flows, or *out-flows*, drawing upon what we have already explained for in-flows.

The rationale of Rule 6.16 below is like Rule 6.13's except that this time a state variable holds for the flow's origin and does not for its end.

> *F is a flow of material Mt from state variable $\mathcal{E}(S)$ with unit of measure U into an entity E_{to}, and is measured in the unit of measure U/U_t, if there exists a data term Q_{term} that can, by abduction, be associated with a term representing transfer of Mt from entity E_{from} into entity E_{to}, also measured in U/U_t; if U_t is the model time unit; if a state variable $\mathcal{E}(S)$ with unit of measure U can be established representing material Mt in entity E_{from}; and if a state variable with the same unit U cannot be established representing material Mt in entity E_{to}:*

$$\begin{aligned}
&\mathit{flow}(F, Mt, sv(\mathcal{E}(S), U), E_{to}, U/U_t) \leftarrow \\
&\quad \exists Q_{term} \ . \ data(Q_{term}) \ \wedge \\
&\qquad\qquad association_{abd}(Q_{term}, mattrans(F, Mt, E_{from}, E_{to}, U/U_t)) \ \wedge \\
&\quad model_time_unit(U_t) \ \wedge \\
&\quad sv(\mathcal{E}(S), Mt, E_{from}, U) \ \wedge \\
&\quad \neg \ sv(S', Mt, E_{to}, U)
\end{aligned} \tag{6.16}$$

Similar to Rule 6.14, the next rule defines out-flows backed by an influence description. For out-flows the influence must be negative, since it is to support a flow 'taking material away' from a state variable.

F is a flow of material Mt from state variable $\mathcal{E}(S)$ with unit of measure U into the outside of the model's scope, and is measured in the unit U/U_t, if data exists of a negative influence of a quantity F into a quantity S; if U_t is the model time unit; if a state variable $\mathcal{E}(S)$, with unit of measure U, can be established representing material Mt in some entity; and if the integrity constraints for flows to the outside are satisfied for F, Mt, $\mathcal{E}(S)$ and U/U_t:

$$\begin{aligned}
&\mathit{flow}(F, Mt, sv(\mathcal{E}(S), U), outside, U/U_t) \leftarrow \\
&\quad data(\mathit{influences}(F, S, -)) \ \wedge \\
&\quad model_time_unit(U_t) \ \wedge \\
&\quad sv(\mathcal{E}(S), Mt, E, U) \ \wedge \\
&\quad \mathit{flow}_{out}integrity(F, Mt, \mathcal{E}(S), U/U_t)
\end{aligned} \tag{6.17}$$

And similarly to Rule 6.15 we have:

The integrity of an out-flow F of material Mt from a state variable $\mathcal{E}(S)$, with unit of measure U/U_t, is verified if it does not hold that: F is a state variable of Mt with unit U; or F is a flow of Mt with unit U/U_t into a state variable with unit U; or that F is a flow of Mt with unit U/U_t, from a state variable S with unit U into an entity E_{to} that is not 'outside':

$$\begin{aligned}
&\mathit{flow}_{out}integrity(F, Mt, \mathcal{E}(S), U/U_t) \leftarrow \\
&\quad \neg \begin{pmatrix}
sv(\mathcal{E}(F), Mt, E, U) \ \vee \\
\mathit{flow}(F, Mt, From, sv(S', U), U/U_t) \ \vee \\
(\mathit{flow}(F, Mt, sv(\mathcal{E}(S), U), E_{to}, U/U_t) \ \wedge \ \neg \ E_{to} = outside)
\end{pmatrix}
\end{aligned} \tag{6.18}$$

In-between-flows

Now we come to the constraint rules for flows in between state variables. Again, a similar rationale applies, except that state variables must be established for both ends of the flow.

> *F is a flow of material Mt with unit of measure U/U_t from a state variable $\mathcal{E}(S_1)$ with unit U into a state variable $\mathcal{E}(S_2)$ with unit U, if there exists a data term Q_{term} that can, by abduction, be associated with a term representing transfer of Mt from entity E_{from} into entity E_{to}, also measured in U/U_t; if U_t is the model time unit; and if state variables $\mathcal{E}(S_1)$ and $\mathcal{E}(S_2)$ with unit of measure U can both be established representing material Mt in entities E_{from} and E_{to} respectively:*

$$
\begin{aligned}
&\mathit{flow}(F, Mt, sv(\mathcal{E}(S_1), U), sv(\mathcal{E}(S_2), U), U/U_t) \leftarrow \\
&\quad \exists Q_{term} \;.\; data(Q_{term}) \;\wedge \\
&\qquad\qquad association_{abd}(Q_{term}, mattrans(F, Mt, E_{from}, E_{to}, U/U_t)) \;\wedge \\
&\quad model\_time\_unit(U_t) \;\wedge \\
&\quad sv(\mathcal{E}(S_1), Mt, E_{from}, U) \;\wedge \\
&\quad sv(\mathcal{E}(S_2), Mt, E_{to}, U)
\end{aligned}
\tag{6.19}
$$

Alternatively, a flow between state variables can also be established based on an influence description. Two influences are needed: a negative influence of the ‘flow-to-be’ quantity on the starting ‘state variable-to-be’ quantity S_1, and a positive one on the end ‘state variable-to-be’ quantity S_2.

> *F is a flow of material Mt with unit of measure U/U_t from a state variable $\mathcal{E}(S_1)$ with unit U into a state variable $\mathcal{E}(S_2)$ with unit of measure U, if data exists of a negative influence of a quantity F into a quantity S_1, as well as a positive influence of F into a quantity S_2; if U_t is the model time unit; if state variables $\mathcal{E}(S_1)$ and $\mathcal{E}(S_2)$ with unit of measure U can be established representing material Mt in some entities; and if the integrity constraint for flows in between state variables is satisfied for F, Mt and U:*

$$\begin{aligned}
&\mathit{flow}(F, Mt, sv(\mathcal{E}(S_1), U), sv(\mathcal{E}(S_2), U), U/U_t) \leftarrow \\
&\quad \mathit{data}(\mathit{influences}(F, S_1, -)) \wedge \mathit{data}(\mathit{influences}(F, S_2, +)) \wedge \\
&\quad \mathit{model_time_unit}(U_t) \wedge \\
&\quad sv(\mathcal{E}(S_1), Mt, E_1, U) \wedge \\
&\quad sv(\mathcal{E}(S_2), Mt, E_2, U) \wedge \\
&\quad \mathit{flow}_{between}\mathit{integrity}(F, Mt, U)
\end{aligned} \tag{6.20}$$

Having state variables in both ends simplifies the integrity constraint. Since both ends of the flow hold as state variables, no checks are necessary of a more informed source or sink than the generic *outside*. The only constraint is that F should not hold as a state variable.

The integrity of an in-between state variables flow F of material Mt, with unit of measure U, is verified if it does not hold that F is a state variable of Mt with unit U:

$$\mathit{flow}_{between}\mathit{integrity}(F, Mt, U) \leftarrow \neg\, sv(\mathcal{E}(F), Mt, E, U) \tag{6.21}$$

6.3.3.2 Link Connections

Links are another kind of connection in the models. While flows connect state variables, links connect other kinds of model elements, also in pairs and with a direction. Flows and links have very distinct roles in a model. Each flow represents a *material transfer* whose rate is defined by an equation; this is what makes flows model elements too. A link, in turn, represents *information transfer* between model elements and does not have an equation for itself. A link from, say, a model element A to a model element B denotes that B is a function of A. Quantitatively this means that values of A will inform, be used in, the calculation of values of B during simulation. The equation defining B will include a variable to hold values of A, as well as variables for all other elements linked into B.

In the synthesis rules, links are represented by the relation $link(Mt, Elem_{from}, Elem_{to})$, where

- *Mt* is the model's material;
- $Elem_{from}$ is the link's initial model element, restricted to *state variables*, *intermediate variables*, *parameters* and *driving variables*; and
- $Elem_{to}$ is the link's terminal model element, restricted to *flows and intermediate variables*.

The rules for links are unlike the rules of model elements in respect to the kind of metadata evidence predominantly used. Model elements represent quantities in the model, numerical values that are regulated by equations. Therefore, metadata to support their synthesis will consist mainly of descriptions of quantities. Links, on the other hand, do not represent quantities but functional relationships between them. Hence, metadata support for links will consist mainly of descriptions of influences between quantities.

We now show the constraint rules for links between each combination of initial ($Elem_{from}$) and terminal ($Elem_{to}$) model elements.

Links into flows

We start with a rule for inferring links from state variables to their flows. This kind of link is commonplace in system dynamics models. They represent the phenomenon where the amount of material the state variable stands for has an effect on the rate of its incoming and/or outgoing flows.

The metadata evidence used is a description of an influence of a quantity *S* into a quantity F. F must be established as a flow and *S* as either the state variable flow *F* leaves or the state variable it enters.

> *A link is established from a state variable element $\mathcal{E}(S)$ with unit of measure U to a flow element F in a model of flow of material Mt, if data exists of an influence of a quantity S into a quantity F; if a flow F of Mt from or to a state variable $\mathcal{E}(S)$ with unit U can be established; and if the indirect link integrity constraint is satisfied for Mt, $sv(\mathcal{E}(S), U)$, and F:*

$$\begin{aligned}
&link(Mt, sv(\mathcal{E}(S),U), flow_{id}(F)) \leftarrow \\
&\quad data(influences(S,F,Sign)) \ \wedge \\
&\quad (flow(F,Mt,sv(\mathcal{E}(S),U),To,U_f) \ \vee \ flow(F,Mt,From,sv(\mathcal{E}(S),U),U_f)) \ \wedge \\
&\quad link_{indirect}integrity(Mt,sv(\mathcal{E}(S),U),F)
\end{aligned} \tag{6.22}$$

Note that the state variable is established by way of establishing the flow, as prescribed in the flows' constraint rules showed previously.

The integrity constraint watches for an overlapping indirect link between, in this case, a state variable and a flow through an interposing intermediate variable. That is, the direct link is established only if a corresponding indirect link cannot be. The indirect link is preferred to the direct one because it is a more complex substructure that requires more metadata evidence, and we want to exploit metadata as much as possible.

Terminal elements of links are either intermediate variables or flows. Since we do not model flow-to-flow links, this fixes the interposing element as an intermediate variable in the integrity constraint 6.23 below.

> *The indirect link integrity constraint is satisfied for a link from a model element Elem to F in a model of flow of material Mt, if data of a specific rate R in relation to F with unit of measure U cannot be established, and that there can be a link from Elem to an intermediate variable R with unit U and a link from there to the flow F:*

$$\begin{aligned}
&link_{indirect}integrity(Mt,El,F) \leftarrow \\
&\quad \neg \left(\begin{array}{l} data(spf_rate(R,F,Mt,U)) \wedge \\ link(Mt,El,iv(\mathcal{E}(R),U)) \ \wedge \ link(Mt,iv(\mathcal{E}(R),U),flow_{id}(F)) \end{array} \right)
\end{aligned} \tag{6.23}$$

The interposing intermediate variable must originate from a quantity of type specific rate. As explained in Section 3.2, these rates are specified in relation to their absolute counterparts. Amongst the quantity types prescribed in Ecolingua, only specific rates can suggest an intrinsic regulating relation between an intermediate variable and a flow, if so they become in the model.

This integrity constraint is necessary because descriptions that support both a direct and an indirect link between a model element and a flow may derive from the metadata sources (Chapter 5). References can be made to influences of a quantity on both a process — that may become a flow — and a rate that regulates the process — that may become an intermediate variable. See, for example, in Appendix B, the descriptions of influences of aquatic plants biomass on both the consumption process and the consumption rate.

The next rule, analogous to Rule 6.22, defines the synthesis of a link from an intermediate variable to a flow based on an influence description.

> *A link is established from an intermediate variable element $\mathcal{E}(I)$ with unit of measure U to a flow element F in a model of flow of material Mt, if data exists of an influence of a quantity I into a quantity F; if an intermediate variable $\mathcal{E}(I)$ with unit U in the flow model of Mt can be established, as well as a flow F of Mt; and if the indirect link integrity constraint is satisfied for Mt, $iv(\mathcal{E}(I),U)$ and F:*

$$
\begin{aligned}
& link(Mt, iv(\mathcal{E}(I),U), flow_{id}(F)) \leftarrow \\
& \quad data(influences(I,F,Sign)) \ \wedge \\
& \quad iv(\mathcal{E}(I),Mt,U) \ \wedge \\
& \quad flow(F,Mt,From,To,U_f) \ \wedge \\
& \quad link_{indirect} integrity(Mt, iv(\mathcal{E}(I),U),F)
\end{aligned}
\tag{6.24}
$$

Recall that in the constraint rules for flows the units of measure of state variables and flows are required to harmonise. In Rule 6.22 for links between a state variable and a flow this also takes place by way of establishing the flow element. Rule 6.24 above is different in this respect. It does not encode any agreement between the units of measure of the intermediate variable and the flow (the flow's unit, fifth argument in the *flow* predicate, does not share). Other than linked state variables and flows, the units of measure of model elements at the two ends of links need not accord (Grant et al., 1997). This is so for links into both flows and intermediate variables.

The indirect link integrity constraint applies here in the same way as in Rule 6.22, except that this time the link goes from an intermediate variable to a flow.

Links between intermediate variables and flows can also be supported by quantity descriptions. The quantity description in the rule below suggests an intermediate variable as well as a link from it to a flow.

A link is established from an intermediate variable element $\mathcal{E}(I)$ with unit of measure U to a flow element F in a model of flow of material Mt, if there exists a data term or model goal variable Q_{term} that can, by abduction, be associated with an $iv(I,U)$ term and a $link(I,F)$ term; and if $\mathcal{E}(I)$ can be established as an intermediate variable with unit U, and F as a flow, in this flow model of Mt:

$$\begin{aligned}
& link(Mt, iv(\mathcal{E}(I),U), flow_{id}(F)) \leftarrow \\
& \quad \exists Q_{term} \,.\, (data(Q_{term}) \vee model_goal_var(Q_{term})) \wedge \\
& \qquad\qquad association_{abd}(Q_{term}, iv(I,U) \wedge link(I,F)) \wedge \\
& \quad iv(\mathcal{E}(I), Mt, U) \wedge \\
& \quad flow(F, Mt, From, To, U_f)
\end{aligned} \tag{6.25}$$

Like in the flow constraints, descriptions of quantities are a richer kind of metadata evidence than descriptions of influences for establishing links into flows, and dispense with the indirect link integrity constraint. A direct link into the flow is suggested straight from the quantity description term. This is firm support for the direct link and should prevail over a possible indirect one.

The next two rules, 6.26 and 6.27, are the 'parameter-to-flow' equivalent of rules 6.24 and 6.25 for 'intermediate variable-to-flow' links.

A link is established from a parameter element $\mathcal{E}(P)$ with unit of measure U to a flow element F in a model of flow of material Mt, if data exists of an influence of a quantity P into a quantity F; if a parameter $\mathcal{E}(P)$ with unit U in a flow model of Mt can be established, as well as a flow F of Mt; and if the indirect link integrity constraint is satisfied for Mt, $param(\mathcal{E}(P),U)$ and F:

$$\begin{aligned}
& link(Mt, param(\mathcal{E}(P),U), flow_{id}(F)) \leftarrow \\
& \quad data(influences(P,F,Sign)) \wedge \\
& \quad param(\mathcal{E}(P), Mt, U) \wedge \\
& \quad flow(F, Mt, From, To, U_f) \wedge \\
& \quad link_{indirect}integrity(Mt, param(\mathcal{E}(P),U), F)
\end{aligned} \tag{6.26}$$

A link is established from a parameter element $\mathcal{E}(P)$ with unit of measure U to a flow element F in a model of flow of material M, if there exists a data term Q_{term} that can, by abduction, be associated with a $param(P,U)$ term and a $link(P,F)$

term; and if $\mathcal{E}(P)$ can be established as a parameter with unit U, and F as a flow, in this flow model of Mt:

$$\begin{aligned} & link(Mt, param(\mathcal{E}(P), U), flow_{id}(F)) \leftarrow \\ & \quad \exists Q_{term} \,.\, data(Q_{term}) \;\wedge \\ & \qquad\qquad association_{abd}(Q_{term}, param(P, U) \;\wedge\; link(P, F)) \;\wedge \\ & \quad param(\mathcal{E}(P), Mt, U) \;\wedge \\ & \quad flow(F, Mt, From, To, U_f) \end{aligned} \tag{6.27}$$

And Rules 6.28 and 6.29 next, the ‘driving variable-to-flow’ equivalent.

A link is established from a driving variable element $\mathcal{E}(D)$ with unit of measure U to a flow element F in a model of flow of material Mt, if data exists of an influence of a quantity D into a quantity F; if a driving variable $\mathcal{E}(D)$ with unit U, as well as a flow F can be established in this model of flow of Mt; and if the indirect link integrity constraint is satisfied for Mt, $dv(\mathcal{E}(D), U)$ and F:

$$\begin{aligned} & link(Mt, dv(\mathcal{E}(D), U), flow_{id}(F)) \leftarrow \\ & \quad data(influences(D, F, Sign)) \;\wedge \\ & \quad dv(\mathcal{E}(D), Mt, U) \;\wedge \\ & \quad flow(F, Mt, From, To, U_f) \;\wedge \\ & \quad link_{indirect} integrity(Mt, dv(\mathcal{E}(D), U), F) \end{aligned} \tag{6.28}$$

A link is established from a driving variable element $\mathcal{E}(D)$ with unit of measure U to a flow element F in a model of flow of material Mt, if there exists a data term Q_{term} that can, by abduction, be associated with a $dv(D, U)$ term and a $link(D, F)$ term; and if $\mathcal{E}(D)$ can be established as a driving variable with unit U, and F as a flow, in this flow model of Mt:

$$\begin{aligned} & link(Mt, dv(\mathcal{E}(D), Mt, U), flow_{id}(F)) \leftarrow \\ & \quad \exists Q_{term} \,.\, data(Q_{term}) \;\wedge \\ & \qquad\qquad association_{abd}(Q_{term}, dv(D, U) \;\wedge\; link(D, F)) \;\wedge \\ & \quad dv(\mathcal{E}(D), Mt, U) \;\wedge \\ & \quad flow(F, Mt, From, To, U_f) \end{aligned} \tag{6.29}$$

Links into intermediate variables

The remaining synthesis rules are for links into intermediate variables. They all use an influence-between-quantities description as metadata evidence, and the link is established if the influencer quantity can be established as the link's initial model element, and the influenced quantity as the link's terminal model element.

A link is established from a state variable element $\mathcal{E}(S)$ with unit of measure U_s to an intermediate variable element $\mathcal{E}(I)$ with unit U_i in a model of flow of Mt, if exists data of an influence of a quantity S into a quantity I; if a state variable $\mathcal{E}(S)$ of material Mt with unit U_s can be established, as well as an intermediate variable $\mathcal{E}(I)$ with unit U_i, in this flow model of Mt:

$$\begin{aligned} & link(Mt, sv(\mathcal{E}(S), U_s), iv(\mathcal{E}(I), U_i)) \leftarrow \\ & \quad data(influences(S, I, Sign)) \ \wedge \\ & \quad sv(\mathcal{E}(S), Mt, E, U_s) \ \wedge \\ & \quad iv(\mathcal{E}(I), Mt, U_i) \end{aligned} \tag{6.30}$$

A link is established from an intermediate variable element $\mathcal{E}(I_1)$ with unit of measure U_1 to another intermediate variable element $\mathcal{E}(I_2)$ with unit U_2 in a model of flow of material Mt, if exists data of an influence of a quantity I_1 into a quantity I_2; and if $\mathcal{E}(I_1)$ with unit U_1 and $\mathcal{E}(I_2)$ with unit U_2, can be established as intermediate variables in this flow model of Mt:

$$\begin{aligned} & link(Mt, iv(\mathcal{E}(I_1), U_1), iv(\mathcal{E}(I_2), U_2)) \leftarrow \\ & \quad data(influences(I_1, I_2, Sign)) \ \wedge \\ & \quad iv(\mathcal{E}(I_1), Mt, U_1) \ \wedge \\ & \quad iv(\mathcal{E}(I_2), Mt, U_2) \end{aligned} \tag{6.31}$$

A link is established from a parameter element $\mathcal{E}(P)$ with unit of measure U_p to an intermediate variable element $\mathcal{E}(I)$ with unit U_i in a model of flow of material Mt, if exists data of an influence of a quantity P into a quantity I; if a parameter $\mathcal{E}(P)$ with unit U_p, as well as an intermediate variable $\mathcal{E}(I)$ with unit U_i, can be established in this flow model of Mt:

$$
\begin{aligned}
& link(Mt, param(\mathcal{E}(P), U_p), iv(\mathcal{E}(I), U_i)) \leftarrow \\
& \quad data(influences(P, I, Sign)) \ \wedge \\
& \quad param(\mathcal{E}(P), Mt, U_p) \ \wedge \\
& \quad iv(\mathcal{E}(I), Mt, U_i)
\end{aligned} \tag{6.32}
$$

A link is established from a driving variable element $\mathcal{E}(D)$ with unit of measure U_d to an intermediate variable element $\mathcal{E}(I)$ with unit U_i in a model of flow of material Mt, if exists data of an influence of a quantity D into a quantity I; if a driving variable $\mathcal{E}(D)$ with unit U_d can be established, as well as an intermediate variable $\mathcal{E}(I)$ with unit U_i, in this flow model of Mt:

$$
\begin{aligned}
& link(Mt, dv(\mathcal{E}(D), U_d), iv(\mathcal{E}(I), U_i)) \leftarrow \\
& \quad data(influences(D, I, Sign)) \ \wedge \\
& \quad dv(\mathcal{E}(D), Mt, U_d) \ \wedge \\
& \quad iv(\mathcal{E}(I), Mt, U_i)
\end{aligned} \tag{6.33}
$$

6.3.4 Summary of Model Connections

As we did for model elements, we now provide a model connections summary in schematic form for ease of reference. This time, however, the emphasis is not as much on conflicts, since not many occur between connections.

Model connections are:

- **Flows**, which can be:
  - **In-flows** (from source to state variable). Their synthesis is based on:
    - a description of a **quantity**; or
    - a description of a **positive influence** between quantities. In this case, the **source** is the *outside* constant, and

      · an **integrity constraint** gives preference (over the in-flow) to the following model components originating from the same quantity:
        - a state variable; or
        - a flow from a state variable; or
        - a flow from an entity.

  – **Out-flows** (from state variable to sink). Their synthesis is based on:
    ∗ a description of a **quantity**; or
    ∗ a description of a **negative influence** between quantities. In this case, the **sink** is the *outside* constant, and
      · an **integrity constraint** gives preference (over the out-flow) to the following model components originating from the same quantity:
        - a state variable; or
        - a flow into a state variable; or
        - a flow into an entity.

  – **In-between-flows** (from state variable to state variable). Their synthesis is based on:
    ∗ a description of a **quantity**; or
    ∗ a description of a **positive influence** and a description of a **negative influence** between quantities. In this case:
      · an **integrity constraint** gives preference (over the in-between-flow) to a state variable originating from the same quantity.

And

- **Links**,which can be:
  – **Links into flows**
    ∗ from **state variables**. Their synthesis is based on:
      · a description of an **influence** between quantities.
        - an **integrity constraint** gives preference (over the direct link from the state variable to the flow) to an indirect link via an intermediate variable.
    ∗ from **intermediate variables**, **parameters** and **driving variables**. Their synthesis is based on:
      · a description of a **quantity**; or

        · a description of an **influence** between quantities. In this case:
            - an **integrity constraint** gives preference (over the direct link from the intermediate variable, parameter or driving variable to the flow) to an indirect link via an intermediate variable.

- **Links into intermediate variables**
    * from **state variables**, **intermediate variables**, **parameters** and **driving variables**. Their synthesis is based on:
        · a description of an **influence** between quantities.

Chapter 7

The Synthesis-0 System

We call Synthesis-0 the first of the two model synthesis systems we have developed. The current chapter is dedicated to such system, with emphasis on the module that executes the synthesis task, the **synthesis mechanism**. Ecolingua (Chapter 3 and Appendix A), Metadata (Chapter 5), and Model Components Constraints (Chapter 6) are re-addressed as the other three modules in Synthesis-0, and their relations to the synthesis mechanism are explained.

7.1 System Architecture

Figure 7.1 shows Synthesis-0's architecture in terms of knowledge modules and their interrelations. Ecolingua, a Metadata Set and the Model Components Constraint rules are the requisite knowledge modules for the synthesis mechanism, which manipulates such knowledge to yield model solutions.

Quickly recapitulating, users describe given ecological data through the Ecolingua vocabulary, creating a metadata set; the descriptions in the metadata set are then used as evidence to support establishing model components through constraint rules that represent heuristic modelling knowledge.

What we call the synthesis mechanism encapsulates algorithms as well as reasoning

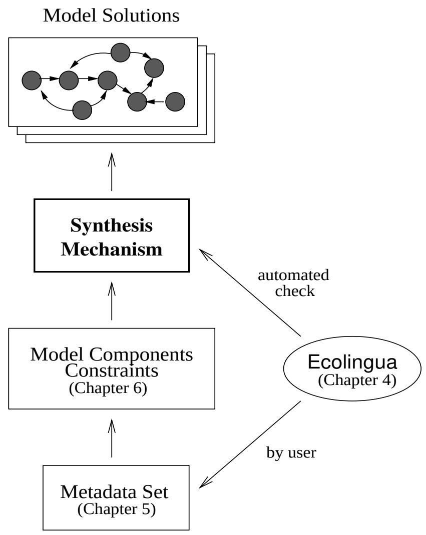


Figure 7.1: The architecture of the Synthesis-0 system.

patterns, techniques and mechanisms. We start to describe it in Section 7.2 showing the synthesis algorithms that put model components together to form full, possibly multiple, model structure solutions. Section 7.3 addresses the multiplicity of model solutions. Switching to reasoning patterns, in Section 7.4 we discuss the deductive and abductive patterns in the system as well as mechanisms built to carry out specific reasoning tasks. Section 7.5 illustrates the model synthesis mechanism with a worked example. And to conclude, in Section 7.6 we discuss the use of the meta-interpretation technique applied in Synthesis-0 and Synthesis-$\mathcal{R}$.

7.2 Algorithms

A connected[1] directed graph is an appropriate structure to represent a system dynamics model. The synthesis algorithms build graph structures of this kind, with metadata determining arcs and their nodes.

The arcs represent model connections (*flows* and *links*). The nodes represent model elements (*state variables*, *intermediate variables*, *parameters*, *driving variables* and

[1] A graph is connected when there is a path connecting every pair of nodes.

flows — flows are both connection and element (Section 6.3.3.1)) and also *sources* and *sinks* (see Section 2.4 to review system dynamics model structure).

Let us note that the algorithms are not general model synthesis algorithms, but tailored to system dynamics models, since our metadata-based synthesis approach is grounded in this domain. Hence, adopting a graph structure to represent models is particularly useful in that it adds some generality to the algorithms. Besides, it simplifies the explanation of the algorithms a great deal.

The *Top-level* Algorithm[2]

The top-level synthesis algorithm consists, quite simply, of establishing an initial node for the model, and starting from this *seed*, incrementally growing, or expanding the model until no further growth is possible; in other words, until the metadata support for establishing arcs (and their nodes) is exhausted. A synthesised model is specified by its arcs (flows and links) as detailed below.

Alg. 7.1 (Top-level) $synthesise0(\Delta, Model)$ — input: Δ; output: $Model$. Given a metadata set Δ, a model $Model$ is synthesised by:

$\underline{seed\_model}(\Delta, Mt, SeedNode)$ (Alg. 7.2), establish a material and a seed node for the model; and

$\underline{grow\_model}(\Delta, Mt, \{SeedNode\}, \{\}, Model)$ (Alg. 7.3), grow $Model$, having a set containing the seed node as initial set of nodes, and an empty set as initial model.

Where $Model$ is a set containing, if not empty, elements representing arcs of the forms (see Section 6.3 to review representation of model components):

- $flow(F, Node_{from}, Node_{to}, U)$, where
  - F is the flow's name;
  - $Node_{from}$ is the flow's initial node, which is either of the form:
    - $sv(\mathcal{E}(S), U)$ representing a *state variable*, with S being its name, $\mathcal{E}$ its evidence class, and U its unit of measure; or

[2] Calls to algorithms specified elsewhere appear underlined.

        - a constant representing a *source*
    - $Node_{to}$ is the flow's terminal node, which is either
        - a *state variable*, represented as above; or
        - a constant representing a *sink*
    - U is the flow's unit of measure.

and/or

- $link(Node_{from}, Node_{to})$, where
    - $Node_{from}$ is the link's initial node, which can be of one of the forms:
        - $iv(\mathcal{E}(I), U)$, representing an *intermediate variable*;
        - $param(\mathcal{E}(P), U)$, representing a *parameter*;
        - $dv(\mathcal{E}(D), U)$, representing a *driving variable*; or
        - a *state variable*, represented as above
    - $Node_{to}$ is the link's terminal node, which can be of the form:
        - $flow_{id}(F)$, representing a *flow*; or
        - an *intermediate variable*, represented as above

    where I, P, D and F, denote the model element's name; $\mathcal{E}$ its evidence class; and U its unit of measure.

The *Seed Model* Algorithm

The *seed_model* algorithm determines that seeds are state variables. This is also a domain-specific heuristic, like the ones in Chapter 6. It is inspired by system dynamics modelling practice (Haefner, 1996; Ford, 1999) where model design starts off by identifying state variables, since they are the key variables, representing where accumulation of material takes place in the system (Section 6.3.1.1). They usually are *model goal quantities* (Section 5.2.1), or, if not, directly influence them.

Support for establishing a seed, as for every node, consists of metadata specifications. The useful descriptions here are those of *amount* quantities, either physical or non-physical, since these are the quantities that suggest state variables, according to the library of Metadata↔Model association rules (Table 6.1).

Importantly, seeding the model also determines the *model material* (Section 5.2.2): the material the amount quantity refers to becomes the unique material to flow throughout the model solution, which, therefore, constrains the rest of the synthesis henceforth. The majority of system dynamics models used in practice represent flow of a single material. Models of flow of multiple materials are structured as interconnected submodels, one to each material. Synthesis-0 is able to synthesise submodels as separate model solutions (see Section 7.3).

Alg. 7.2 $seed\_model(\Delta, Mt, sv(\mathcal{E}(S), U))$ — input: Δ; outputs: Mt, $\mathcal{E}(S)$, U. For a given metadata set Δ, a state variable model element $sv(\mathcal{E}(S), U)$ is established as a seed for a model of flow of material Mt. (A number of seeds may hold for the same Δ.)

Find in Δ:

A description $data(Q_{type}(S, Mt, E, U))$ **OR** $model\_goal\_var(Q_{type}(S, Mt, E, U))$
such that $Q_{type} \in PhysAmts$, **OR**

A description $data(Q_{type}(S, E, U))$ **OR** $model\_goal\_var(Q_{type}(S, E, U))$
such that $Q_{type} \in NonPhysAmts$ **AND** $nonphys\_mat(Q_{type}) = Mt$

Where:

- *PhysAmts* and *NonPhysAmts* are the sets of relations in Ecolingua prescribing amounts of physical and non-physical quantities, respectively; and
- $nonphys\_mat(Q_{type})$ is a function that gives the non-physical material intrinsic to Q_{type}.

In the current specification of Ecolingua the *PhysAmts* and *NonPhysAmts* sets are:

- $PhysAmts = \{amt\_of\_mat, mat\_dens\}$
- $NonPhysAmts = \{amt\_of\_money\}$, with $nonphys\_mat(amt\_of\_money) = money$.

The two relations in the *PhysAmts* set are comprehensive, encompassing quantities of the *mass* and $mass/length^n$ physical dimensions, respectively. As discussed in Chapter 3, Ecolingua is amenable to being extended to include concepts prescribing quantities of other physical dimensions such as *energy*, for example.

amt_of_money (amount of money) is illustrative of types of non-physical quantities. Each non-physical quantity type requires specific definition(s) (see Section 3.2.1.1).

The *Grow Model* Algorithm

Every time a node (seed and subsequent model elements) is added to the model graph, arcs (model connections) from and to it are sought for. This is done recursively with the same behaviour applying to each node, starting with the seed: all arcs from and to the node are found, those not yet in the model graph are added to it, and the newly introduced nodes are identified and included in the set of nodes from which the graph expansion will continue. The synthesis is through when this set of nodes becomes empty.

Alg. 7.3 $grow\_model(\Delta, Mt, NewNodes, Model, Model'')$ — inputs: Δ, Mt, $NewNodes$, $Model$; output: $Model''$. From the set of nodes $NewNodes$, $Model$ is grown into $Model''$.

IF $NewNodes \neq \{\}$ **THEN**

Select a *Node* from *NewNodes* leaving the remaining nodes *NewNodesR*

$\underline{new\_arcs}(\Delta, Mt, Node, Model, Model', NewNodesN)$ (Alg. 7.4)

$NewNodes2 = NewNodesN \cup NewNodesR$

$grow\_model(\Delta, Mt, NewNodes2, Model', Model'')$, keep growing $Model'$ from the updated set of new nodes *NewNodes2*

ELSE $Model'' = Model$, there is no new node to expand, model growth is finished.

Viewed as search, the algorithm will have a depth-first or breadth-first strategy depending on the position in the *NewNodes* set of the *Node* element selected in turn to be expanded, and on the order in which the elements of *NewNodesN* and *NewNodesR* are put together to form *NewNodes2*. We have it implemented with a depth-first search strategy — the first *Node* in *NewNodes* is selected each time, and *NewNodesN* elements are placed in front of *NewNodesR* elements (duplicates are removed). The deepening of a path stops when, for the current *Node*, the *new_arcs* function returns an empty *NewNodesN* set.

The search is exhaustive, since the model graph is expanded exhaustively and where multiple solutions exist all of them can be searched for (see Section 7.3). It is also guaranteed to terminate, since the metadata sets are finite and Alg. 7.6 does not allow search loops to occur.

The *New Arcs* Algorithm

Alg. 7.4 finds arcs connected to each current node, adds to the model the new ones, i.e., those not yet in the model (arcs connecting the current node to previously visited nodes will already have been synthesised and included) and returns the set of new nodes amongst those connected to the current node through the new arcs (it is possible to have a new arc connected to an "old" node, already included as part of a previously established arc).

New arcs are identified by first finding one of the current node's clusters, which includes all the arcs the node is part of, and then taking the subset of new arcs from the cluster set.

Alg. 7.4 $new\_arcs(\Delta, Mt, Node, Model, Model', NewNodesN)$ — inputs: Δ, *Mt*, *Node*, *Model*; outputs: *Model′*, *NewNodesN*. Given Δ and *Mt*, arcs (model connections) from and to *Node* (a model element) are found that are not yet in *Model*; *NewNodesN* is the set of model elements connected to *Node* (through the new arcs) that are not yet in *Model*; and *Model′* is *Model* with the new arcs added:

$\underline{find\_cluster}(\Delta, Mt, Node, Cluster)$ (Alg. 7.5)

IF $Cluster \neq \{\}$ **THEN**

$\quad\underline{add\_new\_arcs}(Cluster, Model) = (Model', NewNodesN)$ (Alg. 7.6)

ELSE $Model' = Model$

$\quad NewNodesN = \{\}$

The *Find Cluster* Algorithm

In Alg. 7.5 is where, as indicated, the heuristic rules in Chapter 6 for synthesis of model elements and connections come through (proving a connection rule involves proving the model element(s) rule(s) it connects — Section 6.3.3). Each arc connected to the current node is either a flow or link, established through one of the corresponding rules.

The algorithm first finds an exhaustive set of arcs connected to the node. Remember that the synthesis process is rooted in a data set and that a description of a single quantity from the data set can give rise to multiple model components (see Section 6.1.2). Because of that, in the exhaustive set of one node's arcs, there may be arcs connecting the node to multiple other nodes (model elements) originated from the same quantity. To include them all in the same model would lead to inconsistencies — a single quantity can only be held in one model by a single model element.

To ensure this, a subset we call a cluster is taken from the exhaustive set of the node's arcs, containing a single arc per originating quantity (source and sink nodes originate from entities rather than quantities; in the algorithm the term 'datum' is used to refer to both quantities and entities). On backtracking, the algorithm yields alternative clusters, if they exist, each one to compose a separate model (see Section 7.3).

Alg. 7.5 $\mathit{find\_cluster}(\Delta, Mt, Node, Cluster)$ — inputs: Δ, Mt, $Node$; output: $Cluster$. $Cluster$ is a set of arcs and respective connected nodes to $Node$. (There may be a number of mutually exclusive $Cluster$ sets.)

IF $Connecs$ is a non-empty set of elements of the form $(C, Node_c)$, where C is an arc and $Node_c$ is the model element connected to $Node$ through C, with each element determined by:

$connec(\Delta, Mt, Node, Node_c, C)$

CASE It is established:

$\mathit{flow}(F, Mt, Node, Node_{to}, U)$ (Rules 6.13, 6.14, 6.16, 6.17, 6.19, 6.20)

THEN $C = \mathit{flow}(F, Node, Node_{to}, U)$

$Node_c = Node_{to}$

$\mathit{flow}(F, Mt, Node_{from}, Node, U)$ (Rules 6.13, 6.14, 6.16, 6.17, 6.19, 6.20)

THEN $C = \mathit{flow}(F, Node_{from}, Node, U)$

$Node_c = Node_{from}$

$link(Mt, Node, Node_{to})$ (Rules 6.22, 6.24 - 6.33)

THEN $C = link(Node, Node_{to})$

$Node_c = Node_{to}$

$link(Mt, Node_{from}, Node)$ (Rules 6.22, 6.24 - 6.33)

THEN $C = link(Node_{from}, Node)$

$Node_c = Node_{from}$

THEN Find $Cluster \subseteq Connecs$ by:

Form the set $\mathcal{D}$ of datum δ, such that each δ is the quantity or entity each connected node $Node_c$ holds

Form the set $Cluster$ (of elements $(C, Node_c)$) such that it contains a unique $Node_c$ to each $\delta \in \mathcal{D}$

ELSE No arc is found, $Cluster = \{\}$

The *Add New Arcs* Algorithm

Lastly, the function shown in Alg. 7.6 is responsible for, given a cluster and the current partial model, adding to the model graph the new arcs in the cluster, and for returning the new connected nodes to be expanded.

One may wonder why the algorithm checks if $F \notin NewNodesN$ after having already checked that there is no flow named $F \in Model'$. Recall from Section 6.3.3.1 that flows are both elements and connections. The above check is necessary because the node F may have been established (as the terminal node of a link arc) without the arc F having yet been. Every synthesised model with a node F will also contain a flow F, since the node is only established if the flow holds. Yet, a model may have a flow F without the node F, if no link to such node exists.

Notice also the rejection (for inclusion in the current model) of the whole cluster when at least one flow or other connected element in the cluster holds a quantity that has already been established in the model as some other type of model element. This is so, firstly because the model element type of a quantity must be unique in each model (Section 6.2). Secondly, all the other arcs in the cluster can be rejected, because since the synthesis of arcs and clusters is exhaustive (Alg. 7.5), the nodes rejected in this instance will have been dealt with as part of the cluster already included.

Alg. 7.6 $add\_new\_arcs(Cluster, Model) = (Model', NewNodesN)$ — inputs: $Cluster$, $Model$; outputs: $Model'$, $NewNodesN$. Arcs in $Cluster$ not yet in the current model $Model$ are added to it to form $Model'$, and $NewNodesN$ is the set of newly added nodes.

Initialise $Model' = Model$ **AND** $NewNodesN = \{\}$

FOR Each $(C, Node_c) \in Cluster$
- **IF** Arc $C \notin Model$ **THEN**
  - **IF** C is a flow named F **THEN**
    - **IF** $Node_c$ is a source or sink **THEN**
      - **IF** There is no other flow named $F \in Model'$ **THEN**
        - Add C to $Model'$
        - **IF** $F \notin NewNodesN$ **THEN** Add F to $NewNodesN$
      - **ELSE** **STOP**, $Cluster$ rejected
        - $Model' = Model$
        - $NewNodesN = \{\}$
    - **ELSE** $Node_c$ is a state variable
      - **IF** There is no other flow named $F \in Model'$ **AND** The quantity $Node_c$ holds is not already held in $Model'$ by any other node of a different type **THEN**
        - Add C to $Model'$
        - **IF** $F \notin NewNodesN$ **THEN** Add F to $NewNodesN$
        - **IF** $Node_c \notin NewNodesN$ **THEN** Add $Node_c$ to $NewNodesN$
      - **ELSE** **STOP**, $Cluster$ rejected
        - $Model' = Model$
        - $NewNodesN = \{\}$
  - **ELSE** C is a link
    - **IF** The quantity $Node_c$ holds is not already held in $Model'$ by any other node of a different type **THEN**
      - Add C to $Model$
      - **IF** $Node_c \notin NewNodesN$ **THEN** add $Node_c$ to $NewNodesN$
    - **ELSE** **STOP**, $Cluster$ rejected
      - $Model' = Model$
      - $NewNodesN = \{\}$

7.3 Multiple Model Solutions

Firstly, let us make clear what we mean by model solution: a model solution corresponds to the value in *Model* after one execution of $synthesise0(\Delta, Model)$ (Alg. 7.1).

Synthesis-0 is able to synthesise a range of model solutions for one given metadata set. As mentioned earlier in Chapter 6, this is motivated by the ecological modelling domain, where, as it happens in modelling disciplines in general, one data set does not uniquely determine a model solution, as different interpretations of the data may yield different models.

Multiplicity of model solutions has been introduced in the previous chapter when we discussed the library of Metadata↔Model association rules (Section 6.1.2) and Integrity Constraints (Sections 6.2 and 6.3.2). The very source of multiple model solutions is the association, through the library, of single quantity types with multiple model element types (in an abductive fashion, as we shall soon see in Section 7.4.2). Integrity constraints are then used to resolve conflicts by giving preference to certain model elements over others and, where preferences are not defined and legitimate alternatives exist, to avoid them being over-generated.

In the previous chapter we showed how the integrity constraints work at the level of synthesis of individual model components; now, as promised, we highlight how multiple model solution come to being as the synthesis algorithms above assemble model components together.

In fact, a mechanism for synthesis of multiple solutions is not built-in in the algorithms. They are designed as if to yield a single solution. If more solutions are desired, we use backtracking in Prolog's execution mechanism to make Synthesis-0 search for them (Synthesis-0's implementation in Prolog is briefly discussed in Section 7.4). We can, however, identify the features in the algorithms that give rise to multiple solutions. Namely, they arise from multiple seeds and model materials, and multiple clusters, in combination.

Seeds and Model Materials

The *seed_model* algorithm (Alg. 7.2) is purposefully easily solvable. It is meant to comprehensively explore the metadata set in search of seeds, hoping that from at least one or a few of them, significant models can be grown. Thus, for a single metadata set, multiple seed elements (state variables) and/or model materials can be established.

As specified in Alg. 7.1 the *seed_model* algorithm is followed by the *grow_model* algorithm. Each solution of *seed_model* gives rise to one or more new models, depending on the occurrence or not of multiple clusters (see below).

The relationship between model solutions grown from distinct solutions of *seed_model* should be interpreted, by human expertise, in different ways, depending on whether each of the seeding solutions consists, in relation to the others, of a *new* material and/or a *new* state variable.

Importantly, it is assumed that the metadata set at hand describes the modelling data of a single system-of-interest (Chapter 5), therefore all synthesised models are somehow interrelated, even if the metadata set and/or Synthesis-0 are not able to capture such interrelations.

Also recall that the most distinguishing attribute of a model is its material (Section 5.2.2)— each model solution represents flow of a unique material.

If *seed_model* can establish multiple model materials, each **distinct material** will lead to a subset of solutions for what in system dynamics is called a **submodel —** system dynamics models that represent flow of more than one material are subdivided into submodels that interconnected (through intermediate variables) compose one overall model. Synthesis-0 is not equipped to interconnect submodels, each submodel is synthesised as a separate model solution. Also, the system does not avoid model elements of different types in separate model solutions holding a same quantity.

Now, for a **same material** *seed_model* may find **distinct state variable seeds**. The model solutions grown from them (all within the subset of solutions for a single submodel) may contain:

- Identical solutions. Suppose a model with state variables X and Y. The same model will be grown if either X or Y is taken as seed.
- Solutions that have at least one state variable in common. This means that these models, at least in part, represent the same portion of the system-of-interest, and therefore, should be interpreted as alternative solutions.
- Solutions that do not share state variables with any other. These are assumed to be disconnected parts of the same model. They occur when the metadata lack support for synthesis of components that would tie up the disconnected parts. Here again, there may be inconsistency of disconnected parts including model elements of different types holding the same quantity.

Clusters

For each seed and model material, Alg. 7.5 may then yield multiple *clusters* as model nodes are expanded if it is the case that there are quantities that suggest multiple types of model elements. That is, multiple clusters can be computed by the algorithm if there are multiple associations between a quantity and model elements in the heuristic rules level. Clusters are mutually exclusive, each one goes in a separate model solution.

7.4 Patterns and Mechanisms of Inference and their Implementation

The pattern of inference that applies over most knowledge specifications in Synthesis-0 is conventional deduction. The specifications have been written as to allow Synthesis-0 to infer model components, models, the validity of a metadata term, for example, as logical consequences of the theory they represent.

To be specific, deduction applies over: the model components synthesis constraints (Section 6.3); the synthesis algorithms (Section 7.2); the ontological checking mechanism (forthcoming Section 7.4.1.1), which makes use of Ecolingua axioms (Chapter

3); and, the meta-interpreter (forthcoming Section 7.6.1).

To perform the deductions, we employ the standard logic programming framework (Lloyd, 1993; Apt, 1997) with resolution by refutation as proof procedure over Horn clauses with negation under the closed world assumption. Evidently, Prolog is the natural choice of implementation language. SICStus 3.8.5 was the interpreter used, running on a Unix platform.

Note that the synthesis algorithms are presented in a procedural notation. We chose to do so due to the strong element of control in the algorithms, but in the implemented version of Synthesis-0, they too are specified as Horn clauses.

There is, however, one inference step in Synthesis-0 that is non-deductive. It occurs where metadata is connected to model structure via the association rules in Table 6.1. In Section 7.4.2 we argue that abduction is the inference pattern that best characterises the Metadata↔Model connections, and show the abductive mechanism built to establish them.

Connecting metadata to model structure through abduction is, however, the second step in the process of directly using metadata evidence to support establishing model components. The first step is to validate metadata with respect to Ecolingua constraints. For this purpose, a specialist (deductive) mechanism has also been built.

For the rest of the section we discuss these two inference steps, in precedence order, together with the mechanisms that execute them.

7.4.1 Ontological Checking: Establishing Ecolingua-Compliant Metadata

So far in this chapter we have focused on the mechanisms in Synthesis-0 that assemble components into models. Clearly, the system must also encompass prior mechanisms which establish the components in the first place. The foremost of these mechanisms examines metadata that feeds into the system, ensuring that only metadata that conform to the underlying ontology gets through. Ontological checking as explained here also applies to Synthesis-$\mathcal{R}$, the reuse-synthesis system presented in the next chapter.

Looking back at Synthesis-0's algorithms, metadata feeds into the synthesis mechanism through Alg. 7.2, where descriptions of amount quantities are retrieved, and through Alg. 7.5, where flows and links are established through the components heuristic rules in Chapter 6. But, only metadata that satisfy the applicable *ontological constraints* (Section 6.1.1), i.e., only the Ecolingua-compliant descriptions, are carried forward into the synthesis process.

Metadata terms are retrieved from the metadata set when *descriptional predicates* are solved. As shown in Chapter 5, the forms of these expressions are:

- $data(C_{term})$, where
  - *data* is the generic descriptional predicate that denotes descriptions of all quantities and their relations;
  - C_{term} is a ground term, resulting from instantiating some Ecolingua concept to describe a piece of ecological data;
- $model\_goal\_var(C_{term})$, $model\_mat(Mt)$ and $model\_time\_unit(U)$, the specific descriptional predicates that denote model requirements.

We define that when a descriptional predicate is solved, the metadata term it describes also holds, in that it exists as part of the metadata set. Thus, for example:

$$data(C_{term}) \ \rightarrow \ C_{term}$$

Now, for the described metadata term (which, from the above, is *true* if retrieved from the metadata set) to be established as ontologically valid, it must unify with one of the Ecolingua concepts and be proven over the corresponding axiom.

As defined in Section 3.1, Ecolingua axioms are written as:

$$Cpt \rightarrow Ctt$$

where *Cpt* is an Ecolingua concept and *Ctt* is the concept's interpretation constraint.

Because in the synthesis systems these axioms are only reasoned upon when a C_{term}

with logical value *true* unifies with *Cpt*, the use of the axiom can be reduced to solving *Ctt*, since its logical value alone will correspond to the logical value of the whole expression.

Therefore, each Ecolingua axiom is re-represented in the synthesis systems as a clause of the form:

$$c\_ctt(Cpt, Ctt)$$

which links concepts to their respective interpretation constraints. An additional gain is that the formula conforms to the Horn Clause representational system adopted throughout Synthesis-0 (and Synthesis-$\mathcal{R}$).

We can now define the Ecolingua compliance checking mechanism.

7.4.1.1 The Ecolingua Compliance Checking Mechanism

Let C_{term} be an instance of an Ecolingua concept *Cpt*. As defined by the Ecolingua axioms formula, C_{term} being true and unified with *Cpt* implies that the consequent constraint *Ctt* must be true. If, however, the concept in question is one that lacks an axiomatic definition (e.g., the Ecological Entity concept — see Section 3.2.2) it suffices to verify that C_{term} unifies with a defined Ecolingua concept. Hence, the mechanism in Figure 7.2 below. Later in Section 7.6 we show its implementation in connection with the meta-interpreter applied.

$$\begin{aligned}
&onto_check(C_{term}) \leftarrow \\
&\qquad c_ctt(C_{term},\ Ctt)\ \wedge\ Ctt \\
&onto_check(C_{term}) \leftarrow \\
&\qquad \neg\ c_ctt(C_{term},\ _Ctt)\ \wedge \\
&\qquad eco_concept(C_{term})
\end{aligned}$$

Figure 7.2: Ecolingua compliance checking mechanism.

Which checks are needed over each retrieved metadata term is determined by the descriptional predicate at hand:

- $data(C_{term})$ triggers $onto\_check(C_{term})$ — the ground concept term is checked;
- $model\_goal\_var(C_{term})$ triggers $onto\_check(model\_goal\_var(C_{term}))$ and $onto\_check(C_{term})$ — the descriptional predicate itself is also an Ecolingua concept (see Section 3.3), hence the two checks;
- $model\_mat(Mt)$ triggers $onto\_check(model\_mat(Mt))$ and $model\_time\_unit(U)$ triggers $onto\_check(model\_time\_unit(U))$ — only the descriptional predicates are Ecolingua concepts.

Satisfied the Ecolingua compliance check, a metadata term is established as ontologically valid, and is then fit to back the synthesis of model components. The next section is about how this is done in Synthesis-0.

7.4.2 Abduction: Connecting Metadata to Model Structure

A crucial inference step of the synthesis process is where metadata is associated with model structure. In fact, this has to be the nub of a system whose overall goal is precisely that, to synthesise model structure based on metadata. Such associations, as we know, are drawn between metadata, which is a set of declarative descriptions, and model structure, which is represented as a graph structure. Let us first look at the associations' granularity.

In system dynamics practice, quantities in a data set (assumed to be suitable for the model objectives and to have been processed for modelling) can be initially related to model components in the conceptual formulation stage, and later used to calibrate them in the quantitative specification stage. Shifting this practice to the Synthesis-0 system, each association is between a quantity description and a fragment of model structure, such as a single component (element or connection) in most cases. In some cases, descriptions that are meaningful enough to support so are associated with slightly more complex fragments, such as an element and a connection from or to it.

The next aspect of the associations between metadata and model structure to consider is the inference pattern that best suits them. As givens we have the metadata specifications, established through ontological checking (Section 7.4.1). Their role in the associations is to provide support, or evidence, from which model components are synthesised, in an exploratory hypothesis-formation fashion. As opposed to *implying* model components (in the logical implication sense), it is more appropriate to say that descriptions only *suggest* them. In other words, the associations are a kind of unsound inference. Even if an association is defined where a description D suggests a component C, C will not necessarily hold in every model given D. Moreover, the same description may suggest not only one but alternatives of components which should not all apply to the same model.

Clearly, deduction does not lend itself for formalising this kind of reasoning. Model components are not logical consequences of metadata descriptions. Also, where descriptions suggest alternative components, *all* the components, as deductive consequents, would necessarily hold, leading to inconsistent models.

In contrast, here we have a non-deductive reasoning pattern that is typically abductive.

Abduction is generally formulated as (Kakas et al., 1998; Flach, 1994):

$$T \cup E \models O$$

> *Given a theory T and an observation O that does not logically follow from T, an explanation (or evidence) E that might imply O is conjectured and added to T to allow for the entailment of O.*

Thus, an unsound rule of inference is in effect of the form $\dfrac{\begin{array}{l} E \rightarrow O \\ O \end{array}}{E}$.

(Kakas et al., 1998) classifies uses of abduction into *causal* and *non-causal*. The classic application where abduction has a causal interpretation is *diagnosis*. Hypotheses (*explanations*) are generated which are causes for observed symptoms or effects (*observations*). There are applications, however, where the relationship between explanations and observations is non-causal. *Default reasoning* is an example, where conclusions (*observations*) are explained by means of assumptions (*explanations*) that hold by default unless some contradiction takes place.

In our *model synthesis* application, the interpretation of abduction is also non-causal, namely, model components (*explanations*) are synthesised that are suggested by given metadata descriptions (*observations*).

Our use of abduction draws upon (Robertson et al., 1991), a compilation of logic-based approaches for ecological modelling, where a non-causal interpretation of abduction for model synthesis has been first proposed. Abduction is used in (Robertson et al., 1991) to bridge knowledge of ecological systems and structure of simulation models, which not necessarily belong to the system dynamics paradigm. In comparison, we use abduction to bridge more restricted forms of ecological knowledge and model structure, respectively, quantity descriptions (metadata) and system dynamics model components.

7.4.2.1 The Abductive Mechanism

Abduction takes place every time an $association_{abd}(Q_{term}, ModelFrag)$ constraint is verified as the model components synthesis rules are solved. The constraint is defined as follows, where Q_{term} is a quantity term, and $ModelFrag$ is a model fragment consisting of a single model component, K_0, or of a number of model components, which are represented as a conjunction of the form $K_1 \wedge \ldots \wedge K_n$:

$$\begin{aligned} &association_{abd}(Q_{term}, ModelFrag) \leftarrow \\ &\qquad abduce(Q_{term}, Ks) \ \wedge \ \forall K_i \, . \, K_i \in Ks \end{aligned} \tag{7.1}$$

The *abduce* predicate establishes the set Ks of abduced model components suggested by the quantity described in Q_{term}.

Figure 7.3 shows the Prolog specification of the mechanism in Synthesis-0 that carries out the abductive proofs. Having a separate, modular abductive mechanism allows us to plug-in different association mechanisms, possibly non-abductive, if wanted. In fact, that is what we do in Synthesis-$\mathcal{R}$, our second synthesis system (Section 8.5).

```
abduce(Qterm, KS) :-
    abduce(Qterm, [], KS).            % components so far initialised with []
abduce(true, KS, KS) :- !.
abduce((Ki, Kjs), KS1, KS) :- !,
    abduce(Ki, KS1, KS2),
    abduce(Kjs, KS2, KS).
abduce(Qterm, KS1, KS) :-
    assoc_rule((Qterm :- MFrag)),     % matching association library rule
    abduce(MFrag, KS1, KS).
abduce(Ki, KS, [Ki|KS]) :-
    abducible(Ki).

abducible(Ki) :-
    functor(Ki, T, _),                % T - abducible model component type
    member(T, [sv, iv, param, dv, mattrans, link]).
```

Figure 7.3: Prolog specification of the abductive mechanism in Synthesis-0.

Note that it is at this point that the library of Metadata↔Model association rules (Table 6.1) is used. The abducibles are the terms that represent model components suggested by ecological quantities. Again, reasoning over the association rules with abduction allows inferring alternative hypotheses of model fragments (suggested by a quantity), whilst the standard logical meaning of the association rules is preserved.

There is, however, one other point in Synthesis-0, the seeding algorithm (Alg. 7.2), where quantities feed into the model synthesis process not through the abductive mechanism. As mentioned before, the seeding procedure is a heuristic meant to tentatively establish seeds from where models might be grown. They are directly established from descriptions of amount quantities in a deductive fashion. But, at this stage a seed is not yet a model element. A seed only becomes a model element once a connection (flow or link) it is part of is established, involving the abductive mechanism. Therefore, it is sound to say that abduction applies to every instance where quantities are associated with model components.

7.5 Worked Example

To illustrate Synthesis-0 at work we now show part of an example model being gradually synthesised. We shall again refer to the familiar pond management system example, upon which we illustrated the process of metadata generation in Chapter 5, and specified the metadata set in Appendix B.

Given such metadata set, Synthesis-0 synthesises a set of model solutions including the model in Figure 2.2, depicted in systems dynamics diagrammatic notation. We will talk through the first states of the synthesis search tree, generated as top-level Alg. 7.1 is executed, that ultimately leads to that model.

The partial search tree is represented in Figure 7.4. Its nodes[3] represent intermediate model solutions, or synthesis states, in diagrammatic form (see symbols key in the figure). In its highest level are the model seeds established by Alg. 7.2. From one of the seeds descends a subtree with nodes representing successive states of model growth as specified by Alg. 7.3 — each node corresponds to the value of variable *Model* after one iteration of the algorithm.

In order to comment the example in a concise and easy-to-follow manner, we make a few **assumptions** and notational **simplifications**:

i. As we know, model components are established by solving synthesis constraints, formalised in the heuristic modelling rules in Chapter 6. It would be tedious to go over every constraint solved to establish each model component. Instead, we concentrate just on metadata constraints, as they illustrate best the metadata-supported synthesis task under discussion. Furthermore, we do not show every metadata constraint solved. For each synthesised model component, we show only the primary metadata description involved (possibly one among other suitable descriptions in the metadata set). Primary descriptions carry the most significant evidence. In the synthesis rules, they are the descriptions required first, followed, in the case of descriptions of quantities, by the abductive association

[3]Do not confuse search tree nodes we are referring to here with model nodes that represent model elements as in the synthesis algorithms.

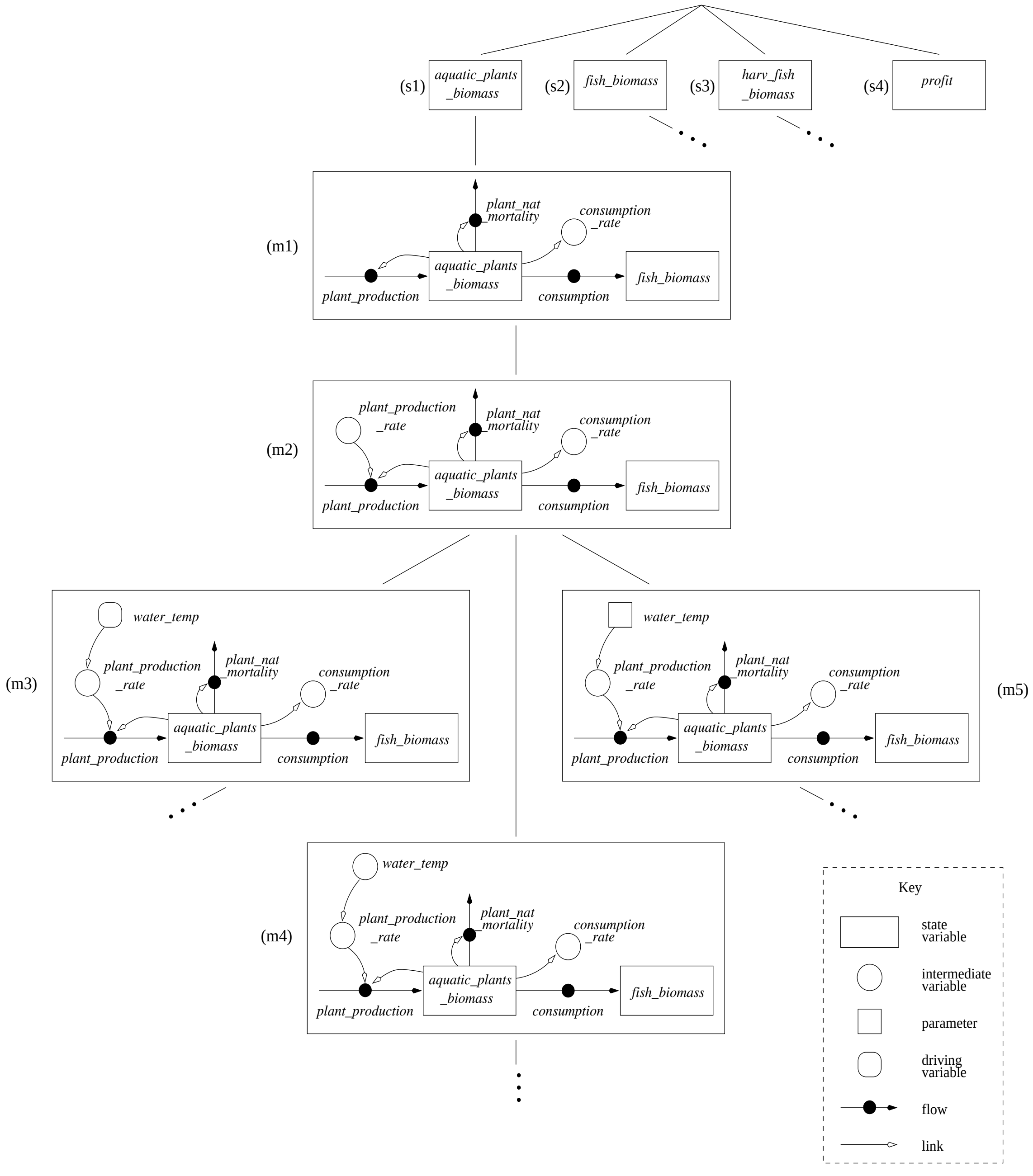


Figure 7.4: A portion of a Synthesis-0 search tree.

of the quantity with model component(s), in the way discussed in Section 7.4.2. For example, the primary description for the in-flows Rule 6.13 is the description of the quantity that is associated with the material transfer term (rather than the model time unit description or the further description(s) required to establish the in-flow's state variable).

ii. All descriptions retrieved from the metadata set are assumed to be checked for Ecolingua compliance successfully.

iii. Model components are denoted by type and name only. We omit from the components' specification: material, evidence class, unit of measure, as well as origin and destination in the case of flows. For example, a flow fully specified as $flow(plant\_production, biomass, outside, sv(stg(aquatic\_plants\_biomass), kg/ha), kg/ha/day)$ is denoted, for short, $flow(plant\_production)$; a state variable $sv(stg(aquatic\_plants\_biomass), kg/ha)$ is denoted $sv(aquatic\_plants\_biomass)$.

7.5.1 Tracing the Synthesis Search Tree

State (s1) Alg. 7.2 seeds a model with $biomass$ for model material, and with the seed node:

- $sv(aquatic\_plants\_biomass)$ based on the description

  $data(mat\_dens(aquatic\_plants\_biomass, aquatic\_plants, biomass, kg/ha))$.

As assumed, $onto\_check(mat\_dens(aquatic\_plants\_biomass, aquatic\_plants, biomass, kg/ha))$ is carried out successfully.

State (m1) $sv(aquatic\_plants\_biomass)$ is a fertile seed. The metadata set contains evidence that allows establishing a number of arcs connected to it (Alg. 7.4), namely (along with the primary descriptions their synthesis is based on):

- $flow(plant\_production)$ from

  $data(abs\_rate(plant\_production, outside, aquatic\_plants, biomass, kg/ha/day))$;

- $link(sv(aquatic\_plants\_biomass), flow(plant\_production))$ from
  $data(influences(aquatic\_plants\_biomass, plant\_production, ?))$;
- $flow(plant\_nat\_mortality)$ from
  $data(abs\_rate(plant\_nat\_mortality, aquatic\_plants, outside, biomass, kg/ha/day))$;
- $link(sv(aquatic\_plants\_biomass), flow(plant\_nat\_mortality))$ from
  $data(influences(aquatic\_plants\_biomass, plant\_nat\_mortality, ?))$;
- $flow(consumption)$ from
  $data(abs\_rate(consumption, aquatic\_plants, fish, biomass, kg/ha/day))$; and
- $link(sv(aquatic\_plants\_biomass), iv(consumption\_rate))$ from
  $data(influences(aquatic\_plants\_biomass, consumption\_rate, +))$.

To illustrate the use of the abductive mechanism, let us take for example the $flow(plant\_production)$ arc. It is established by solving the constraints in Rule 6.13, including:

$$association_{abd}\begin{pmatrix} abs\_rate(plant\_production, biomass, outside, aquatic\_plants, kg/ha/day), \\ mattrans(F, Mt, E_{from}, E_{to}, U/U_t) \end{pmatrix}$$

The constraint is solved as defined by expression 7.1, the subgoal to be solved by the abductive mechanism being:

```
abduce(abs_rate(plant_production,biomass,outside,aquatic_plants,kg/ha/day), KS)
```

which succeeds with:

```
KS = [mattrans(plant_production,biomass,outside,aquatic_plants,kg/ha/day)]
```

The six established arcs make up a single cluster (Alg. 7.5) and are all added to the model, since they are all new (Alg. 7.6), composing the partial model (m1).

State (m2) $flow(plant\_production)$ is the next node to be expanded (Alg. 7.3). Two arcs are found to be connected to it: $link(sv(aquatic\_plants\_biomass), flow(plant\_production))$, already included in the model, and:

- $link(iv(plant\_production\_rate), flow(plant\_production))$ based on
  $data(spf\_rate(plant\_production\_rate, plant\_production, biomass, g/kg/day))$.

Again, the arcs compose a single cluster. The new link is added to the model, composing the partial model (m2).

States (m3), (m4), (m5) $iv(plant\_production\_rate)$ is expanded next. Connected to it are found (Alg. 7.5) $link(iv(plant\_production\_rate), flow(plant\_production))$, already in the model, and:

- $link(dv(water\_temp), iv(plant\_production\_rate))$;
- $link(iv(water\_temp), iv(plant\_production\_rate))$; and
- $link(param(water\_temp), iv(plant\_production\_rate))$;

all based on the primary description $data(influences(water\_temp, plant\_production\_rate, ?))$.

Three clusters are formed out of the arcs found, one to each model element type holding the water temperature (*water_temp*) quantity (as in Table 6.5, a (strong) driving variable, a (weak) intermediate variable, and a (weak) parameter are non-conflicting, i.e., can be co-established in separate models).

The new arc in each of the clusters is added to the model, giving the three alternative partial models (m3), (m4) and (m5).

The synthesis process continues, as illustrated, with the expansion of other nodes (model elements). At this stage of the synthesis, in (m3) for example, the nodes yet to be expanded are $dv(water\_temp)$, $iv(consumption\_rate)$, $sv(fish\_biomass)$, $flow(consumption)$ and $flow(plant\_nat\_mortality)$. Of course, as specified in the model growth algorithm, the set of nodes to be expanded is updated as new arcs from and to new nodes are established. A fully synthesised model derived from (m3) (which would be a leaf node had the synthesis search tree been completed) appears in Figure 2.2.

States (s2), (s3) Other seed nodes the metadata set gives rise to, still with *biomass* for model material, are:

- $sv(fish\_biomass)$, based on $data(mat\_dens(fish\_biomass, fish, biomass, kg/ha))$; and
- $sv(harv\_fish\_biomass)$, based on $data(mat\_dens(harv\_fish\_biomass, harv\_fish, biomass, kg/ha))$.

The models grown from both these seeds are the same as those grown from $sv(aquatic\_plants\_biomass)$.

State (s4) The fourth seed node determines a different model material, namely $money$:

- $sv(profit)$ based on $model\_goal\_var(amt\_of\_money(profit, pond\_system, \$))$.

This turns out to be a sterile seed, however. Synthesis-0 is not able to find metadata evidence to grow models from it.

7.6 Meta-Interpreting Synthesis-0 (and Synthesis-$\mathcal{R}$)

Synthesis-$\mathcal{R}$ is our second synthesis system which will be presented in the next chapter. Meta-interpretation as discussed here also applies to that system.

Meta-interpretation is a theoretically simple yet powerful logic programming technique which gives access to the computation model of the programming language, making it amenable to modifications and/or augmentations to achieve desired functionalities of particular applications (Sterling and Shapiro, 1994; O'Keefe, 1990). In particular, Prolog's uniform representation of programs and data makes its meta-interpreters fairly straightforward to write, which contributes to the popularity of the technique.

We use meta-interpretation in Synthesis-0 and Synthesis-$\mathcal{R}$ for integration of special-purpose features, such as ontology compliance checking, without interweaving the features with the systems' heuristic rules or algorithms, which helps to keep the implementation modular and clear.

The meta-interpreter specified, which is shared by Synthesis-0 and Synthesis-$\mathcal{R}$, replicates the standard Prolog interpreter and adds to it **ontology compliance checking** as well as **caching**.

Caching is an efficiency-improving logic programming technique. Prolog programs are commonly slow to run due, in part, to unnecessary recomputation of uniquely satisfiable goals. Any goal in standard Prolog will be solved as many times as it is called. Clearly, this is inefficient if the goal in question has only one solution. In these cases, we can instead store the goal's solution when it is first solved and retrieve it whenever the goal is called again, rather than repeatedly performing the whole proof. This is the caching technique. As implied, the technique requires the uniquely satisfiable goals to be identified (which can be a tricky task) and stated in the program.

Our implementation of caching also draws on (Robertson et al., 1991), where the technique has been applied to reduce run time of Prolog-specified ecological simulation models.

7.6.1 The Meta-Interpreter

The Prolog specification of Synthesis-0's and Synthesis-$\mathcal{R}$'s meta-interpreter $solve(Goal)$ is given in Figure 7.5[4].

It is written on top of a standard meta-interpreter known as 'vanilla' (Sterling and Shapiro, 1994), which replicates Prolog's standard computation model applying goal reduction as an inference strategy. This is specified in clauses (1) to (3) and in part of (12) and (13) with `clause(X,B), solve(B)`. The calls to `clause(X,B)` is what actually interfaces the meta-interpreter with the program under interpretation, by unifying goals with the heads of clauses in the program and "handing over" their bodies for solution.

It is interesting to note that the goals of the abductive mechanism, of the form `abduce(Qterm, KS)`, are among those interpreted by these clauses in the meta-interpreter that replicate Prolog's computation model which is deductive. So, in effect, the abductive mechanism in Synthesis-0 is interpreted by Prolog's deductive mechanism.

[4]Figure 7.5 shows the meta-interpreter without cut handling, which has been implemented as in Robertson et al. (1991). The cut, denoted '!', is a Prolog control facility used to prevent unnecessary backtracking. Including the cut handling feature in the meta-interpreter shown here would lead the presentation astray from the other features we focus on.

```
(1)  solve((A,B)) :- !, solve(A), solve(B).
(2)  solve((A;B)) :- !, (solve(A); solve(B)).
(3)  solve(\+ X) :- !, \+ solve(X).
(4)  solve(X) :- meta_pred(X, M), !, M.
(5)  solve(X) :- builtin_pred(X), !, X.
(6)  solve(X) :- imported_pred(X), !, X.
(7)  solve(X) :- data_pred(X, T), onto_check(T).
(8)  solve(T) :- data_term(T), onto_check(T).
(9)  solve(T) :- model_req(T), onto_check(T).
(10) solve(T) :- model_req(T, Q), onto_check(T), onto_check(Q).
(11) solve(X) :- \+ (data_pred(X,_); model_req(X); model_req(X, _)),
                 unique_soln(X), lemma(X), !.
(12) solve(X) :- \+ (data_pred(X,_); model_req(X); model_req(X, _)),
                 unique_soln(X), !, clause(X, B), solve(B), !, add_lemma(X).
(13) solve(X) :- \+ (data_pred(X,_); model_req(X); model_req(X, _)),
                 clause(X, B), solve(B).
```

Figure 7.5: Prolog meta-interpreter shared by Synthesis-0 and Synthesis-$\mathcal{R}$.

Clause (4) handles meta-predicates. The `meta_pred/2` predicate redefines the goal `X` as `M` to avoid internal goals in `X` being directly interpreted by Prolog, rather than being meta-interpreted. For example, for the `setof/3` meta-predicate we have `meta_pred(setof(X, G, S), setof(X, solve(G), S))`.

Clauses (5) and (6) specify that built-in predicates and predicates imported from libraries, for example, must be interpreted directly by Prolog, since their clauses are not part of the program under interpretation.

The Ecolingua compliance checking feature figures in clauses (7) to (10). Only descriptional goals, i.e., goals that involve Ecolingua concept terms, are subject to ontological checking. The predicates that precede the call to the ontological checking mechanism, retrieve descriptions from the metadata set discriminating the terms in them to be checked against the ontology. `data_pred/2` succeeds for goals of the form *data*(`T`); `data_term/1` succeeds for goals which are described terms in the metadata set (the difference between this and the previous goal is that here the goal is the term `T` itself, rather than being of the form $data(\texttt{T})$); `model_req/1` succeeds for goals of the forms $model\_mat(Mt)$ and $model\_time\_unit(U)$; and, `model_req/2` succeeds for goals of the form $model\_goal\_var(\texttt{Q})$.

Now, the caching feature is implemented through clauses (11) and (12). If the goal is known to have a unique solution and a lemma for it exists (i.e., the solution has been already found and stored), then commit to the lemma; otherwise, if the goal is uniquely satisfiable but a lemma not yet exists, then try and find a solution and, if successful, add it as a lemma. `lemma/1` is a dynamic predicate, which allows clauses of it to be added to the program during execution.

And lastly, clause (13) is the most general clause which solves all goals that do not fall in any of the cases above.

For a complete synthesis process by Synthesis-0, the meta-interpreter is accessed via the call `solve(synthesise0(M))`, where `synthesise0/1` is the corresponding predicate to $synthesise0(\Delta, Model)$, Alg. 7.1, the top-level algorithm (the metadata set Δ is a consulted file).

To conclude, we show in Figure 7.6 the Prolog specification of the Ecolingua compliance checking mechanism described in Section 7.4.1, also shared with Synthesis-$\mathcal{R}$. Since it suffices to check each descriptional term just once, the mechanism in its implementation form is also enhanced with caching.

```
onto_check(X) :-
    onto_checked(X), !.
onto_check(X) :-
    c_ctt(X, Ctt),
    solve(Ctt), !,
    add_checked(X).
onto_check(X) :-
    \+ c_ctt(X, _),
    eco_concept(X),
    add_checked(X).
```

Figure 7.6: Prolog specification of the Ecolingua compliance checking mechanism.

Chapter 8

Reuse via Synthesis and the Synthesis-$\mathcal{R}$ System

Synthesis of model structure based on metadata is a problem that is difficult to constrain. A complete and precise characterisation of how data and models mesh does not seem attainable. Even if it were, modelling practice tells us that innumerable models can be designed supported by the same data, which makes the space of solutions infinite.

The metadata-only synthesis approach and the Synthesis-0 system that realises it (presented in the last two chapters) are reasonably successful in tackling the task as defined in Section 1.2, but do not appear to scale well. What is more, such success comes at the cost of substantial engineering effort in honing the synthesis constraints to tame model inconsistencies (Section 6.3).

As discussed before, Ecolingua and the Metadata$\leftrightarrow$Model associations library are amenable to extensions. Data sets more diverse than the ones we have been able to describe would require Ecolingua to encompass new concepts, and the associations library to encompass other rules to represent connections between these new kinds of data and model structure. Such extensions would enlarge the class of models synthesisable by the approach, but they alone cannot sustain the scalability of the approach; they take effort to construct, which means that we cannot increase Synthesis-0's scale indefinitely in this way.

If Synthesis-0 is to be of practical use, another factor relevant to the scalability of its synthesis approach is metadata volume, as this can have an impact on metadata search times, particularly in a domain such as ecology where large data sets are commonplace.

In Chapter 9 we show an empirical evaluation of Synthesis-0's run time performance under, of course, the current specifications of Ecolingua and the Metadata$\leftrightarrow$Model associations library. In the evaluation experiment we have used actual and fictitious metadata sets of sizes proportional to the sizes of the models synthesisable from them. Synthesis-0's run times appear to grow exponentially with model size.

Reuse-synthesis is a second, more parsimonious synthesis approach where such scalability issues raised by the metadata-only approach are alleviated. It was initially motivated by the idea of gaining more benefit from Ecolingua, or more generally from having an ontology at the foundation of the synthesis process, taking its use further than its traditional use as an unifying set of concepts for representing ecological data.

8.1 Reuse via Synthesis

The metadata-only approach lays down precise connections between ontological concepts as metadata, once they are instantiated to a data set, and model structure. In a nutshell, Synthesis-0 performs synthesis by exploring the space of such connections. This corresponds to the solid arrow in Figure 8.1 below.

The reuse-synthesis approach is about building models by drawing again these same connections but in a more controlled, efficient manner, by way of reusing existing models to guide the synthesis, as opposed to synthesising models "from scratch" supported by metadata only. We call such existing models *reference models*.

Reference models are not necessarily models synthesised by one of our systems (Synthesis-0 and Synthesis-$\mathcal{R}$), they can indeed also come from other sources, such as literature and experts. The value of reference models for synthesis lies in the modelling knowledge they encapsulate. We assume this is trustworthy knowledge because

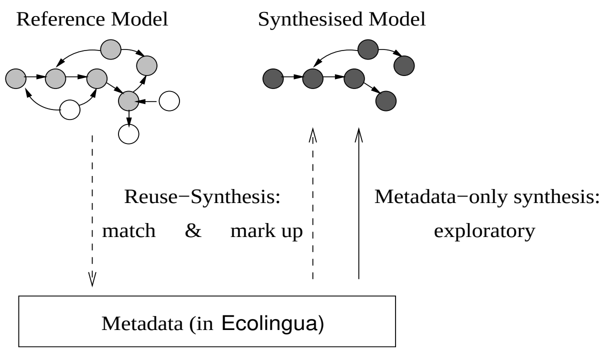


Figure 8.1: The metadata-only and the reuse-synthesis approaches.

the models to have been established, by a synthesis mechanism like ours or otherwise, have had to satisfy all pertinent constraints. One can envisage a library of such reference models made available for reuse.

Back to our Figure 8.1, notice the dashed arrows. They illustrate the twofold way of reuse-synthesis. The reference model's structure (or topology) works as a structural skeleton which is matched with the given metadata and then marked up, or filled in, wherever possible with content in the metadata to give a new model. To match is, in effect, to determine how much of the connecting structure between old data/metadata and corresponding (reference) model can apply to, or can find support in, the new metadata.

To do the matching we harness the same connections between metadata and model structure defined in the metadata-only approach, drawing them in the opposite direction (note in Figure 8.1 the directions of the 'Metadata to Synthesised Model' solid arrow and the 'Reference Model to Metadata' dashed arrow). It arises from the applicability of the same connections that by adapting the metadata-only synthesis algorithms we can obtain a useful mechanism to match reference models with metadata.

The heart of such adaptation is the reversal of the associations between metadata and model structure terms specified in Table 6.1. In the metadata-only approach, associations occur from metadata to model structure, with abduction applied to allow for that, as explained in Section 7.4.2. In the reuse-synthesis approach, in turn, associations occur from model structure to metadata — the synthesis is driven by the reference model structure, rather than by metadata, reversing the orientation of the associations, which become deductive. This change in inference pattern is discussed in Section 8.5.

The Ecolingua ontology underlies the synthesis process in both approaches, we should stress. In the metadata-only approach synthesis originates from the foundational ontological concepts, while in reuse-synthesis the reference structure is reduced back to them. In fact, having an ontology is the very factor that enables reuse of existing models (derived through whatever means), in that they can be levelled down to the standard concepts that inform the synthesis process.

Because Ecolingua is founded on universal notions of physical dimensions, a wide range of reference models can be reduced to its concepts (i.e., the degree of ontological commitment in Ecolingua is low (Gruber, 1995)). There are no specific requirements to be fulfilled by models used as reference structures (as long as they are system dynamics models in standard notation as exemplified in Section 2.4).

Synthesis-$\mathcal{R}$ is a working system where we realise the reuse-synthesis idea. It is presented in the remainder of this chapter, starting with a section on the system's architecture. The sections have been structured in line with Chapter 7. We omit a meta-interpretation section in the current chapter, as the two systems share the same meta-interpreter and Ecolingua compliance checking mechanism presented in Section 7.6.

Along the forthcoming sections we will see how Synthesis-$\mathcal{R}$ is able to lessen the scalability issues in the metadata-only approach we mentioned earlier. In Section 8.4 we show that even though the reuse-synthesis state space is still large, it can be pruned a little by using the reference models backbones as benchmarks. Section 8.6 shows how the synthesis constraints are made simpler and softer, bringing about an efficiency gain on metadata search, whereby metadata volume becomes less of an issue. And Chapter 9 shows Synthesis-$\mathcal{R}$'s much faster run times in comparison to Synthesis-0's.

8.2 System Architecture

The architecture of the Synthesis-$\mathcal{R}$ system is depicted in Figure 8.2.

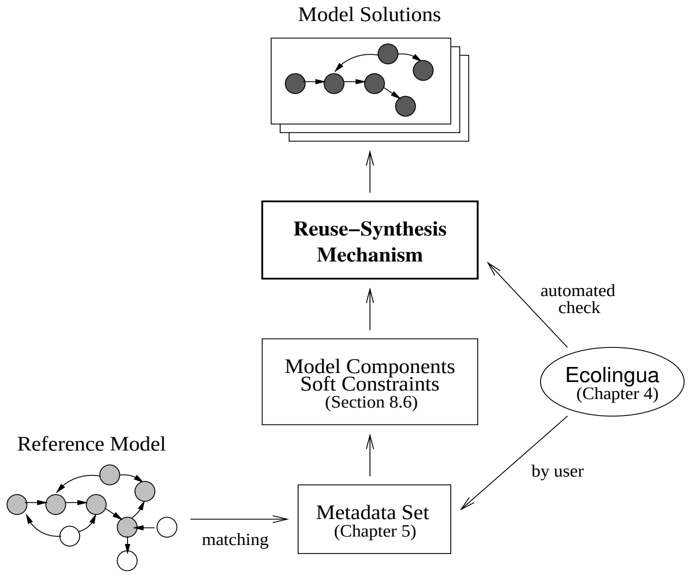


Figure 8.2: The architecture of the Synthesis-$\mathcal{R}$ system.

The Ecolingua, Metadata Set and Model Solutions knowledge modules are exactly the same as in Synthesis-0's architecture. As before, the metadata set consists of Ecolingua descriptions of data about the system-of-interest, and supports synthesis acting as evidence for satisfying constraints that establish model components. In addition to metadata evidence, the system is also resourced with an existing model whose structure is used as reference to guide the synthesis. The modelling knowledge encapsulated in the reference structure allows model solutions to be established through satisfaction of softer constraints in comparison with the metadata-only approach.

8.3 Algorithms

Synthesis-$\mathcal{R}$'s algorithms are many as well as more elaborate compared to Synthesis-0's. This is a trade-off. In the metadata-only approach most of the synthesis knowledge resides in the constraints, making the algorithms simpler and relatively more general.

In contrast, in reuse-synthesis most of the synthesis knowledge is embedded in a reference model structure, which makes the constraints simpler in turn. However, the system's algorithms become more complicated for they encode all the control needed to unravel the model structuring knowledge embedded in the reference model and pass it on to the new model, at the same time observing the metadata evidence given.

In Synthesis-$\mathcal{R}$'s algorithms, like in Synthesis-0's (Section 7.2), the general representation of models is that of directed graphs, only that this time the graphs may be disconnected (this is explained in Appendix C, the *traverse backbone and match* algorithm). Synthesis-$\mathcal{R}$'s algorithms are essentially a number of specialised procedures, each specifically designed to distill the model structuring knowledge contained in distinct sections of the graphs, characterised by different forms of flow and link arcs.

All possible forms of flows and links are depicted in Figures 8.3 and 8.4. In discussing the algorithms in the sequel we refer to these forms of flows and links, in the hope of facilitating understanding through visualisation. In Figures 8.3 and 8.4 and throughout the chapter we use the blanket term *converter* from (Ford, 1999) to refer to intermediate variables, parameters and driving variables.

The algorithms are hierarchically organised as shown in Figure 8.5. To avoid the chapter becoming too dense with all these algorithms, we focus on the higher and lower level ones, as highlighted in the figure, which are more fundamental to appreciating how the system works. The other intermediate algorithms are presented in Appendix C.

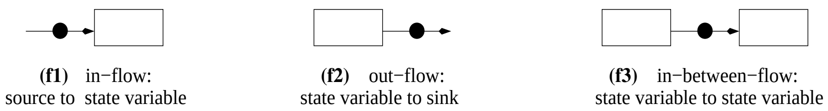


Figure 8.3: Forms of flows.

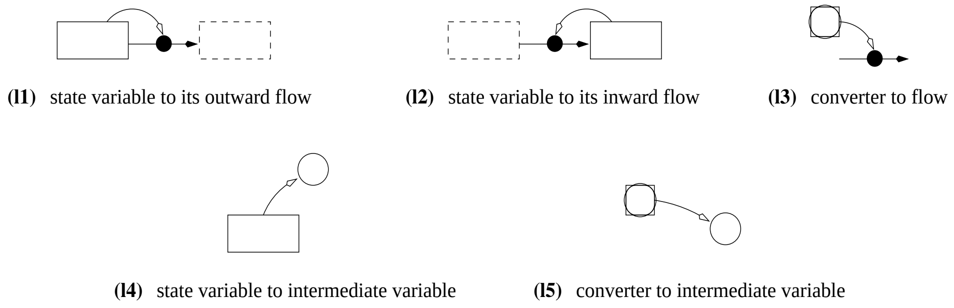


Figure 8.4: Forms of links.

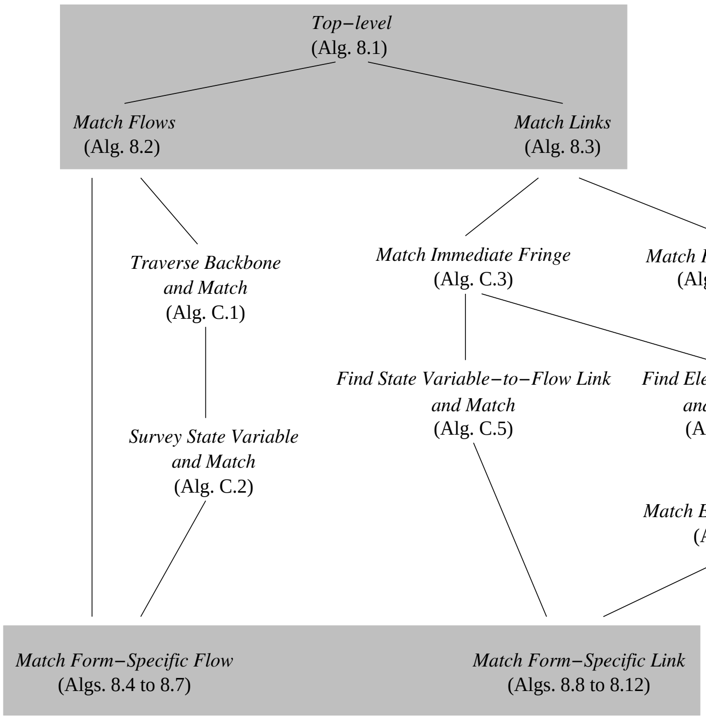


Figure 8.5: Hierarchy of Synthesis-$\mathcal{R}$'s algorithms.

The *Top-level* Algorithm

What fundamentally distinguishes Alg. 8.1 below from Synthesis-0's top-level algorithm is that besides a metadata set it has an extra synthesis resource: a reference model structure to follow. The algorithm traverses the reference model finding matches in the metadata for the model components it visits as it goes along. For each successful match a component of the same form is synthesised in the new model.

The traversal of the reference model and the synthesis of the new occur in parallel. The models are traversed/synthesised in a breadth-first fashion, starting with flow arcs and then carrying on with link arcs. One can view this as a centred graph structure (the centre being the flows and state variables chain(s)) whose boundaries advance from the centre outward. We shall hereafter refer to a model's chain(s) of flows and state variables as the model's *backbone*.

This is how the built-in structural knowledge in the reference model can be best exploited, from the most to the least specific types of model components. Flow arcs are the most constrained model components. Consequently, the metadata matching for them is equally constrained, bearing a more informed, and thus efficient, search procedure. Working from the flows outward, by the time the links are reached, which are more general, less constrained components, the already matched flows act as a core structure that anchors the rest of the synthesis.

Alg. 8.1 (Top-level) $synthesise\mathcal{R}(\Delta, M_{ref}, Model')$ — inputs: Δ, M_{ref}; output: $Model'$. Given a metadata set Δ and a reference model M_{ref}, a new model $Model'$ is synthesised through reuse of M_{ref} by:

$\underline{match\_flows}(\Delta, Flows_{ref}, \{\}, FlowsPairs)$ (Alg. 8.2), match the flows in M_{ref}, $Flows_{ref}$, with Δ, to give a set of matched flows, $FlowsPairs$, with elements of the form F_{ref}-F_{new}.

IF $FlowsPairs \neq \{\}$ **THEN**

$Model = \{F_{new} \ : \ F_{ref}\text{-}F_{new} \in FlowsPairs\}$

IF $Links_{ref} \neq \{\}$, where $Links_{ref}$ are the links in M_{ref} **THEN**

$\underline{match\_links}(\Delta, FlowsPairs, Links_{ref}, Model, Model')$ (Alg. 8.3), match $Links_{ref}$ with Δ, to give new links which are added to the current model $Model$ forming $Model'$.

ELSE $Model' = Model$

ELSE $Model' = \{\}$

Note that the algorithm only returns a non-empty $Model'$ when at least one successful flows match occurs.

The *FlowsPairs* set works as a record of matched flows. Each match is represented by a F_{ref}-F_{new} element in *FlowsPairs*. F_{ref}, we call *matched reference flow*, is a flow in the reference model which has been successfully matched with the metadata to synthesise F_{new}, we call *matched new flow*, a component of the new model $Model'$. Keeping a record of established matches serves the purpose of avoiding reference flows being re-matched with different new flows, which would corrupt the overall matching of the two structures (reference and new).

$Model'$ is a set that represents a system dynamics model structure in the same way as in Synthesis-0 (see Section 7.2), except that model elements are not qualified with evidence classes. The reason for this is explained in Section 8.6.

The algorithm is executed through the meta-interpreter in Figure 7.5 with the call `solve(synthesiseR(Mr,M))`, where `synthesiseR/2` is the corresponding predicate to $synthesise\mathcal{R}(\Delta, M_{ref}, Model')$, with the metadata set Δ being a consulted file.

The *Match Flows* Algorithm

The matching of flows starts by trying to determine the model's material (Section 5.2.2). If the metadata set contains one or more described model materials, then only models of flow of such materials are synthesised. Otherwise, the matching initialisation is called with an undetermined model material, represented by the *AnyMt* variable. In this case, the model material is determined by way of the first successful flow match, that is, whichever material involved in the metadata that match with the reference flow in question becomes the model material to constrain the synthesis all the way through.

The matching initialisation procedure establishes the first pair of matched flows through Algs. 8.4 to 8.7, one to each type of flow, namely, in-flows, out-flows and in-between-flows (either Alg. 8.6 or Alg. 8.7 could have been used, with the same effect, for in-between-flows). The flows in the reference model which are of the most common type with respect to each state variable are tried first for for the initial match. For example, suppose we have the model in Figure 2.2 as reference model. The first of its flows to be tried out for matching would be *respiration* and *excretion*. They are two out-flows, connected to the *fish_biomass* state variable, compared to at most one out-flow, in-flow, outward or inward in-between-flow connected to any of the state variables. The purpose of this is an efficiency gain. The reference flow already matched here will be excluded from the follow-up exhaustive matches by Alg. C.2 (see Section 8.4). Excluding a flow of one of the most common types (per state variable) can dramatically reduce the number of exhaustively found matches for the set of flows of that type.

Clearly, since state variable nodes are intrinsic to flow arcs, establishing a match between a reference flow and a new flow encompasses matching the flows' state variables as well. To an in-flows or out-flows match corresponds a single pair of matched state variables, while to an in-between-flows match corresponds two pairs of matched state variables. Alg. 8.2, in matching the initial pair of flows, establishes the initial pair(s) of matched state variables, which are then the starting point for the traversal/expansion of the models' backbones.

Flows connected to the reference state variable in the current pair are identified and matched with the metadata. Successful matches give rise to new state variable pairs from where the traversal/expansion process proceeds in the same fashion. This is detailed in Algs. C.1 and C.2. Algs. 8.4 to 8.7 are again referred to by Alg. C.2 to carry out the matching of reference flows with the metadata, according to the form of each flow in relation to the current reference state variable.

Alg. 8.2 avoids re-matches that would lead to spurious duplicated components in the new model, by verifying whether new flows and new state variables returned by the flows matching procedures have been previously matched. Reference state variables are also checked for this. It is necessary for reference state variables to be checked because they can be revisited, as they can be shared by more than one flow. Matched reference flows, on the other hand, are not revisited, they are removed from the set of reference flows once matched, and thus do not require to be checked for re-matches.

Alg. 8.2 $match\_flows(\Delta, Flows_{ref}, FPsf, FlowsPairs)$ — inputs: Δ, $Flows_{ref}$, $FPsf$; output: $FlowsPairs$. Match the reference flows in $Flows_{ref}$ with Δ, according to the form of each reference flow (Algs. 8.4 to 8.7), adding successful matches to the pairs of flows found so far, $FPsf$, giving the set $FlowsPairs$.

IF $\exists Mt \ . \ model\_mat(Mt)$ **THEN**

$initialise\_match(\Delta, Flows_{ref}, Mt, FPsf, FlowsPairs)$

ELSE $initialise\_match(\Delta, Flows_{ref}, AnyMt, FPsf, FlowsPairs)$

$initialise\_match(\Delta, Flows_{ref}, Mt, FPsf, FlowsPairs)$

From the most to the least common flow type in $Flows_{ref}$ wrt each state variable

IF $\exists F_{ref} \ . \ F_{ref} \in Flows_{ref}$ **AND**

$$\left(\begin{array}{l} \underline{match\_in\_flow}(\Delta, F_{ref}, Mt, S_{ref}\text{-}S_{new}, F_{new}) \text{ (Alg. 8.4)} \quad \textbf{OR} \\ \underline{match\_out\_flow}(\Delta, F_{ref}, Mt, S_{ref}\text{-}S_{new}, F_{new}) \text{ (Alg. 8.5)} \quad \textbf{OR} \\ \underline{match\_between_{in}\_flow}(\Delta, F_{ref}, Mt, S_{ref}\text{-}S_{new}, S1_{ref}\text{-}S1_{new}, F_{new}) \text{ (Alg. 8.6)} \end{array}\right)$$

AND There is no matched new flow in $FPsf$ with the same name as F_{new}

AND None of the newly matched state variables, reference or new, are already matched with other state variables in $FPsf$ **THEN**

$Flows_{ref} = \{F_{ref} \mid Flows_{ref}1\} \qquad FP1 = \{F_{ref}\text{-}F_{new} \mid FPsf\}$

$\underline{traverse\_bb\_n\_match}(\Delta, SVpairs, Flows_{ref}1, Mt, FP1, FlowsPairs)$ (Alg. C.1), starting from the pair(s) of newly matched state variables in $SVpairs$, traverse the reference model's backbone in $Flows_{ref}1$ (Alg. C.2), and match it with Δ (Algs. 8.4 to 8.7) to find new pairs of matching flows, which are added to the pair(s) found so far, $FP1$, giving $FlowsPairs$.

STOP

ELSE $FlowsPairs = FPsf$

The *Match Links* Algorithm

Once a backbone (one or more chains of flows and their state variables) is established for the new model, Alg. 8.3 attempts to expand it with a network of links. For the same reason that flows are synthesised first followed by links, in decreasing degree of constraintship, the algorithm starts with the network of links directly connected to flows and state variables, which we call the *immediate fringe*, and then continues with the outer network of links, the *far fringe*.

Of the link forms depicted in Figure 8.4, in the immediate fringe there can be:

- links from state variables into their flows (link forms (l1) and (l2));
- links from converters into flows (link form (l3)); and
- links from state variables into intermediate variables (link form (l4)).

Reference links of such forms are found (as detailed in Algs C.5 and C.6) and then matched with the metadata: links of form (l1) and (l2) by Alg. 8.8, links of form (l3) by Alg. 8.9, and links of form (l4) by Alg. 8.10.

The matching of the two latter forms of links introduces pairs of matched converters, collected in the *CvsPairs* set, from where the far fringe matching begins.

Pairs of converters are for the matching of links what pairs of state variables are for the matching of flows. They function as stepping stones from where proceed the traversing of the reference model's network of links and the expansion of the new model. Moreover, similarly to the *FlowsPairs* set, the record of established converters pairs in *CvsPairs* is also used to avoid inconsistent re-matches.

The far fringe is fairly uniform compared to the backbone and immediate fringe model sections. Again, it is this uniformity that makes this model section the least constrained of all, embodying relatively little structural information, being thus, the last to be synthesised in the reuse approach.

There is a single link form in the far fringe:

- links from converters into intermediate variables (link form (l5) of Figure 8.4)

since, by definition, links cannot end in parameters or driving variables.

Here again specific algorithms are used to match reference links of such form with the metadata once they are found (as detailed in Alg. C.6). Alg. 8.11 is used for outgoing links from the current reference converter, and Alg. 8.12, for incoming links to the current reference converter.

Not every new far fringe link established will as well establish a new pair of converters. The algorithm then keeps a log of expanded pairs to make sure only new pairs are selected for expansion, and to end the synthesis when no more new pairs can be established.

Alg. 8.3 $match\_links(\Delta, FlowsPairs, Links_{ref}, Msf, Model)$ — inputs: Δ, $FlowsPairs$, $Links_{ref}$, Msf; output: $Model$. Given Δ and the set of matched flows $FlowsPairs$, match $Links_{ref}$, the links of the reference model stemming from its backbone, with Δ, to establish links which are added to the new model so far, Msf, composed of flows only, to form a final new model $Model$.

$CvsPairs = \{\}$, initialise the current set of pairs of matched converters.

$\underline{match\_immediate\_fringe}(\Delta, FlowsPairs, Links_{ref}, Links'_{ref}, Msf, M, CvsPairs, CvsPairs')$ (Alg. C.3), find the links in $Links_{ref}$ composing the reference model's immediate fringe (Algs. C.5 and C.6) and match them with Δ according to the form of each reference link (Algs. C.7 and 8.8 to 8.10), with a view to synthesising the new model's immediate fringe, expanding the new model so far Msf into M, and giving a set of pairs of matched converters $CvsPairs'$. Links outside of the reference model's immediate fringe are returned in the set $Links'_{ref}$.

$CPlog = CvsPairs' \quad CPtoExpd = CvsPairs'$

$\underline{match\_far\_fringe}(\Delta, CvsPairs', CPlog, CPtoExpd, Links'_{ref}, M, Model)$ (Alg. C.4), find the links in $Links'_{ref}$ composing the reference model's far fringe (Alg. C.6) and match them with Δ according to the form of each reference link (Algs. C.7, 8.11 and 8.12), with a view to synthesising an outer network of links, which is added to the new model so far M to give a final model $Model$. The synthesis process ends when all the reference converters have been reached.

The *Match Form-Specific Flow* Algorithms

The actual matching of reference flows with the metadata set is carried out by Algs. 8.4, 8.5, 8.6 and 8.7, one to each flow form with respect to the current reference state variable (see their points of call in Algs. 8.2 and C.2).

A match succeeds if the new flow's reuse-synthesis constraints, formulated as an $\mathcal{R}$-rule (Section 8.6), are satisfied over the metadata set.

In the algorithms, the S_{new} variable, which represents the new state variable that matches the current reference state variable, may or may not be ground, meaning that the state variable may or may not have already been established in the new model. It will not have been when the match is the initial one (Alg. 8.2), and it will when the flows of a particular established pair of matching state variables are being surveyed (Alg. C.2).

Alg. 8.4 $match\_in\_flow(\Delta, F_{ref}, Mt, S_{ref}\text{-}S_{new}, flow(F, E_{from}, sv(S_{new}, U), U/U_t))$ — inputs: Δ, F_{ref}; inputs/outputs: Mt, S_{ref}, S_{new}; outputs: F, E_{from}, U, U_t. F_{ref} is an in-flow into the state variable S_{ref} (flow form (f1) of Fig. 8.3) **IF**

$flow_{in}\mathcal{R}(F, Mt, E_{from}, sv(S_{new}, U), U/U_t)$ ($\mathcal{R}$-rules 8.6, 8.7)

Alg. 8.5 $match\_out\_flow(\Delta, F_{ref}, Mt, S_{ref}\text{-}S_{new}, flow(F, sv(S_{new}, U), E_{to}, U/U_t))$ — inputs: Δ, F_{ref}; inputs/outputs: Mt, S_{ref}, S_{new}; outputs: F, E_{to}, U, U_t. F_{ref} is an out-flow from the state variable S_{ref} (flow form (f2) of Fig. 8.3) **IF**

$flow_{out}\mathcal{R}(F, Mt, sv(S_{new}, U), E_{to}, U/U_t)$ ($\mathcal{R}$-rules 8.8, 8.9)

Alg. 8.6 $match\_between_{in}\_flow\begin{pmatrix} \Delta, F_{ref}, Mt, S_{ref}\text{-}S_{new}, S1_{ref}\text{-}S1_{new}, \\ flow(F, sv(S1_{new}, U), sv(S_{new}, U), U/U_t) \end{pmatrix}$ — inputs: Δ, F_{ref}; inputs/outputs: Mt, S_{ref}, S_{new}; outputs: $S1_{ref}$, $S1_{new}$, F, U, U_t. F_{ref} is an in-between-flow into the state variable S_{ref} (flow form (f3) of Fig. 8.3) **IF**

$flow_{bt}\mathcal{R}(F, Mt, sv(S1_{new}, U), sv(S_{new}, U), U/U_t)$ ($\mathcal{R}$-rules 8.10, 8.11)

Alg. 8.7 $match\_between_{out}\_flow\begin{pmatrix} \Delta, F_{ref}, Mt, S_{ref}\text{-}S_{new}, S2_{ref}\text{-}S2_{new}, \\ flow(F, sv(S_{new}, U), sv(S2_{new}, U), U/U_t) \end{pmatrix}$ — inputs: Δ, F_{ref}, Mt, S_{ref}, S_{new}; outputs: $S2_{ref}$, $S2_{new}$, F, U, U_t. F_{ref} is an in-between-flow out of the state variable S_{ref} (flow form (f3) of Fig. 8.3) **IF**

$flow_{bt}\mathcal{R}(F, Mt, sv(S_{new}, U), sv(S2_{new}, U), U/U_t))$ ($\mathcal{R}$-rules 8.10, 8.11)

The *Match Form-Specific Link* Algorithms

Algs. 8.8 to 8.12 are the links' equivalent of the algorithms for matching form-specific flows. There is an algorithm to each link form, which succeeds conditioned to the satisfaction of the appropriate reuse-synthesis constraints, formulated as an $\mathcal{R}$-rule (see Algs. C.5 and C.7 for the algorithms' points of call). Alg. 8.8 is a little different from the others in that it establishes only the link itself — the new model elements (a state variable and a flow) at the two ends of the link are already established and given to the algorithm. Whereas in Algs. 8.10 to 8.12, just one element in one of the new link's ends is given. In establishing the new link these algorithms also establish the pair of matched elements (reference and new) at the other end of the link.

Alg. 8.8 $establish\_SVFlLk(\Delta, SV_{new}, Fid_{new}, link(SV_{new}, flow_{id}(Fid_{new})))$ — inputs: Δ, SV_{new}, Fid_{new}. Establish from Δ a link of form (l1) or (l2) of Fig. 8.4, between SV_{new} and the flow named Fid_{new}

IF

$link\mathcal{R}(S_{new}, flow_{id}(Fid_{new}))$ ($\mathcal{R}$-rule 8.12)

Alg. 8.9 $match\_toFlLk(\Delta, F_{new}, L_{ref}, link(CV_{new}, flow_{id}(Fid_{new})), CV_{ref}\text{-}CV_{new})$ — inputs: Δ, F_{new}, L_{ref}, Fid_{new}, CV_{ref}; output: CV_{new}. Match with Δ the reference link L_{ref} of form (l3) of Fig. 8.4, from the reference converter CV_{ref} to a flow, and establish in the new model a form-equivalent link from CV_{new} to the flow, which isnamed Fid_{new}

CASE CV_{ref} is an intermediate variable **THEN**

IF $link\mathcal{R}(iv(I_{new}, U), flow_{id}(Fid_{new}))$ ($\mathcal{R}$-rules 8.13, 8.14) **THEN**
$CV_{new} = iv(I_{new}, U)$

CV_{ref} is a parameter **THEN**

IF $link\mathcal{R}(param(P_{new}, U), flow_{id}(Fid_{new}))$ ($\mathcal{R}$-rules 8.15, 8.16) **THEN**
$CV_{new} = param(P_{new}, U)$

CV_{ref} is a driving variable **THEN**

IF $link\mathcal{R}(dv(D_{new}, U), flow_{id}(Fid_{new}))$ ($\mathcal{R}$-rules 8.17, 8.18) **THEN**
$CV_{new} = dv(D_{new}, U)$

Alg. 8.10 $match\_frSVLk(\Delta, SV_{new}, L_{ref}, link(SV_{new}, IV_{new}), IV_{ref}\text{-}IV_{new})$ — inputs: Δ, SV_{new}, L_{ref}, IV_{ref}; output: IV_{new}. Match with Δ the reference link L_{ref} of form (l4) of Fig. 8.4, from a state variable to the reference intermediate variable IV_{ref}, and establish in the new model a form-equivalent link from SV_{new} to IV_{new} **IF**

$link\mathcal{R}(SV_{new}, IV_{new})$ ($\mathcal{R}$-rule 8.19)

Alg. 8.11 $match\_frCVLk(\Delta, CV_{new}, L_{ref}, link(CV_{new}, IV_{new}), IV_{ref}\text{-}IV_{new})$ — inputs: Δ, CV_{new}, L_{ref}, IV_{ref}; output: IV_{new}. Match with Δ the reference link L_{ref} of form (l5) of Fig. 8.4, from a converter to the reference intermediate variable IV_{ref}, and establish in the new model a form-equivalent link from CV_{new} to IV_{new} **IF**

$link\mathcal{R}(CV_{new}, IV_{new})$ ($\mathcal{R}$-rule 8.20)

Alg. 8.12 $match\_toIVLk(\Delta, IV_{new}, L_{ref}, link(CV_{new}, IV_{new}), CV_{ref}\text{-}CV_{new})$ — inputs: Δ, IV_{new}, L_{ref}, CV_{ref}; output: CV_{new}. Match with Δ the reference link L_{ref} of form (l5) of Fig. 8.4, from the reference converter CV_{ref} to an intermediate variable, and establish in the new model a form-equivalent link from CV_{new} to IV_{new}

CASE CV_{ref} is an intermediate variable **THEN**

IF $link\mathcal{R}(iv(I_{new}, U), IV_{new})$ ($\mathcal{R}$-rule 8.21) **THEN** $CV_{new} = iv(I_{new}, U)$

CV_{ref} is a parameter **THEN**

IF $link\mathcal{R}(param(P_{new}, U), IV_{new})$ ($\mathcal{R}$-rule 8.22) **THEN** $CV_{new} = param(P_{new}, U)$

CV_{ref} is a driving variable **THEN**

IF $link\mathcal{R}(dv(D_{new}, U), IV_{new})$ ($\mathcal{R}$-rule 8.23) **THEN** $CV_{new} = dv(D_{new}, U)$

8.4 Multiple Model Solutions

Analogously to Synthesis-0, a Synthesis-$\mathcal{R}$ model solution corresponds to one output of the system's top-level algorithm, i.e., the value in $Model'$ after one execution of $synthesise\mathcal{R}(\Delta, M_{ref}, Model')$ (Alg. 8.1). For a given metadata set, Δ, and reference model, M_{ref}, Synthesis-$\mathcal{R}$ is able to synthesise multiple model solutions.

Once again, the algorithms are designed as though to give a single solution, and once again, we take advantage of Prolog facilities — backtracking and an all-solutions predicate (Sterling and Shapiro, 1994) — to enable the implemented Synthesis-$\mathcal{R}$ system to find multiple solutions.

In Synthesis-0 no selection of local or global solutions takes place. In Synthesis-$\mathcal{R}$, however, having a reference model compels making some use of it as benchmark. A model solution (new model) which is structurally identical to the reference model is as good as a solution can get. Hence, the best solutions, local or global, are those nearest in size to the corresponding part of or to the whole of the reference model, respectively. Size, expressed as the number of elements in the sets that represent solutions, is thus our criterion for solution selection. However simple, size provides a good measure of the extent (which we want to maximise) to which both reference model and metadata set are exploited.

One could firstly think of finding *all* possible global solutions and then select those that best approximate the reference model. There can be a very large number of global solutions, however, which makes the cost of such approach prohibitive.

Instead, in a hill-climbing heuristic fashion, one can select local solutions which are likely to make up the best global ones, and prune the search space by only further expanding the selected local solutions. Solutions are *local* with respect to each reference element paired up with a new model element, since the reference model traversal and new model expansion stem from each of such pairs.

It is again infeasible to do an exhaustive selection of local solutions, i.e., to find to each pair of elements all local solutions and then select the best ones. The problem is with link arcs. A single model element can have many links connected to it. As we saw in Section 6.3.3.2, links represent information transfers. Modellers are unrestrained in modelling information transfer. They will include in a model as many links as they believe necessary to express their understanding of the system of interest. If the reference model contains one or more heavily linked elements and the metadata set supports successful matches of most or all of such links, we could have a combinatorial explosion of local solutions.

In contrast, it is unusual to see models where the state variables have many flows connected to them. As opposed to information, flows represent physical processes of transfer of material (Section 6.3.3.1). Modellers tend to focus on a small number of such processes, since models with state variables overloaded with flows usually produce simulation results that are difficult to interpret. Moreover, each state variable can have up to four types of flows connected to it, namely, in-flows, out-flows, inward and outward in-between-flows.

Having said that, it is unlikely for a combinatorial explosion to happen in finding all solutions of matching flows per type of flow per state variable. These are, therefore, the local solutions upon which selection is done in Synthesis-$\mathcal{R}$.

This means that just the backbone of the reference model is used as benchmark. In other words, it would be too expensive to select the best local matches for arcs (links and flows) connected to every node (element) in the reference model, in an attempt to direct the search toward the best global solutions. It is feasible, however, to parsimoniously select the best local matches of flow arcs (which have state variables as nodes) only, in an attempt to direct the search toward global solutions where at least the matching of the core of the reference model, its backbone, is maximised. This does not necessarily reduce the number of possible global solutions significantly, however, as there can be many possibilities of matches for the network of links.

The points of selection of local solutions are in Alg. C.2. As specified in the algorithm, four sets of reference flows are found, one to each type of flow in relation to the current reference state variable. For each of the four sets, are found all the matching sets of new (not yet included in the new model) flows and their state variable(s). Then the largest of these matching sets are selected (the search for all such local solutions and their selection is not shown in the algorithm). As an example, say a reference state variable R has two in-flows. These can be successfully matched with a new state variable A with *two* new in-flows, as well as with a new state variable B with *one* new in-flow. The former match would be selected and the new components added to the new model.

It should be understood that the simple selection (or search) method above is heuristic, so it is not guaranteed that the optimal backbones, not to mention the optimal complete models, are synthesised. A selected largest local solution might not lead to the largest possible backbone (or model), when a smaller local solution thrown away would. This makes Synthesis-$\mathcal{R}$, unlike Synthesis-0, non-exhaustive. In both systems the search is guaranteed to terminate.

Finally, the reuse of a reference model simplifies how each synthesised model solution is to be interpreted with respect to the others (several possibilities apply in Synthesis-0 — see Section 7.3). Each model solution represents a successful different way of matching the reference model against the given metadata set, and should be seen as an alternative, independent solution.

8.5 Inference Pattern

We know from the previous chapter that inferences in Synthesis-0 are mostly deductive, except for the abductive associations between metadata and model structure (see Section 7.4.2).

In Synthesis-$\mathcal{R}$, in turn, all inferences are performed deductively — the associations between metadata and model structure are no longer abductive — employing the standard logic programming framework and the Prolog language as before.

The abductive vs deductive associations, performed over one shared library of Metadata $\leftrightarrow$ Model association rules (Table 6.1), constitute a key distinction between the two systems.

In Synthesis-0, **associations occur from given metadata to model structure**. Put briefly, the given metadata terms unify with the consequents of the association rules, so the model component terms are conjectured through abduction to allow for the entailment of the metadata terms.

In Synthesis-$\mathcal{R}$, **associations occur from given model structure to (given) metadata**. The synthesis is driven by the reference model structure, rather than by metadata, reversing the orientation of the associations, which become deductive. The given model component terms (determined by the reference model structure) unify with the antecedents of the association rules.

Therefore, the non-abductive $association(Q_{term}, ModelFrag)$ constraint, as opposed to Expression 7.1, is defined as follows, where Q_{term} is a quantity term and $ModelFrag$ is a model fragment containing one or more model components:

$$\begin{aligned} & association(Q_{term}, ModelFrag) \leftarrow \\ & \quad assoc\_rule((Q_{term} \leftarrow ModelFrag)) \end{aligned} \tag{8.1}$$

We can see in the association rules library that the same model fragment may imply multiple quantity terms. That would pose a problem were deduction applied exhaustively to the rules, requiring ill-specified metadata sets with single quantities described as belonging to orthogonal physical dimensions; but deduction is not applied exhaustively.

The $association(Q_{term}, ModelFrag)$ constraint is applied in the $\mathcal{R}$-rules, the constraints rules for reuse-synthesis of model components (Section 8.6). The rules existentially quantify quantity terms and define that a deduced quantity term is (tentatively) unified with a quantity description in the metadata set. Upon unification, the described quantity is assigned to the model fragment, which is then established in the current model solution, provided all other relevant constraints, if any, are satisfied. If, in this way, multiple described quantities can be assigned to a model fragment, the synthesis algorithms take care of placing each distinctly assigned model fragment in a separate model solution.

Ontological Checking

Just as in Synthesis-0, in Synthesis-$\mathcal{R}$ all retrieved metadata (e.g., following an association as above) are checked for Ecolingua-compliance. The same model of execution and mechanism for ontological checking presented in Section 7.4.1 apply.

8.6 Reuse-Synthesis Constraints of Model Components

The algorithms that match form-specific flows and links (Algs. 8.4–8.7 and 8.8–8.12) use implication rules to match reference model components with the metadata set, which, if proved successfully, establishes components of the same form in the new model.

Such rules, we call $\mathcal{R}$-rules, are the reuse-synthesis counterparts of the heuristic rules in Section 6.3, which define the synthesis constraints of model components in the metadata-only approach.

The metadata-only rules are formulated as general metadata-based definitions of model components, acting in synthesis as a representation of constrained modelling knowledge. Because they rely solely on metadata evidence, which is liable to multiple interpretations, tight constraints are necessary to control the synthesis process, to make sure that each synthesised component does not bring inconsistencies to the model.

While the $\mathcal{R}$-rules encode essentially the same domain-specific heuristic knowledge, they are not as general as the metadata-only rules but tailored for the reuse algorithms. The structural knowledge encapsulated in the reference model and its orderly core-outwards traversal in parallel with the synthesis of the new model, allows synthesis constraints to be softened. While in the metadata-only approach the synthesis rules are formulated to maximise the use of metadata, since this is all there is, the $\mathcal{R}$-rules, in turn, are formulated to maximise the use of the reference model structure — the only model fragments rejected are those to which contradictory metadata evidence exists. Ultimately, softer constraints lead to a more efficient synthesis, as we show in Chapter 9.

Recall from Chapter 6 that synthesis constraints are classified into metadata and integrity constraints. These are softened in the reuse-synthesis approach, in the following general ways:

Metadata constraints are lessened in quality and quantity.

> In the metadata-only approach model elements are assigned the *strong* or *weak* evidence class depending on availability in the metadata set of what we generally call *additional* evidence, i.e., descriptions of certain properties applied to the quantities, such as the constancy property (Section 6.1.4). In the reuse approach evidence classes are not assigned. Instead, all that is required is that the metadata

set does not hold evidence which, according to the Ecolingua ontology, would contradict relevant data properties for establishing model component types. We call these constraints *least constraints*.

To establish an arc, that is, a flow or link model connection, besides the metadata constraints of the arc itself, the constraints of at most only one of the nodes are verified in practice (with one exception). This is possible for two reasons:

1. Each connection in the new model is a structural copy of a connection in the reference model. In other words, in synthesising a new connection, its form is pre-determined by the form of the reference connection at hand.

2. The traversal of the reference model and synthesis of the new one are done in a pre-defined orientation — flows and state variables first, then the immediate fringe of links, followed by the far fringe of links. The $\mathcal{R}$-rules are designed in tune with this, defining flow and link arcs connecting a node (model element) at the borderline of the model synthesised thus far to a new node. The node at the borderline is already established and so do not require its constraints to be verified again. Only the constraints of the new node require verification, by way of which a new model element is established. In Synthesis-0, models are also expanded from established nodes, but which nodes these will be cannot be pre-determined. They could though be identified on the fly, and flows and links synthesis rules comprising only the constraints of the not yet established node could be applied. This is not done in order to preserve the generality of the metadata-only synthesis rules.[1]

The exception hinted above refers to when it is necessary to establish an in-between-flow as the new model's initial arc (Alg. 8.2), requiring the verification of the metadata constraints of both its state variables.

[1]But, practically, constraints of established elements are not re-computed by Synthesis-0. The caching technique is used to avoid this (see Section 7.6).

Integrity constraints defining preferences between components are waived.

This goes together with the waiving of evidence classes, on which the definitions of this kind of integrity constraints in Synthesis-0 are mostly based.

Model elements are not preferred over others in order to maximise the use of the reference model structure. Any element that has metadata to match a reference element, satisfying the softened reuse constraints is taken in.

Nonetheless, the global kind of integrity constraint, i.e., that each quantity can only appear once in each alternative model solution, remains applicable. But with reuse it is much simpler to respect this constraint, for the new model structure inherits the integrity of the reference model structure. This is handled by the reuse-synthesis algorithms, by simply making sure that each quantity is introduced only once into the model.

We now show the $\mathcal{R}$-rules for each type of model component, in relation to their metadata-only rules counterparts in Section 6.3, adding details on the constraints softening, where needed.

For ease of reference we have kept the notation of the $\mathcal{R}$-rules as similar as possible to that of the metadata-only rules, at the expense of having a slightly larger than necessary set of $\mathcal{R}$-rules.

8.6.1 Constraints of Model Elements

$\mathcal{R}$-rule of State Variables – Counterpart of Rules 6.1 and 6.3

S is a state variable, representing material Mt in entity E, with unit of measure U, if a $sv(S,Mt,E,U)$ term can be associated with some quantity term Q_{term}, and Q_{term} is described:

$$\begin{aligned} sv\mathcal{R}(S,Mt,E,U) \leftarrow \\ \exists Q_{term} \,.\, association(Q_{term}, sv(S,Mt,E,U)) \;\wedge \\ (data(Q_{term}) \;\vee\; model\_goal\_var(Q_{term})) \end{aligned} \tag{8.2}$$

In the rule's reading, as well as in the reading of the rules that follow, we say that the model component term is associated with the quantity term, rather than the other way round, to stress the deductive associations from model structure to metadata in reuse-synthesis (Section 8.5).

Also note in the $\mathcal{R}$-rules that the *association* relation goal now precedes the descriptional goals. Although declaratively it makes no difference, this change in the goals order, motivated by the deductive associations, brings about an efficiency gain within Prolog's computational model of execution in which the rules are used. Solving the *association* relation first (through the corresponding Prolog predicate) instantiates the quantity term, either fully or partially depending on the algorithm being executed, which renders a more informed, efficient subsequent search for the term in the metadata set.

What is more, since evidence classes are not assigned to model elements, the *model material* constraint (which determines *strong* and *weak* state variables in the metadata-only approach) as well as the integrity constraint are waived.

$\mathcal{R}$-rule of Intermediate Variables – Counterpart of Rules 6.4 and 6.6

I is an intermediate variable with unit of measure U, if an $iv(I,U)$ term can be associated with some quantity term Q_{term}, if Q_{term} is described, and if I is not described as a constant:

$$
\begin{aligned}
& iv\mathcal{R}(I,U) \leftarrow \\
& \quad \exists Q_{term} \,.\, association(Q_{term}, iv(I,U)) \;\wedge \\
& \qquad\qquad (data(Q_{term}) \;\vee\; model\_goal\_var(Q_{term})) \;\wedge \\
& \quad \neg\, data(constant(I))
\end{aligned}
\tag{8.3}
$$

The same softening of constraints for intermediate variables, as well as for parameters and driving variables in the sequel, is the same as explained for state variables above.

Moreover, there is a change in the constraint over metadata evidence used in addition to the quantity description (see Section 6.3.1.2). It becomes that the quantity to be modelled as an intermediate variable must not be described as constant. This is the *least* constraint, in that, according to Ecolingua Influence relation (Section 3.2.1.7), the constancy property is the only evidence that would imply that the quantity could not be influenced, which is the fundamental characteristic of intermediate variables.

Another change is in the intermediate variable relation in the head of the rule: the model material, *Mt*, is no longer an argument of the relation. Such argument is relevant in the metadata-only approach for verification of integrity constraints. Since integrity constraints are not explicitly defined in reuse-synthesis, the model material argument can be dispensed with. The same applies to $\mathcal{R}$-rules of parameters, driving variables and links.

$\mathcal{R}$-rule of Parameters – Counterpart of Rules 6.8 and 6.9

P is a parameter with unit of measure U, if a $param(P,U)$ term can be associated with some quantity term Q_{term}, if Q_{term} is described, and if P is not described as influenced:

$$\begin{aligned} & param\mathcal{R}(P,U) \leftarrow \\ & \quad \exists Q_{term} \,.\, association(Q_{term}, param(P,U)) \;\wedge \\ & \quad\quad data(Q_{term}) \;\wedge \\ & \quad \neg\, data(influences(Q,P,Sign)) \end{aligned} \tag{8.4}$$

For parameters, reversely to the intermediate variables $\mathcal{R}$-rule, the least constraint, besides the quantity description, is that the parameter quantity must not to be described as influenced, since this would prevent it from holding the constancy property, the parameters' fundamental property.

$\mathcal{R}$-rule of Driving Variables – Counterpart of Rule 6.11

D is a driving variable with unit of measure U, if a $dv(D,U)$ term can be associated with some quantity term Q_{term}, if Q_{term} is described, and if D is not described as constant nor as influenced:

$$
\begin{aligned}
& dv\mathcal{R}(D,U) \leftarrow \\
& \quad \exists Q_{term} \,.\, association(Q_{term}, dv(D,U)) \;\wedge \\
& \quad\quad data(Q_{term}) \;\wedge \\
& \quad \neg\, data(constant(D)) \;\vee\; (data(influences(Q,D,Sign)))
\end{aligned} \tag{8.5}
$$

For driving variables, the least constraint, in addition to the quantity description, is as in the metadata-only approach, i.e., the lack of evidence of both constancy and ‘influenced’ properties for the described quantity — any of these properties would prevent the quantity from being modelled as a driving variable (see Section 6.3.1.4).

8.6.2 Constraints of Model Connections

$\mathcal{R}$-rules of In-Flows

– Counterpart of Rule 6.13

F is a flow of material Mt from an entity E_{from} into a state variable S with unit of measure U, and is measured in the unit of measure U/U_t, if U_t is the model time unit, if S can be established as a state variable with unit of measure U representing material Mt in entity E_{to}, if a term $mattrans(F,Mt,E_{from},E_{to},U/U_t)$ can be associated with some quantity term Q_{term} and Q_{term} is described, and if a state variable with unit of measure U cannot be established representing material Mt in entity E_{from}:

$$
\begin{aligned}
& flow_{in}\mathcal{R}(F,Mt,E_{from},sv(S,U),U/U_t) \leftarrow \\
& \quad model\_time\_unit(U_t) \;\wedge \\
& \quad sv\mathcal{R}(S,Mt,E_{to},U) \;\wedge \\
& \quad \exists Q_{term} \,.\, association(Q_{term}, mattrans(F,Mt,E_{from},E_{to},U/U_t)) \;\wedge \\
& \quad\quad data(Q_{term}) \;\wedge \\
& \quad \neg\, sv\mathcal{R}(S',Mt,E_{from},U)
\end{aligned} \tag{8.6}
$$

$\mathcal{R}$-rule 8.6 is used to establish a flow in the new model that matches an in-flow — from a source into a state variable — in the reference model.

In the implemented system the state variable constraints represented in the rule by $sv\mathcal{R}(S, Mt, E_{to}, U)$ have different procedural meanings depending on whether or not the state variable S at the flow's end has already been established (as determined by the usage of Alg. 8.4). If so (S is ground), the constraints do not need to be recomputed — the caching technique through meta-interpretation caters for this (Section 7.6). The constraints' purpose in the rule is simply to bind the state variable S, through the E_{to} entity, to the *material transfer* term, ensuring in this way that the quantity term to be retrieved will be appropriate to give support to an in-flow into such state variable. If, otherwise, the state variable S is not yet established, then the constraints, besides the purpose above, establish it, provided that the subsequent constraints are also verified.

– Counterpart of Rule 6.14

> *F is a flow of material Mt from the outside of the model's scope into a state variable S with unit of measure U, and is measured in the unit of measure U/U_t, if U_t is the model time unit, if a state variable S with unit of measure U can be established representing material Mt in some entity, and if data exists of a positive influence of a quantity F into the quantity S:*

$$
\begin{aligned}
&\mathit{flow}_{in}\mathcal{R}(F, Mt, \mathit{outside}, sv(S,U), U/U_t) \leftarrow \\
&\quad \mathit{model\_time\_unit}(U_t) \ \wedge \\
&\quad sv\mathcal{R}(S, Mt, E, U) \ \wedge \\
&\quad \mathit{data}(\mathit{influences}(F, S, +))
\end{aligned}
\tag{8.7}
$$

Here again, the $sv\mathcal{R}(S, Mt, E, U)$ constraints may be computed or the cached solution retrieved in the implemented system depending on the status of S.

We saw in Section 6.3.3.1 that the use of influence data to synthesise in-flows and out-flows calls for elaborate integrity constraints, for such descriptions make weaker metadata evidence compared to quantity descriptions. Also, as mentioned earlier, in the reuse approach integrity constraints are waived to maximise the use of the reference model structure. For $\mathcal{R}$-rule 8.7, in particular, this means that other more informed synthesisable flows are not given preference over the in-flow F from *outside* if it happens to be synthesised first (see Rule 6.15) —

Algs. 8.2 and C.2 guarantee that a unique flow named F occurs in the model. The algorithms do not avoid, however, a flow F and a state variable F occurring in the same model, in which case the flow should be renamed by the model user.

$\mathcal{R}$-rules of Out-Flows

These are equivalent to the in-flows $\mathcal{R}$-rules for flows from state variables into sinks.

– Counterpart of Rule 6.16

F is a flow of material Mt from a state variable S with unit of measure U into an entity E_{to}, and is measured in the unit of measure U/U_t, if U_t is the model time unit, if S can be established as a state variable with unit of measure U representing material Mt in entity E_{from}, if a term $mattrans(F,Mt,E_{from},E_{to},U/U_t)$ can be associated with some quantity term Q_{term} and Q_{term} is described, and if a state variable with unit of measure U cannot be established representing material Mt in entity E_{to}:

$$
\begin{aligned}
& flow_{out}\mathcal{R}(F,Mt,sv(S,U),E_{to},U/U_t) \leftarrow \\
& \quad model\_time\_unit(U_t) \ \wedge \\
& \quad sv\mathcal{R}(S,Mt,E_{from},U) \ \wedge \\
& \quad \exists Q_{term} \ . \ association(Q_{term},mattrans(F,Mt,E_{from},E_{to},U/U_t)) \ \wedge \\
& \qquad\qquad data(Q_{term}) \ \wedge \\
& \quad \neg \ sv\mathcal{R}(S,Mt,E_{to},U)
\end{aligned}
\tag{8.8}
$$

– Counterpart of Rule 6.17

F is a flow of material Mt from a state variable S with unit of measure U into the outside of the model's scope, and is measured in the unit of measure U/U_t, if U_t is the model time unit, if a state variable S with unit of measure U can be established representing material Mt in some entity, and if data exists of a negative influence of a quantity F into the quantity S:

$$
\begin{aligned}
& flow_{out}\mathcal{R}(F,Mt,sv(S,U),outside,U/U_t) \leftarrow \\
& \quad model\_time\_unit(U_t) \ \wedge \\
& \quad sv\mathcal{R}(S,Mt,E,U) \ \wedge \\
& \quad data(influences(F,S,-))
\end{aligned}
\tag{8.9}
$$

$\mathcal{R}$-rules of In-between-Flows

As we know, these are flows between two state variables. Their $\mathcal{R}$-rules verify the constraints of the flow itself and of the two state variables. Although, similarly to the in-flows and out-flows $\mathcal{R}$-rules, in all but one situation one of the state variables will already have been established, and its constraints will not be verified again in practice. The exceptional situation is when the very first matching flow for the new model is an in-between-flow (Alg. 8.2), in which case $\mathcal{R}$-rule 8.10 or 8.11 establishes a new flow with two new state variables.

– Counterpart of Rule 6.19

> *F is a flow of material Mt with unit of measure U/U_t from a state variable S_1 with unit U into a state variable S_2 with unit of measure U, if U_t is the model time unit, if state variables S_1 and S_2 with unit of measure U can both be established representing material Mt in entities E_{from} and E_{to} respectively, and if a term $mattrans(F, Mt, E_{from}, E_{to}, U/U_t)$ can be associated with some quantity term Q_{term} and Q_{term} is described:*

$$\begin{aligned}
&flow_{bt}\mathcal{R}(F, Mt, sv(S_1, U), sv(S_2, U), U/U_t) \leftarrow \\
&\quad model_time_unit(U_t) \ \wedge \\
&\quad sv\mathcal{R}(S_1, Mt, E_{from}, U) \ \wedge \ sv\mathcal{R}(S_2, Mt, E_{to}, U) \ \wedge \\
&\quad \exists Q_{term} \ . \ association(Q_{term}, mattrans(F, Mt, E_{from}, E_{to}, U/U_t)) \ \wedge \\
&\qquad\qquad data(Q_{term})
\end{aligned} \tag{8.10}$$

– Counterpart of Rule 6.20

> *F is a flow of material Mt with unit of measure U/U_t from a state variable S_1 with unit U into a state variable S_2 with unit of measure U, if U_t is the model time unit, if state variables S_1 and S_2 with unit of measure U can both be established representing material Mt in some entities, and if data exists of a negative influence of a quantity F into the quantity S_1, as well as of a positive influence of F into the quantity S_2:*

$$\begin{aligned}
&flow_{bt}\mathcal{R}(F, Mt, sv(S_1, U), sv(S_2, U), U/U_t) \leftarrow \\
&\quad model_time_unit(U_t) \ \wedge \\
&\quad sv\mathcal{R}(S_1, Mt, E_1, U) \ \wedge \ sv\mathcal{R}(S_2, Mt, E_2, U) \ \wedge \\
&\quad data(influences(F, S_1, -)) \ \wedge \ data(influences(F, S_2, +))
\end{aligned} \tag{8.11}$$

$\mathcal{R}$-rules of Links into Flows

As determined by the orderly traverse-and-synthesise process where flows and state variables are synthesised first, in all the $\mathcal{R}$-rules below the link's terminal node is, being a flow, an established model element. Therefore, the only metadata constraints that require verification are the link's own constraints and the initial node's constraints, except for when the initial node is a state variable which will be an established element as well, not requiring constraints verification ($\mathcal{R}$-rule 8.12).

– Counterpart of Rule 6.22

> *A link is established from an established state variable element S with unit of measure U to its established flow element F, if data exists of an influence of the quantity S into the quantity F:*

$$
\begin{aligned}
& link\mathcal{R}(sv(S,U), flow_{id}(F)) \leftarrow \\
& \quad data(influences(S,F,Sign))
\end{aligned}
\tag{8.12}
$$

– Counterpart of Rule 6.24

> *A link is established from an intermediate variable element I with unit of measure U to an established flow element F, if data exists of an influence of a quantity I into a quantity F, and if an intermediate variable I with unit U can be established:*

$$
\begin{aligned}
& link\mathcal{R}(iv(I,U), flow_{id}(F)) \leftarrow \\
& \quad data(influences(I,F,Sign)) \ \wedge \\
& \quad iv\mathcal{R}(I,U)
\end{aligned}
\tag{8.13}
$$

– Counterpart of Rule 6.25

A link is established from an intermediate variable element I with unit of measure U to an established flow element F, if an iv(I,U) term and a link(I,F) term can be associated with some quantity term Q_{term} *and* Q_{term} *is described, and if I is not described as constant:*

$$\begin{aligned}
&link\mathcal{R}(iv(I,U), flow_{id}(F)) \leftarrow \\
&\quad \exists Q_{term} \,.\, association(Q_{term}, iv(I,U) \;\wedge\; link(I,F)) \;\wedge \\
&\qquad\quad (data(Q_{term}) \;\vee\; model_goal_var(Q_{term})) \;\wedge \\
&\quad \neg\, data(constant(I))
\end{aligned} \tag{8.14}$$

– Counterpart of Rule 6.26

A link is established from a parameter element P with unit of measure U to an established flow element F, if data exists of an influence of a quantity P into a quantity F, and if a parameter P with unit U can be established:

$$\begin{aligned}
&link\mathcal{R}(param(P,U), flow_{id}(F)) \leftarrow \\
&\quad data(influences(P,F,Sign)) \;\wedge \\
&\quad param\mathcal{R}(P,U)
\end{aligned} \tag{8.15}$$

– Counterpart of Rule 6.27

A link is established from a parameter element P with unit of measure U to an established flow element F, if a param(P,U) term and a link(P,F) term can be associated with a quantity term Q_{term} *and* Q_{term} *is described, and if P is not described as influenced:*

$$\begin{aligned}
&link\mathcal{R}(param(P,U), flow_{id}(F)) \leftarrow \\
&\quad \exists Q_{term} \,.\, association(Q_{term}, param(P,U) \;\wedge\; link(P,F)) \;\wedge \\
&\qquad\quad data(Q_{term}) \;\wedge \\
&\quad \neg\, data(influences(Q,P,Sign))
\end{aligned} \tag{8.16}$$

– Counterpart of Rule 6.28

A link is established from a driving variable element D with unit of measure U to an established flow element F, if data exists of an influence of a quantity D into a quantity F, and if a driving variable D with unit U can be established:

$$\begin{aligned}&link\mathcal{R}(dv(D,U),flow_{id}(F)) \leftarrow \\ &\quad data(influences(D,F,Sign)) \ \wedge \\ &\quad dv\mathcal{R}(D,U)\end{aligned} \tag{8.17}$$

– Counterpart of Rule 6.29

A link is established from a driving variable element D with unit of measure U to an established flow element F, if a dv(D,U) term and a link(D,F) term can be associated with some quantity term Q_{term} and Q_{term} is described, and if D is not described as influenced neither as constant:

$$\begin{aligned}&link\mathcal{R}(dv(D,U),flow_{id}(F)) \leftarrow \\ &\quad \exists Q_{term}\ .\ association(Q_{term},dv(D,U) \ \wedge \ link(D,F)) \ \wedge \\ &\qquad data(Q_{term}) \ \wedge \\ &\quad \neg\ (data(influences(Q,D,Sign)) \ \vee \ data(constant(D)))\end{aligned} \tag{8.18}$$

$\mathcal{R}$-rules of Links into Intermediate Variables

– Counterpart of Rule 6.30

A link is established from an established state variable element S with unit of measure U_s to an intermediate variable element I with unit U_i, if data exists of an influence of the quantity S into a quantity I, and if an intermediate variable I with unit U_i can be established:

$$\begin{aligned}&link\mathcal{R}(sv(S,U_s),iv(I,U_i)) \leftarrow \\ &\quad data(influences(S,I,Sign)) \ \wedge \\ &\quad iv\mathcal{R}(I,U_i)\end{aligned} \tag{8.19}$$

Links of this form are synthesised from established state variables, thus only the metadata constraints of the link itself and of the intermediate variable are verified.

– Counterpart of Rules 6.31, 6.32 and 6.33

$\mathcal{R}$-rule 8.20 defines links to intermediate variables where the initial node is an established converter.

A link is established from an established converter element CV to an intermediate variable element I with unit U, if data exists of an influence of the quantity Q, held by CV, into a quantity I, and if an intermediate variable I with unit U can be established:

$$\begin{aligned}
&link\mathcal{R}(CV, iv(I,U)) \leftarrow \\
&\quad el_qtty(CV,Q)\ \wedge \\
&\quad data(influences(Q,I,Sign))\ \wedge \\
&\quad iv\mathcal{R}(I,U)
\end{aligned} \tag{8.20}$$

And $\mathcal{R}$-rules 8.21, 8.22 and 8.23 define links where the established model element is the terminal intermediate variable.

– Counterpart of Rule 6.31

A link is established from an intermediate variable element I with unit U to an established intermediate variable element IV, if data exists of an influence of a quantity I into the quantity Q held by IV, and if an intermediate variable I with unit U can be established:

$$\begin{aligned}
&link\mathcal{R}(iv(I,U), IV) \leftarrow \\
&\quad el_qtty(IV,Q)\ \wedge \\
&\quad data(influences(I,Q,Sign))\ \wedge \\
&\quad iv\mathcal{R}(I,U)
\end{aligned} \tag{8.21}$$

– Counterpart of Rule 6.32

> *A link is established from a parameter element P with unit U to an established intermediate variable element IV, if data exists of an influence of a quantity P into the quantity Q held by IV, and if a parameter P with unit U can be established:*

$$\begin{aligned}
&link\mathcal{R}(param(P,U),IV) \leftarrow \\
&\quad el_qtty(IV,Q)\ \wedge \\
&\quad data(influences(P,Q,Sign))\ \wedge \\
&\quad param\mathcal{R}(P,U)
\end{aligned} \tag{8.22}$$

– Counterpart of Rule 6.33

> *A link is established from a driving variable element D with unit U to an established intermediate variable element IV, if data exists of an influence of a quantity D into the quantity Q held by IV, and if a driving variable D with unit U can be established:*

$$\begin{aligned}
&link\mathcal{R}(dv(D,U),IV) \leftarrow \\
&\quad el_qtty(IV,Q)\ \wedge \\
&\quad data(influences(D,Q,Sign))\ \wedge \\
&\quad dv\mathcal{R}(D,U)
\end{aligned} \tag{8.23}$$

8.7 Worked Example

Figure 8.6 shows part of the reuse-synthesis search tree generated by reusing the depicted reference model upon the pond management system metadata set in Appendix B.

We deliberately use here the same metadata set used to give Synthesis-0's worked example. One of the full model solutions synthesisable from this metadata set by Synthesis-0 is shown in Figure 2.2. The reader may find helpful to compare this full model solution with the solutions (partial and final) in the reuse-synthesis search tree in Figure 8.6. For an easy visual comparison corresponding model components have been placed, as far as possible, in the same position in both figures.

The reference model is based on a grass-deer ecosystem model found in (Haefner, 1996, Chapter 3), with added components to make the example more interesting. Since the reference model content is irrelevant — the reuse-synthesis exploits its structure only — for convenience and clarity we use in the example fictitious short names for the reference model components.

The search tree shown is partial. Only one branch is shown ending in one of the possible final model solutions. The tree nodes, or reuse-synthesis states, are numbered outlining the order in which states are reached (considering the states shown only, obviously): flows and state variables first, then the immediate fringe of links, and for last the far fringe of links.

All assumptions and simplifications made on presenting Synthesis-0's worked example, concerning metadata descriptions shown, Ecolingua checks and notation, also apply here (see Section 7.5). In particular, once again we show only the *primary* metadata description involved in synthesising each model component. They are the description that unify with the non-negated $data$ predicate in the body of the constraints $\mathcal{R}$-rules in Section 8.6.

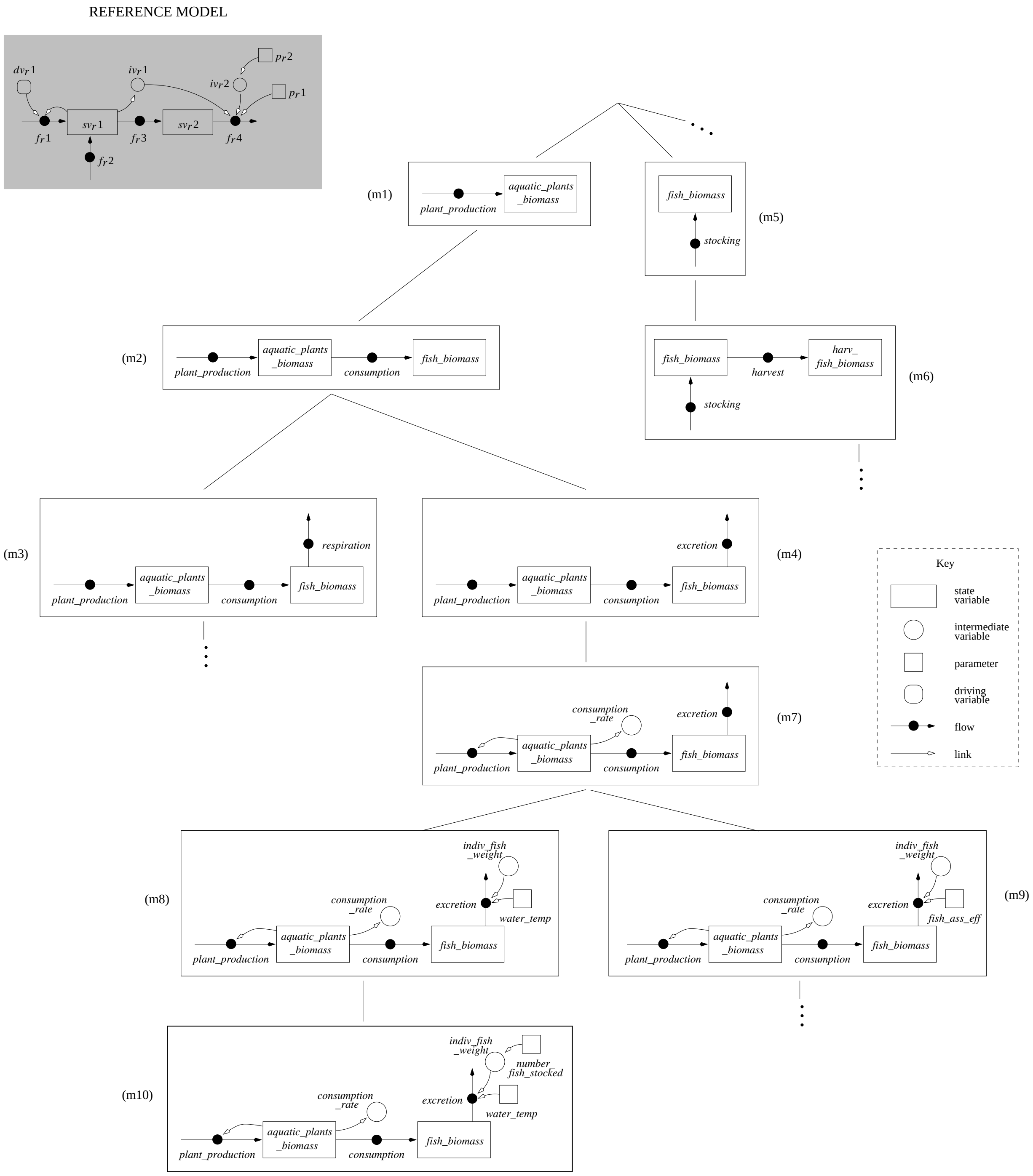


Figure 8.6: A portion of a Synthesis-$\mathcal{R}$ search tree.

8.7.1 Tracing the Reuse-Synthesis Search Tree

Alg. 8.1, the top-level algorithm, starts with the matching of flows, handing the execution over to Alg. 8.2.

The metadata set does have a specification of model material, namely *biomass*, which is retrieved from the metadata set and given to the initialisation procedure. That is, at this point it is determined that the model solutions about to be synthesised will represent flow of biomass.

Notice that in-flow is the most common type of flows connected to any one state variable in the reference model, namely, f_r1 and f_r2. One of these flows is then tried out first to initialise the matching (Alg. 8.4), followed, on backtracking, by the other one and then by other, less common flows. The search tree shows matching initialisation with the f_r1 flow and some of the synthesis states that follow.

State (m1) Alg. 8.4 matches:

- f_r1 with the description $data(abs\_rate(plant\_production, biomass, outside, aquatic\_plants, kg/ha/day))$ in the metadata set giving the new flow $flow(plant\_production)$; and
- sv_r1 with $data(mat\_dens(aquatic\_plants\_biomass, biomass, aquatic\_plants, kg/ha))$ giving $sv(aquatic\_plants\_biomass)$.

These are the initial matches. Recall that to match means to find evidence in the metadata set that supports a model component of the same form as the reference component at hand (an in-flow and its state variable, in this case), and that this is done by solving an appropriate reuse-synthesis constraints $\mathcal{R}$-rule. For the above matches, for example, such $\mathcal{R}$-rule is either 8.6 or 8.7.

Once the initialisation is done, the synthesis process moves on to the *traverse backbone and match* procedure (Alg. C.1).

State (m2) Alg. 8.7 matches:

- f_r3 with $data(abs\_rate(consumption, biomass, aquatic\_plants, fish, kg/ha/day))$ giving $flow(consumption)$; and
- sv_r2 with $data(mat\_dens(fish\_biomass, biomass, fish, kg/ha))$ giving $sv(fish\_biomass)$.

The initial pair of state variables $sv_r1 - sv(aquatic\_plants\_biomass)$ from state (m1) is the first to be surveyed (Alg. C.2). f_r3 matches metadata to establish the $consumption$ flow in the new model, as well as the pair $sv_r2 - sv(fish\_biomass)$, which is the next pair of state variables to be surveyed. f_r2 finds no match in the metadata set.

States (m3), (m4) Alg. 8.5 matches, respectively to the states:

- f_r4 with $data(abs\_rate(respiration, biomass, fish, outside, kg/ha/day))$ giving $flow(respiration)$; or
- f_r4 with $data(abs\_rate(excretion, biomass, fish, outside, kg/ha/day))$ giving $flow(excretion)$.

f_r4 can be matched with the metadata set to establish two different new flows, leading to two alternative states.

That ends the flows traversing and matching procedure for the initialisation as in state (m1).

State (m5) Alg. 8.4 matches:

- f_r1 with $data(abs\_rate(stocking, biomass, outside, fish, kg/ha/day))$ giving $flow(stocking)$; and
- sv_r1 with $data(mat\_dens(fish\_biomass, biomass, fish, kg/ha))$ giving $sv(fish\_biomass)$.

State (m5) shows another initialisation that ultimately leads to a different model solution to that in state (m10).

State (m6) Alg. 8.7 matches:

- f_r3 with $data(abs\_rate(harvest, biomass, fish, harv\_fish, kg/ha/day))$ giving $flow(harvest)$; and
- sv_r2 with $data(mat\_dens(harv\_fish\_biomass, biomass, harv\_fish, kg/ha))$ giving $sv(harv\_fish\_biomass)$.

State (m7)

- Alg. 8.8 matches $link(sv_r1, f_r1)$ with $data(influences(aquatic\_plants\_biomass, plant\_production, ?))$ giving $link(sv(aquatic\_plants\_biomass), flow(plant\_production))$; and
- Alg. 8.10 matches $link(sv_r1, iv_r1)$ with $data(influences(aquatic\_plants\_biomass, consumption\_rate, +))$ giving $link(sv(aquatic\_plants\_biomass), iv(consumption\_rate))$.

Once the matching of flows is completed, with flows and their state variables established in the new model, the matching of links can take place (Alg. 8.3), starting with the immediate fringe of each pair of matched flows in turn.

Three links are found to be connected to the reference flow f_r1 and its state variable sv_r1, out of which two are successfully matched with the metadata set.

States (m8), (m9) Alg. 8.9 matches, respectively to the states:

- $link(iv_r2, f_r4)$ with $data(influences(indiv\_fish\_weight, excretion, ?))$ giving $link(iv(indiv\_fish\_weight), flow(excretion))$,
  $link(p_r1, f_r4)$ with $data(influences(water\_temp, excretion, ?))$ giving $link(param(water\_temp), flow(excretion))$; or
- $link(iv_r2, f_r4)$ with $data(influences(indiv\_fish\_weight, excretion, ?))$ giving $link(iv(indiv\_fish\_weight), flow(excretion))$,
  $link(p_r1, f_r4)$ with $data(influences(fish\_ass\_eff, excretion, ?))$ giving $link(param(fish\_ass\_eff), flow(excretion))$.

The last pair of flows from which successful link matches stem is f_r4-$flow(excretion)$ (f_r3 has no links connected to it). Again, two out of the three links connected to f_r4 are successfully matched. The parameter in $link(p_r1, f_r4)$ can be matched with the metadata set to give $param(water\_temp)$ (state (m8)) or $param(fish\_ass\_eff)$ (state (m9)).

Recall that in Synthesis-0's search tree example (Figure 7.4) the metadata concerning the $water\_temp$ (water temperature) quantity supported it being modelled as either a driving variable, an intermediate variable or a parameter, leading to three alternative partial models. In reuse-synthesis, alternative model representations for a quantity do not occur in this way. Rather, they occur if the quantity's metadata happen to match distinct reference components is distinct model solutions.

What determines that the quantity $water\_temp$ is modelled as a parameter in state (m8) is that a reference parameter component matched its metadata. Although this does not occur in this particular reference model, $water\_temp$ could be modelled as, say, a driving variable in a different model solution if a reference driving variable happened to match its metadata.

At this point, all established flow pairs have been dealt with and the new model's immediate fringe is established. Just the far fringe of links is now left to be synthesised.

State (m10) Alg. 8.12 matches:

- $link(p_r2, iv_r2)$ with $data(influences(number\_fish\_stocked, indiv\_fish\_weight, ?))$ giving $link(param(number\_fish\_stocked), iv(indiv\_fish\_weight))$.

A single link lies in the reference model's far fringe, which matches the metadata set successfully.

One finalised model solution is reached.

The reuse-synthesis approach best case is to synthesise a new model which is a structural copy of the reference model, meaning that all reference components have been successfully matched with the new model's metadata set.

More or less of a reference model's structure will be fit to represent the new model depending on how compatible it is with the new model's metadata set, that is, to which extent it captures the ontological properties of the new model's data. The ideal reference model would have a data set whose properties would match the new model's data set perfectly.[2]

If it so happens that the metadata set has more potential for synthesis than the reference model can fulfill then such potential is undermined. The example presented illustrates this kind of scenario. Compare the model solution reached (in state (m10)) with the model solution in Figure 2.2, synthesised from the metadata set alone. The converse scenario is to have the reference model under-exploited, for being richer than the new model's metadata set.

Such trade-offs are explored in the next chapter where Synthesis-0 and Synthesis-$\mathcal{R}$ are comparatively evaluated.

[2]This is not to say that the data itself should be equivalent in the two data sets. For example, a data set about an animal population and another data set about, say, a crop, may have a matching network of functional relationships between variables (a match of properties), while the variables themselves are distinct, representing distinct objects in the respective systems-of-interest.

Part IV

Evaluation and Conclusions

Chapter 9

Empirical Evaluation of Synthesis Run Times

Let us first have a quick look at Synthesis-0 again. In this system Ecolingua plays a role on metadata specification. This is useful for model synthesis for two interrelated reasons:

- The foundational, ontological concepts can be connected to model components. By precisely defining such connections we were able to automate synthesis of models from metadata, which preserves consistency between models and their data sets; and

- Being an ontology (i.e. a unifying, standardising set of concepts represented by terms) many and diverse data sets can be described through such terms to take advantage of automated synthesis. What is more, since the terms are defined formally, we can also automate the verification of validity of the descriptions.

These benefits stem from metadata which is obtained by directly using the ontology vocabulary to describe data; we can view metadata specification as an instance of ontology-based knowledge specification in general. This is fine, but we do not need to stop there.

One strong motivation for engineering knowledge founded on ontologies is reusability, on the grounds that in order to reuse knowledge we, or systems, need to know its meaning and ontologies make possible to elicit such meaning. So on top of benefitting from *knowledge specified through an ontology* (e.g., our metadata) we should be able to reuse *knowledge that can be derived from knowledge specified through an ontology* (e.g., models that can be synthesised from metadata). The question is:

> Once knowledge reuse supported by ontologies is achieved somehow, what practical gains can not "re-inventing the wheel" bring about to systems?

This is the question that motivates this evaluation experiment, and indeed, the Synthesis-$\mathcal{R}$ system to provide for the 'knowledge reuse supported by ontologies is achieved somehow' part of the question.

The goal of the experiment is to provide evidence from data towards answering the 'gains' part of the question. The overall strategy of the experiment is like this:

Let us call:

Ko the level of knowledge specified through an ontology — the level of metadata specified through Ecolingua

K the level of knowledge that can be derived from knowledge specified through an ontology — the level of models that can be synthesised, through Synthesis-0 or Synthesis-$\mathcal{R}$, from metadata

as illustrated in Figure 9.1.

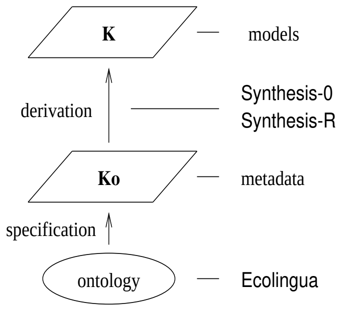


Figure 9.1: Ontology-founded knowledge levels.

Synthesis-0 is a system capable of deriving instances of **K** from instances of **Ko**, supported by Ecolingua; and

Synthesis-$\mathcal{R}$ is a system also capable of deriving instances of **K** from instances of **Ko** but *through reuse of other instances of* **K**, supported by Ecolingua.

Efficiency is the performance criterion on which the systems are compared. For comparison, the systems are run in a batch of trials and their run times are measured.

But, in order to explain efficiency gain, if any, and to know to where to allocate the credit, we need to identify in which features the systems differ and to consider their influence on efficiency. That is what we do in the next section.

After that we formulate our experimental hypothesis in Section 9.2 and describe the experimental procedure in Section 9.3. The experimental data and its collection follow in Section 9.4. And finally the evaluation results are presented, interpreted and generalised in Section 9.5.

9.1 Comparative Featural Characterisation of Synthesis-0 and Synthesis-$\mathcal{R}$

The reuse of a reference model by Synthesis-$\mathcal{R}$ allows various system features to be improved with respect to Synthesis-0. The *model building*, *model components' constraints* and *metadata retrieval* features are interdependently improved in their efficiency, as we shall discuss shortly. On the other hand, the *selection of local solutions* feature, which is exclusive to Synthesis-$\mathcal{R}$, causes an efficiency loss in trying (it is a heuristic) to improve the quality of model solutions.

The characteristics of such features in the two systems are summarised in Table 9.1 followed by a brief discussion. We give references to where previously in the thesis the features have been discussed in more detail.

<table>
<tr><th>Feature</th><th>Synthesis-0 – metadata resource</th><th>Synthesis-$\mathcal{R}$ – metadata + reference model resources</th><th>Efficiency</th></tr>
<tr><td>Model building</td><td>exploratory, exhaustive</td><td>bounded by reference model</td><td rowspan="3">increased in Synthesis-$\mathcal{R}$ compared to Synthesis-0</td></tr>
<tr><td>Model components' constraints</td><td>general, hard, abductive 'metadata-to-model fragment' associations</td><td>specific, soft, deductive 'model fragment-to-metadata' associations</td></tr>
<tr><td>Metadata retrieval</td><td>variable to match metadata terms</td><td>ground or partially instantiated term to match metadata terms</td></tr>
<tr><td>Selection of local solutions</td><td>none</td><td>best local flow matches per type of flow per state variable</td><td>decreased in Synthesis-$\mathcal{R}$ compared to Synthesis-0</td></tr>
</table>

Table 9.1: Featural characterisation of Synthesis-0 and Synthesis-$\mathcal{R}$.

Features of increased efficiency in Synthesis-$\mathcal{R}$ compared to Synthesis-0

Model building is the work of the algorithms described in Sections 7.2 and 8.3. Synthesis-0 builds models by expanding node (model element) by node in an exploratory, exhaustive fashion. For each node an exhaustive set of arcs (model connections) that can be connected to the node and do not violate integrity constraints are searched for and added to the model. All nodes are visited and expanded in this way until no more new nodes can be established.

Synthesis-$\mathcal{R}$, on the other hand, builds models by attempting to copy the reference model's structure. A pre-determined traversal method (from the core of the model outwards) sets the order in which the reference model's nodes are visited and arcs connected to them identified. As opposed to Synthesis-0's exhaustive search for arcs, each arc of the reference model is matched with the metadata until success occurs (which includes satisfaction of integrity constraints) or no suitable metadata can be found. And, rather than exhaustively expanding the model, the synthesis ends with the completion of the traversal of the reference model.

Model components' constraints in Synthesis-0 are general, heuristic metadata-based definitions of model components formulated as if-then rules (Section 6.3). Such generality and the use of abduction to associate metadata with model fragments (Section 7.4.2) require integrity constraints as part of the rules to enforce model consistency (Section 6.2). Besides the cost of verifying integrity constraints, there is also the overhead of the abductive associations, which are established by a specially built mechanism (Section 7.4.2.1). Even though the mechanism is not particularly demanding it is run for every association that takes place. The deductive 'model fragment-to-metadata' associations in Synthesis-$\mathcal{R}$, in turn, requires no more than Prolog's built-in unification algorithm (Section 8.5).

Moreover, having a reference model to bound the synthesis makes it possible for model components' constraints in Synthesis-$\mathcal{R}$ to be softened, hence requiring less computation for solution (Section 8.6). Integrity constraints that define preferences between components can be waived — Synthesis-$\mathcal{R}$'s algorithms try to copy to the model under synthesis whatever component types are there in the reference model. Metadata constraints are reduced to the so called least constraints which just check for contradictions between Ecolingua and relevant metadata descriptions. Also, redundant verifications of metadata constraints are cut out. Loss of generality is the price payed for the softening of constraints, which become useful only if applied in concert with Synthesis-$\mathcal{R}$'s algorithms (the classic trade-off between generality and efficiency).

Metadata retrieval efficiency is affected by the different styles of constraint formulation in the two systems, taking into account their solution within Prolog's model of execution (Section 8.6.1). The solution of constraints is driven by metadata evidence in Synthesis-0. Foremost, some evidence in the form of a metadata term is retrieved from the metadata set and then associated with the model component in question. At the point of retrieval metadata evidence is just a variable with which any described metadata term can unify. The solution of Synthesis-$\mathcal{R}$'s constraints, in turn, is driven by model component terms, reproductions of components in the reference model structure which are, at this stage, fully or partially instantiated to content of the new model. These terms are associated with the metadata terms to be retrieved, which in this way have their functor and some or all arguments determined. That restricts their possibilities of matching with terms in the metadata set, making retrieval more efficient. Clearly, such efficiency gain will be more significant where large metadata sets, which are expected in the ecology domain, are used.

Feature of decreased efficiency in Synthesis-$\mathcal{R}$ compared to Synthesis-0

Selection of local solutions concerns selecting the best possible expansions, or sets of connected arcs, of model nodes. In Synthesis-0 no such selection occurs: in each run of the system (either initial or subsequent backtracking runs), the set of arcs of each node that is first found is taken. On the other hand, Synthesis-$\mathcal{R}$ selects, for each type of flow, the sets of new matching flows that best approximate (in number of elements) the corresponding set of flows connected to the reference state variable, where the selection is preceded by the computation of *all* matching sets (Section 8.4).

Apart from the above differences, all motivated by the reference model resource in Synthesis-$\mathcal{R}$, the two systems are otherwise the same. Namely, they have in common: data structures for representation of metadata and models, library of metadata$\leftrightarrow$model association rules, Ecolingua compliance checking mechanism, meta-interpreter, and programming language and interpreter.

9.2 Experimental Hypothesis and Measured Variable

Having characterised differences in Synthesis-0 and Synthesis-$\mathcal{R}$ systems' features and their contribution to relative increased or decreased efficiency, we can now formulate our experimental hypothesis.

Experimental Hypothesis: Synthesis-$\mathcal{R}$'s improved features through reuse of a reference model give, compared to Synthesis-0, a net increased efficiency reflected in shorter synthesis run times.

The evaluation criterion is thus efficiency, understood as a measure of resources consumed as a function of the size of the task tackled (Bundy, 2000). In the experiment the consumed resource is **CPU time** and the size of the task corresponds to how large, or complex, the synthesised model is. To compare the systems' consumption of CPU time we run them to synthesise a range of models of different complexities.

The metric we define for **complexity of a model**, denoted $cx(M)$, is the number of connections (flows and links) in the model M. It is meant to encompass the notions of size as well as connectedness of the models — complexity is used as an overarching term. This is why it is a count of connections, rather than a count of model elements (nodes), for example.

What is more, within our metadata-based modelling approach, the complexity of a model reflects the complexity of its metadata[1] — the larger and more connected the metadata set, i.e., the more relationships exist between the metadata descriptions, the larger and more connected the models synthesised from it will be.

9.3 Experimental Procedure

Synthesis-0 has a metadata set as single synthesis resource. The procedure to follow in order to measure the system's run times is simply:

(1) Take a sample of metadata sets Δ.

(2) Run Synthesis-0 over each Δ to synthesise a model M.

Figure 9.2: Experimental procedure for measuring Synthesis-0 run times.

Synthesis-$\mathcal{R}$ has two synthesis resources, as we know, a metadata set and a reference model. Both resources bound the synthesised models as prescribed by the system's algorithms and constraints. The reference model determines the structure (or topology)

[1] We could further argue that the complexity of the metadata reflects the complexity of the data. But we will go no further than the metadata level since this work only explores in depth metadata$\leftrightarrow$model associations as opposed to data$\leftrightarrow$metadata$\leftrightarrow$model associations.

of the synthesised model, while the metadata determines its content. The synthesised model will be structurally equal to or a subset of the reference model, and all its components must have metadata support. The ideal, yet unlikely, scenario would be to have both resources thoroughly exploited: synthesised and reference models with identical structures and no metadata evidence wasted.

We can visualise the achieved reuse of the reference model resource by comparing its structure with the structure of the synthesised model. In much the same way, we can also conveniently visualise the achieved use of the metadata resource. Synthesis-0 exhausts the metadata resource, so by applying it to the given metadata set we obtain models we call *target*, in that they are the best one can get as far as use of the metadata is concerned. Through a target model we can structurally relate the synthesised model to the metadata set, just as it relates to the reference model.

The three models are denoted:

M_{ref}: a reference model

M': a model synthesised from Δ, a metadata set, through reuse of M_{ref} by Synthesis-$\mathcal{R}$

M_{target}: a model synthesised from Δ by Synthesis-0

We set up appropriate scenarios for measuring Synthesis-$\mathcal{R}$ run times according to the space of possible relations between the three models. Such relations are the bounding relations we mentioned earlier between the synthesised model, M', and the reference model, M_{ref}, and between M' and metadata through M_{target}, the target model. M_{ref} and M_{target} are not directly related. The relations are as follows. Where the relation between the models is a structural rather than a conventional set relation we indicate it with an s.

- $M_{ref} \overset{s}{\not\subset} M'$, the synthesised model is a structural copy of the whole or parts of the reference model, never outgrowing the reference structure; and
- $M' \not\supset M_{target}$, the synthesised model does not have components other than those the metadata set supports, which are all part of the target model.

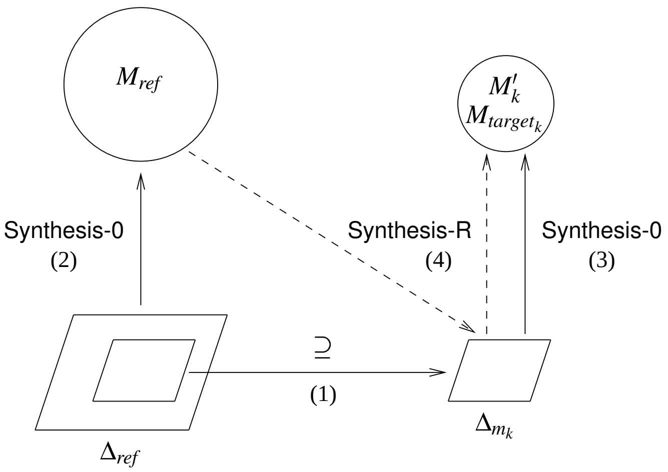
That leaves the complement relations:

$$\boxed{M_{ref} \overset{s}{\supseteq} M' \subseteq M_{target}}$$

That is, M' ranges from structurally equal to M_{ref}, when the reuse of the reference model is optimal, to a structural subset of M_{ref}, when the reuse of the reference model is sub-optimal; and likewise on the side of use of metadata, M' ranges from equal to a subset of M_{target}. As mentioned earlier, complete optimality is unlikely. So the normal scenario in practice should be $M_{ref} \overset{s}{\supset} M' \subset M_{target}$.

The pairwise relations between the models can be enforced by establishing corresponding subset relations, conventional subset relations this time, between the metadata sets of M_{ref} and M'; this causes the structural subset relation between these two models to become a conventional subset relation. Namely, to enforce $M_{ref} \overset{s}{\supseteq} M'$ we establish $\Delta_{ref} \supseteq \Delta_m$, and to enforce $M' \subseteq M_{target}$ we establish $\Delta_{ref} \subseteq \Delta_m$.

This is achieved by applying the twofold procedure shown in Figures 9.3 and 9.4. Part I of the procedure ensures $M_{ref} \supseteq M'_k = M_{target_k}$, and Part II ensures $M_{ref_k} = M'_k \subseteq M_{target}$, where k indicates subset instances, in line with the overall relations $M_{ref} \overset{s}{\supseteq} M' \subseteq M_{target}$.



(1) Given Δ_{ref}, the metadata set of the reference model, take Δ_{ref} or a subset of it as the metadata set, Δ_{m_k}, of the model to be synthesised — $\Delta_{ref} \supseteq \Delta_{m_k}$.

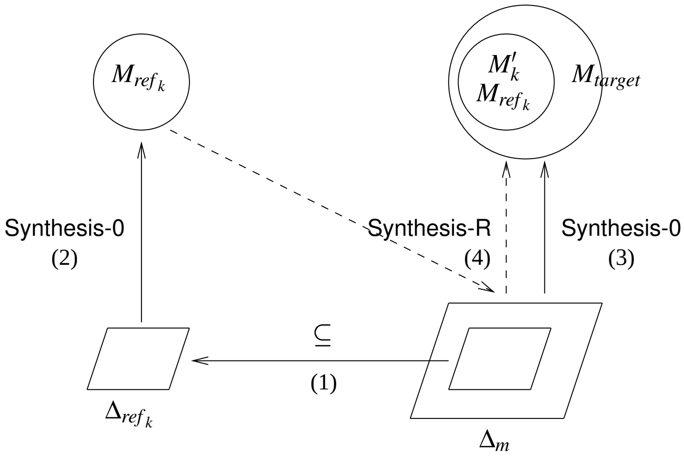
(2) Run Synthesis-0 to synthesise M_{ref} from Δ_{ref}.

(3) Run Synthesis-0 to synthesise M_{target_k} from Δ_{m_k}.

(4) Run Synthesis-$\mathcal{R}$ to synthesise M'_k through reuse of M_{ref} over Δ_{m_k}. In the case where $\Delta_{ref} = \Delta_{m_k}$, the whole M_{ref} will match Δ_{m_k} resulting in $M_{ref} = M'_k$. When $\Delta_{ref} \supset \Delta_{m_k}$, only parts of M_{ref} will match Δ_{m_k}, resulting in $M_{ref} \supset M'_k$ (the normal situation in practice).

It must hold that $M'_k = M_{target_k}$. Both systems are bound to fully exploit Δ_{m_k}: Synthesis-0 by definition, and Synthesis-$\mathcal{R}$ because Δ_{m_k} is a subset of the original metadata set from which M_{ref} was synthesised, so all of Δ_{m_k} must match M_{ref}.

Figure 9.3: Experimental procedure for measuring Synthesis-$\mathcal{R}$ run times. Part I: $M_{ref} \supseteq M'_k = M_{target_k}$.



(1) Given Δ_m, the metadata set of the model to be synthesised, take Δ_m or a subset of it as the metadata set, Δ_{ref_k}, of the reference model — $\Delta_{ref_k} \subseteq \Delta_m$.

(2) Run Synthesis-0 to synthesise M_{ref_k} from Δ_{ref_k}.

(3) Run Synthesis-0 to synthesise M_{target} from Δ_m.

(4) Run Synthesis-$\mathcal{R}$ to synthesise M'_k through reuse of M_{ref_k} over Δ_m. In the case where $\Delta_{ref_k} = \Delta_m$, M_{ref_k} will match the whole of Δ_m resulting in $M'_k = M_{target}$. When $\Delta_{ref_k} \supset \Delta_m$, M_{ref_k} will only match a subset of Δ_m, resulting in $M'_k \subset M_{target}$ (the normal situation in practice).

It must hold that $M_{ref_k} = M'_k$. Δ_{ref_k}, the metadata set from which M_{ref_k} was synthesised, is a subset of Δ_m, so all of M_{ref_k} must match Δ_m, giving rise to an identical M'_k.

Figure 9.4: Experimental procedure for measuring Synthesis-$\mathcal{R}$ run times. Part II: $M_{ref_k} = M'_k \subseteq M_{target}$.

9.3.1 Other Controls and Simplifications

In order to be able to say that any differences in performance are due to the feature differences we identified in the systems, they must be executed under the same experimental conditions.

The experimental procedure prescribes how to correctly replicate scenarios where the run times of the two systems are comparable. Again for comparability, the systems ought to run over the same material. As we show in the next section, all the metadata sets used originate from a single sample of models. Initially, a metadata set is generated to each sample model. Then, following the experimental procedure for Synthesis-$\mathcal{R}$, each of these metadata sets is partitioned into subsets.

Just as a reminder, if we were to compare the systems' run times without following this experimental procedure we would have a common metadata set, manually specified (like the one in Appendix B) over which the two systems would run and a reference model (for Synthesis-$\mathcal{R}$) with an undetermined metadata set.

Control is also needed concerning the systems' ability to find multiple model solutions (Section 7.3 and 8.4). In each run of Synthesis-0 we measure the time to synthesise the model solution that corresponds exactly to the sample model in question, rather than other possible solutions. In each run of Synthesis-$\mathcal{R}$ we measure the time to synthesise the model solution, M'_k, that is equal to M_{target_k}, if executing Part I of the experimental procedure, or equal to M_{ref_k}, if executing Part II.

Because of the controlled experimental procedure and the use of mostly artificial metadata we can easily make the systems synthesise the appropriate solutions in their first run. The first run is the worst case in CPU time consumption in that it takes little advantage of the efficiency-enhancing caching technique (Section 7.6) compared to subsequent backtracking runs.

Lastly, Synthesis-0 cannot synthesise disconnected parts of a single model graph in one run (see Section 7.3), whereas Synthesis-$\mathcal{R}$ can (see Appendix C). We had a few of these cases in the experiment. Since we only measure run time of the first run of each system for each example, we will have Synthesis-0 time of the first disconnected part of the model the system can find while the Synthesis-$\mathcal{R}$ time will be of the whole model. Nevertheless, this does not favour Synthesis-$\mathcal{R}$, the system we expect to be faster; on the contrary.

9.4 Data

This section concerns the experimental data and the procedures we have automated to collect it.

9.4.1 Models Sample

Models have been sampled from three well-known textbooks in system dynamics ecological modelling as indicated in Table 9.2 below.

Model	$cx(M_i)$	Source
M_1	7	Haefner (1996), p. 36 - with alterations
M_2	12	Haefner (1996), p. 36 - with alterations
M_3	18	Grant et al. (1997), p. 220
M_4	21	Ford (1999), p. 42
M_5	27	Ford (1999), p. 45
M_6	31	Grant et al. (1997), p. 159
M_7	33	Grant et al. (1997), p. 236
M_8	38	Grant et al. (1997), p. 261
M_9	43	Grant et al. (1997), p. 274
M_{10}	48	Ford (1999), p. 194 - with alterations
M_{11}	52	Grant et al. (1997), p. 290 - with alterations
M_{12}	56	Grant et al. (1997), p. 329 - with alterations

Table 9.2: Models sample.

Models have been included in the sample on the basis of model complexity, seeking for a representative, evenly spaced range of complexity. To achieve this we have added a few connections to some of the textbook models, namely, M_1, M_2 and M_{10}.

We also made alterations on models M_{11} and M_{12}. Their flow-to-flow links have been replaced by parameter-to-flow links, due to the synthesis systems, as currently implemented, not being able to handle the former type of links; and, parameters shared by many flows have been broken down into several parameters, in order to facilitate the partition of the models' metadata sets into subsets (see Section 9.4.3.1).

The range of models in the sample varies from simple yet non-trivial models to reasonably complex models, amongst the largest in the textbooks. Although this has not been taken into consideration for sampling, it is worth mentioning that these are models of a variety of ecological systems, such as animal population and vegetation dynamics, aquatic systems and energy balance of animals.

9.4.2 Generating Artificial Metadata

We generate an artificial metadata set for each model in the sample. The only exception is model $M6$, which corresponds to the pond management model we have referred to before in several parts of the thesis. The model's metadata set appears in Appendix B and has been manually specified as explained in Chapter 5. This example will illustrate the impact of non-artificial metadata on synthesis time.

The requirements for an artificially generated metadata set are two; it must:

- provide support for synthesis (through Synthesis-0 and Synthesis-$\mathcal{R}$) of the corresponding model; and
- be compliant with Ecolingua, i.e., contain valid instantiated Ecolingua terms only.

We have written a program which takes a model represented as a set of flow and link arcs (Section 7.2) and outputs a file containing a metadata set that meets such requirements. In designing the program we have arbitrarily fixed input settings, used to describe the models, and output settings, which determine the artificial metadata specifications to be generated to each model component. We illustrate the settings in the brief example below and leave the details to Appendix D.

Example

Let us take the least complex model in the sample, M_1, whose diagram appears in Figure 9.5.

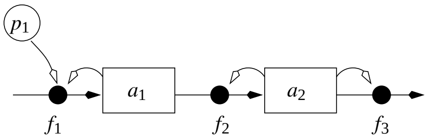


Figure 9.5: Sample model M_1 in system dynamics diagrammatic notation.

M_1's representation as a set of flows and links described according to the input settings is:

$$M_1 = \left\{ \begin{array}{l} \mathit{flow}(f_1, outside, sv(a_1, kg), kg/year), \\ \mathit{flow}(f_2, sv(a_1, kg), sv(a_2, kg), kg/year), \\ \mathit{flow}(f_3, sv(a_2, kg), outside, kg/year), \\ link(param(p_1, kg/kg/year), \mathit{flow}_{id}(f_1)), \\ link(sv(a_1, kg), \mathit{flow}_{id}(f_1)), \\ link(sv(a_2, kg), \mathit{flow}_{id}(f_2)), \\ link(sv(a_2, kg), \mathit{flow}_{id}(f_3)) \end{array} \right\}$$

Given M_1, the artificial metadata generation program produces the set of metadata specifications in Figure 9.6.

$model\_att(m)$
$model\_time\_unit(year)$

$data(abs\_rate(f_1, m, outside, e_1, kg/year))$
$model\_goal\_var(amt\_of\_mat(a_1, m, e_1, kg))$

$data(abs\_rate(f_2, m, e_1, e_2, kg/year))$
$model\_goal\_var(amt\_of\_mat(a_2, m, e_2, kg))$

$data(abs\_rate(f_3, m, e_2, outside, kg/year))$

$data(spf\_rate(p_1, f_1, m, kg/kg/year))$
$data(constant(p_1))$

$data(influences(a_1, f_1, ?))$
$data(influences(a_2, f_2, ?))$
$data(influences(a_2, f_3, ?))$

Figure 9.6: Artificial metadata set generated for sample model M_1.

9.4.3 Collecting Run Time Measurements

We also have programs that automate the experimental procedure for measuring Synthesis-0 and Synthesis-$\mathcal{R}$ run times. The automated procedure in Figure 9.7 corresponds, including the numbered steps, to the experimental procedure in Figure 9.2; likewise, Figure 9.8 corresponds to Figures 9.3 and 9.4.

---

(1) (2) For each metadata set Δ_i of the sample models M_i, $i = 1, 2, \ldots, 12$:
    Run Synthesis-0 over Δ_i to synthesise M_i measuring run time.

Figure 9.7: Automated procedure for collecting Synthesis-0 run time measurements.

For each metadata set Δ_i of the sample models M_i, $i = 1, 2, \ldots, 12$:
    Generate metadata subsets δ_k of Δ_i (Section 9.4.3.1);
    Run Synthesis-0 to synthesise a model N_k from each δ_k.

Part I

(1) Take each δ_k as Δ_{m_k}, the metadata set.
(2) Take Δ_i as Δ_{ref} and M_i as M_{ref}, the reference model.
(3) Take each N_k as M_{target_k}, the target model.
(4) Run Synthesis-$\mathcal{R}$ to synthesise M'_k measuring run time.

Part II

(1) Take Δ_i as Δ_m, the metadata set.
(2) Take each δ_k as Δ_{ref_k} and each N_k as M_{ref_k}, the reference model.
(3) Take M_i as M_{target}, the target model.
(4) Run Synthesis-$\mathcal{R}$ to synthesise M'_k measuring run time.

Figure 9.8: Automated procedure for collecting Synthesis-$\mathcal{R}$ run time measurements.

9.4.3.1 Generating Metadata Subsets

The procedure for collecting Synthesis-$\mathcal{R}$ run times (Figure 9.8) starts with the generation of subsets of the metadata sets. For representativeness and coverage the generated subsets should range in size and cover different parts of the original metadata set. At the same time, each subset should be such that a meaningful model can be synthesised from it.

The simplest meaningful system dynamics model would comprise a flow connected to a state variable and a few parameters connected to the flow. The idea of the metadata subsets generation procedure is to take subsets that support such models of minimal complexity, then take subsets that support models twice as complex, three times as complex, and so on, until we reach the set itself, which is taken as one of the subsets to conform with the experimental procedure[2] (see Figures 9.3 and 9.4).

As determined by the artificial metadata generation settings (Table D.3), the metadata sets, Δ_i, contain an *abs_rate* term, representing a rate quantity in the data set, as primary description to support the synthesis of each flow (the same applies to the manually specified metadata set Δ_6). To form a metadata subset the procedure first forms subsets of *abs_rate* terms and then finds terms directly and indirectly related to the *abs_rate* terms in each subset. An example should illustrate the procedure well. To keep the example concise we shall denote *abs_rate* terms by the rates' identifiers r_j (e.g., r_1 denotes $data(abs\_rate(r_1, m, outside, e_1, kg/year))$).

Example

Suppose a hypothetical metadata set Δ_i is given containing three *abs_rate* terms, namely, r_1, r_2 and r_3 (together with other terms of other kinds). The metadata subsets generation procedure:

> Finds the set A of all *abs_rate* terms in Δ_i, $A = \{r_1, r_2, r_3\}$; then
>
> Finds the set B which is the power set of A except $\emptyset$ and A itself, $B = \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_1, r_2\}, \{r_1, r_3\}, \{r_2, r_3\}\}$.

[2] But, obviously, it is not necessary to apply the procedure (even though it would work) to obtain the set itself.

Each $C \in B$ gives rise to a metadata set δ_k, $k = 1, 2, \ldots, 6$.

Let us take $C = \{r_1\}$. The procedure takes from Δ_i to compose δ_1:

- all terms directly related to r_1, which are:
  - terms that include the identifier r_1;
  - terms that include the identifier a_1; a_1 is obtained through the entity e_1 in the r_1 term and identifies the primary metadata description, representing an amount in the data set, necessary to synthesise the state variable of the flow to be synthesised from r_1.
- all terms indirectly related to r_1 through other data identifiers that are part of the terms above, and all terms directly and indirectly related to these data and so on, recursively, **except**:
  - terms that include the other *abs_rate* identifiers, i.e., r_2 and r_3 (generally, $r_j \notin C$); and
  - terms that include amount identifiers other than a_1 (generally, amount identifiers which are not directly related to any $r_j \in C$).
- the model requirements terms $model\_att(m)$ and $model\_time\_unit(year)$ (Table D.2) if Δ_i is artificial; or otherwise, the manually specified model requirements terms.

Note that it is the number of elements in B, which is $2^n - 2$ where n is the cardinality of A, that determines the number of subsets the procedure generates. In the case of Δ_6, Δ_{11} and Δ_{12}, with 7, 14 and 10 *abs_rate* terms respectively, taking in all possible elements of B would over-generate subsets (e.g., $2^{10} - 2 = 1022$). In these cases we sampled elements to form sets B limited to at most 50 elements. The number of subsets generated from all the metadata sets Δ_i, $i = 1, 2, \ldots, 12$, ranges between 2 and 50.

9.5 Results

The systems were executed using a Sun Blade 100 workstation, with a 502 MHz processor and 128 MB of main memory running SunOS 5.8. Measured run times correspond to elapsed CPU time, excluding time spent in low-level operations such as garbage collection and system calls.

9.5.1 Synthesis-0 Run Times

Figure 9.9 shows Synthesis-0 run times.

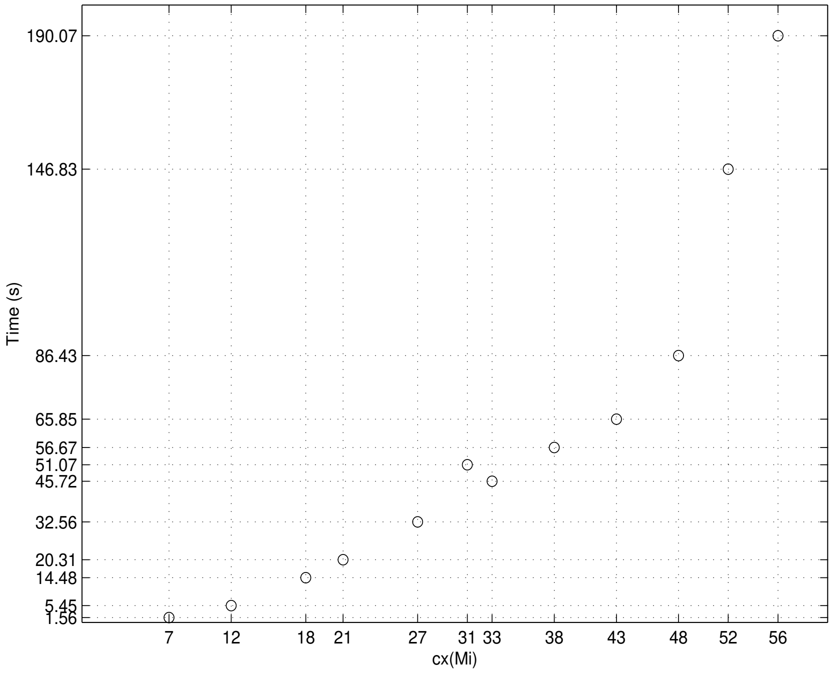


Figure 9.9: Synthesis-0 run times.

The plot suggests that synthesis time grows exponentially with model complexity. Using the system is still practical, though. The longest times are in the scale of a few minutes, which is a reasonable time, considering that the models that take this long are amongst the largest in the textbooks.

Flows, clusters and Synthesis-0 run time

Note the sharp increase in synthesis time from $cx(M_{10}) = 48$, just four complexity units lower, to $cx(M_{11}) = 52$. This illustrates the load of model structures (topologies) that are particularly demanding to synthesise: those with many flows and/or highly clustered elements. The demanding synthesis is due to the *model building* and *model components' constraints* features in Synthesis-0 (Table 9.1):

- Highly clustered nodes have an impact on synthesis time because of the way clusters are determined by Alg. 7.5. Finding a cluster involves computing an exhaustive set of possible arcs connected to the node, which can be costly when the metadata support many arcs. We consider highly clustered a node with 5 or more arcs connected to it. The threshold comes just from a visual inspection of the models. Compared to the other models in its complexity vicinities (complexity over 40), M_{11} has 6 highly clustered model elements while the others have 5 or less.

- Flows are the model component type with the hardest constraints (Section 6.3.3.1). M_{11} has 14 flows, the largest number of flows in the sample, followed by M_{12} with 10.

Artificial metadata and Synthesis-0 run time

The point at $cx(M_6) = 31$ is another to be noted. Unlike all the other models, M_6 is synthesised from a non-artificial metadata set. The restrictiveness of artificial metadata explores the space of Synthesis-0's functionalities narrowly, leading to a simplified, fast mode of execution. Some of the simplifications brought about by artificial metadata are listed below, in a loose decreasing order of impact on synthesis time. The effect of artificial/non-artificial metadata on synthesis time cannot be attributed to any of the features in Table 9.1 as Synthesis-0 and Synthesis-$\mathcal{R}$ do not differ in this respect.

- The metadata is artificially made "just right" for the model. There are no extraneous metadata in the set, so no dead-end searches take place as every metadatum retrieved turns out to be applicable.

- The metadata is fully Ecolingua-compliant. So no time is spent in unsuccessful ontological checks. Moreover, the terms are all simple (e.g., simple units of measure are used), keeping the ontological checks inexpensive.

- In the first run of the system the metadata support synthesis of strongly suggested model components only, which minimises computation of integrity constraints.

9.5.2 Synthesis-$\mathcal{R}$ Run Times

Synthesis-$\mathcal{R}$ run times are shown in the 3-D plot in Figure 9.10. The surface was obtained using the triangle-based linear interpolation method.

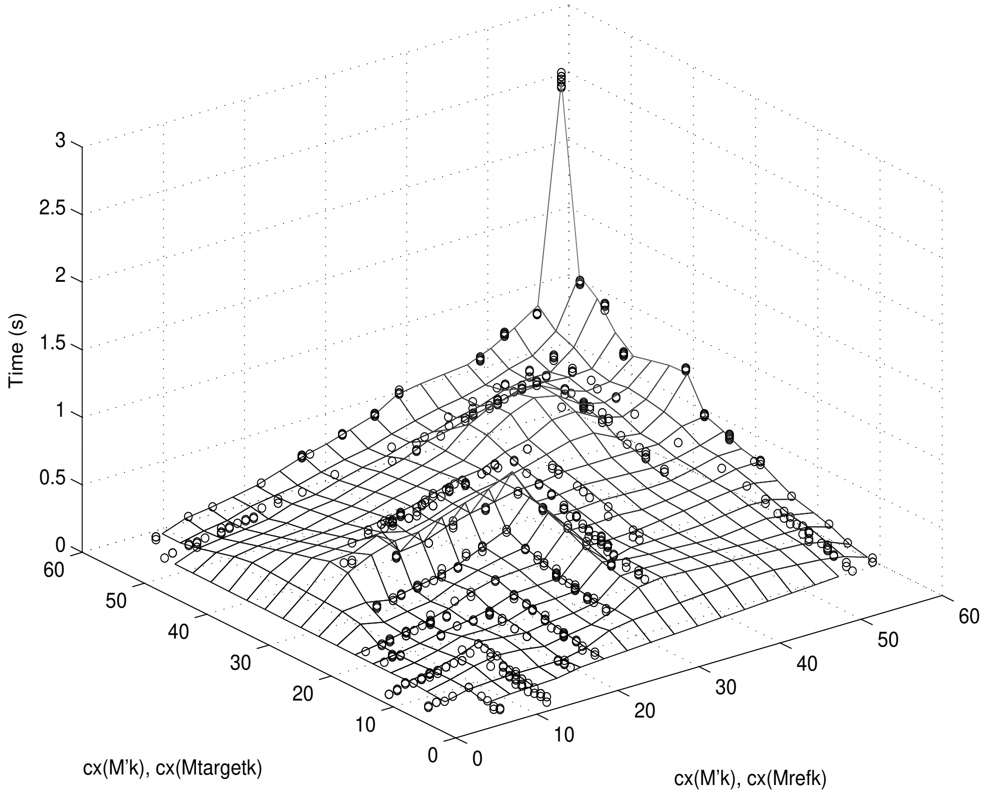


Figure 9.10: Synthesis-$\mathcal{R}$ run times.

We can distinguish a series of wedge-shaped groups of points along the surface. Each of these groups was collected by one execution of the procedure in Figure 9.8 for one of the twelve metadata sets Δ_i.

The descending slope of the surface (from an imaginary line across the wedge tips down to the right) corresponds to Part I of the procedure. It is easier to visualise this looking at Figure 9.3. To each of the groups of points there is one fixed Δ_{ref} and M_{ref}

to varying Δ_{m_k} and M_{target_k}, where $M_{target_k} = M'_k$, the synthesised model. Thus, for the points on this slope, the complexity of the synthesised model of which synthesis time is plotted, is the same as the complexity of the target model, $cx(M'_k) = cx(M_{target_k})$. These points represent scenarios where the metadata restrain the synthesis potential of the reference model resource.

Symmetrically, the points on the ascending slope derive from Part II of the experimental procedure illustrated in Figure 9.4. To each of the groups of points there is one fixed Δ_m and M_{target} to varying Δ_{ref_k} and M_{ref_k}, where $M_{ref_k} = M'_k$; so the complexity of the synthesised model is the same as the complexity of the reference model, $cx(M'_k) = cx(M_{ref_k})$. The scenarios represented here are those where it is the reference model that restrains the synthesis potential of the metadata.

The points at the tip of the wedges occur where $M_{ref} = M'_k = M_{target_k}$ and where $M_{ref_k} = M'_k = M_{target}$, the complexity of the three models being thus the same, with the synthesis potential of both reference model and metadata exploited to the full.

The complexity axes represent that the complexity of the synthesised model: (1) depends on the complexity of both the reference model and the metadata, the latter represented in the complexity of the target model; and (2) grows as the complexities of the reference and target models approximate each other.

On the whole, the surface suggests a linear, gradual increase of synthesis time as $cx(M'_k)$ increases with $cx(M_{target_k})$ and $cx(M_{ref_k})$, followed by a sudden increase where $cx(M'_k)$ approaches 60. In other words, the more metadata and the more complex the reference model, the longer, unsurprisingly, Synthesis-$\mathcal{R}$ takes to synthesise a model.

Selection of flows' local solutions and Synthesis-$\mathcal{R}$ run time

Let us look at the points at the peak of the surface. They show the synthesis time of M_{12} of complexity 56, the last model in the sample, given identical reference and target models. The high synthesis time value in comparison to the others, is due to Synthesis-$\mathcal{R}$'s *selection of local solutions* feature (Table 9.1). The system computes all sets of flow matches per type of flow per state variable in order to select the best (largest) ones. Two of this model's state variables have each three flows of the same type (out-flows) connected to them. Because the metadata set matches the reference model perfectly (a $\Delta_{ref} = \Delta_m$ case of the experimental procedure) many combinations of matching flows exist. It is computing these many combinations in this particular case that gives the sudden increase in synthesis time. Still, a time of about 2.5 seconds is significantly better than the 190.07 seconds Synthesis-0 takes to synthesise the same model.

In a non-experimental setting a case like this is unlikely as two conditions have to coincide:

- The topological characteristic of the model, i.e., state variable(s) having several flows of the same type. It is more common for state variables to have one or two flows of each type.
- A perfect match between reference model and metadata to allow for all possible combinations of matching flows. This only happens experimentally. In reality reference models will have been derived from (meta)data sets that are different from the metadata set of the model under synthesis.

Artificial metadata and Synthesis-$\mathcal{R}$ run time

Like in Synthesis-0, using non-artificial metadata (run of the procedure in Figure 9.8 for Δ_6) has a noticeable effect on Synthesis-$\mathcal{R}$ run times. It causes the ridge we can see half way up the surface, for the same reasons as in Synthesis-0 explained in the previous section, except for the third bullet point since in Synthesis-$\mathcal{R}$ metadata evidence classes (strong, weak) are not assigned to model components.

9.5.3 Comparative Result

The run time measurements of the two systems come in support of the experimental hypothesis. They show that Synthesis-$\mathcal{R}$ is indeed remarkably more efficient than Synthesis-0 as far as run time is concerned. While Synthesis-0 had run times up to 190.07 s, in approximately 600 runs of Synthesis-$\mathcal{R}$ all run times were under 3 s, with the vast majority under 1.2 s.

So using a reference model does improve synthesis performance. However, it should be clear that most of the time the reference model will bring some hindrance to the use of metadata evidence, which does not happen in Synthesis-0. Many different models may be synthesisable from a metadata set through reuse of a single reference model, some will have exploited the metadata set well, others will not. It all comes down to property similarities between the data set behind the metadata set and the data set behind the reference model. The more properties they share, the more of the reference model will be successfully matched with the metadata, and the more the synthesised models will reuse the reference model's structure.

In our experimental procedure (Section 9.3) reference models are synthesised from the metadata sets of our sample models. Outside an experimental setting such as this, the metadata set of the reference model would not be necessarily known, the reference model could possibly not be sufficiently adequate for reuse over the given metadata set of the model under synthesis. In these circumstances, alternative reference models would have to be sought for, and whichever method used — from trial-and-error to methods based on sophisticated selection criteria — would incur Synthesis-$\mathcal{R}$ to run at a higher computational cost. Such issues that selection of reference models involve are not dealt with in the evaluation experiment (see Section 10.2.1).

Alternatively, or complementarily, to reference model selection, the metadata-only and reuse synthesis approaches can be used in combination. Structures of distinct system dynamics models are often not radically different. For any given reference model and metadata set, chances are that, in the worst case, small chunks of the reference model will match the metadata. So, a way to at the same time make the best out of Synthesis-$\mathcal{R}$'s efficiency and Synthesis-0's optimal use of metadata would be to apply Synthesis-$\mathcal{R}$ first to cheaply get initial models, then maybe select the best ones and build them up from there by applying Synthesis-0 to exploit the rest of the relevant metadata, if any.

9.5.4 Generalisation

We now generalise the experimental results by identifying the defining factors, in problems and systems, that are essential for reproduction of the behaviour of the reuse technique as observed in the experiment. A causal explanation for the expected behaviour is given. One follow-up to this work should be testing this generalisation by, for example, applying the proposed reuse technique to model design problems of other domains.

To keep the generalisation concise we delay remarks until the end.

- ☐ The experimental results generalise to the *class of model design problems* where the *models* are:
  - ◇ *structural**; and
  - ◇ *backed by domain data*.

  Solution of such problems can be aided by *model synthesis systems* whose *architecture* comprises:
  - ◇ a *formal*† *ontology of domain data*; and
  - ◇ *definitions of connections*‡ *between the ontology concepts*§ *and model structure*;

  where the *system's task* is:
  - ◇ *given a data set specified in the ontology, to design a model informed by this data*.

- ☐ Taking the *specified data set as an instantiation of the ontology, the connections definitions can be used to perform the task* in (at least) two ways:
  - ◇ *by establishing the connections between the specified data set and a model, which directly constructs the resulting model*; or
  - ◇ *by establishing connections between the structure of an existing model and the specified data set*; *where these connections are reused to form the resulting model*.

- ☐ We propose that:
  - ◇ *the latter reuse-oriented way drastically improves the efficiency of the system at the cost of possible waste of data*

  because
  - ◇ *the structural knowledge embedded in the reused model*:
    - ◇ *bounds the state space*; and
    - ◇ *reduces computation for establishing the connections, in that it already incorporates ontological concepts and relations between them which only need to be matched with the specified data. In most cases the match will not be perfect causing some data to be wasted*.

---

*In the sense of having a conceptual framework which is distinguishable from their content. Structural models are usually compositional. Given a certain type of structural models in a certain domain, all model instances, each representing a specific system in the domain, will share the same framework components, but will differ in the components' assembly and content.

†To give semantics to the connections in the next item.

‡Which can be many-to-many and interrelated.

§Because model design is informed by high-level data properties rather than by low-level data (see Section 1.2).

Chapter 10

Looking Back and Looking Ahead

In this final chapter we look back on the thesis contributions and speculate on directions for further work.

10.1 Contributions

The three major contributions of the thesis are summarised in the sections below.

10.1.1 A Mechanism for Model Reuse via Synthesis and Data Ontologies

We have described a method in which models are synthesised from data described through an ontology, or metadata, by reusing existing models (Section 8.1). This is possible because connections from metadata to model structure have been identified and logically defined. Knowing how to draw a model from metadata enables us to take the converse, more efficient approach of matching and refilling the structure of an existing model with new metadata.

The ontology plays a crucial role in the usefulness of the mechanism. It captures properties of domain data that are not specific to particular data sets. The connections are then defined between these properties and model structure. Thereby, the reuse-synthesis mechanism can be applied across models and data sets of the domain concerned.

In realising the mechanism we produced:

A constraints-based characterisation of connections between ecological metadata and system dynamics model structure (Chapter 6).

There are *metadata constraints* which define the ontological data properties that must hold for model components to be established, and *integrity constraints* whose solution prevent models from having incompatible or conflicting components.

The metadata constraints make explicit data properties that connect to model structure. Such properties are not evident to ecologists when designing models because quantitative data and conceptual model structures belong to different levels of abstraction. When described through the ontology, the data's level of abstraction is raised and connections to model structure can be drawn. The metadata constraints formalise such connections. Moreover, unraveling the data properties has the desirable effect of fostering model-driven data collection. Ecological data sets today are often unnecessarily large yet short of data evidence needed to support the particular models to which they are intended.

Solution of the constraints is at the core of two working synthesis systems we have built:

Synthesis-0, which synthesises models from metadata alone (Chapter 7); and

Synthesis-$\mathcal{R}$, a more efficient system that incorporates the model reuse mechanism (Chapter 8).

The synthesis constraints are essentially the same in both systems except that they are employed in opposite orientations: to connect metadata to model structure in the former, and to connect a (reused) model structure to metadata in the latter (Section 8.6).

A great deal of ecological modelling knowledge is subjective, so the best one can expect as outcome from automated modelling mechanisms is prototypical models which can subsequently be enhanced by human modellers. Such mechanisms can be employed, for example, to supply the standard models and templates that modern ecological modelling tools offer to users. Our two synthesis systems are mechanisms of this kind. However, the prototypical models they synthesise have the advantage of being consistent with the properties of the supporting data, which will counterbalance the inevitable bias towards the modeller's own interpretation of the real system.

Of the two systems Synthesis-$\mathcal{R}$ was empirically shown to have a superior efficiency, leading to the next main contribution of the thesis.

10.1.2 A Procedure for Empirical Evaluation of Synthesis Systems with Reuse

We have described a procedure through which a reuse-synthesis system can be evaluated in relation to a corresponding system that does not incorporate the reuse mechanism (Chapter 9). We applied it to evaluate our mechanism's run-time efficiency in particular, but the procedure is also applicable to evaluate mechanisms in terms of other performance criteria. It lays down steps for setting scenarios where the two mechanisms (with and without reuse) are run over metadata sets that are interrelated in a way so as to give measures of synthesis performance that are empirically comparable.

In generalising our experimental results we propose:

A theory of increased efficiency of synthesis systems furnished with the reuse mechanism.

The theory applies in general to the problem of designing structural models that are backed by domain data (Section 9.5.4). It identifies the architectural features of synthesis systems applied to such problem that are necessary for reproduction of the behaviour observed in the experiment, and in the light of that explains why the reuse mechanism brings about an efficiency gain in the systems. The experimental procedure given can be used to test the theory in other domains.

10.1.3 A Complete Process of Ontology Engineering and Application

The construction of Ecolingua, the ontology at the basis of our synthesis mechanisms, involved:

Reuse of multiple ontologies in ontology conceptualisation and translation.

In Ecolingua's conceptualisation were reused concepts from five other ontologies, all available from an actively and widely used[1] web-based tool for ontology construction (Section 4.1).

The unwieldy outcome of the tool's translation service into our target implementation language led to the next contribution as part of the ontology engineering process.

A method for re-engineering a large implementation-level ontology obtained via theory inclusion of reused ontologies.

The tool's translation service gives an ontology specification that comprises the full content of all ontologies that Ecolingua directly and indirectly refers to through a relatively small number of existing definitions that were explicitly included in Ecolingua's own definitions. The re-engineering method reduces this specification in scope and content and produces appropriate representations in the target implementation language (Section 4.2). The last step in the method — refinement of target language expressions — is of course language-specific. Ecolingua's implementation language

[1] Today there are about 200 ontologies published in the Ontolingua Server ontology library.

is Prolog. So we developed a translator program to transform the KIF-like Prolog notation of the original translated ontology into legitimate Horn clauses.

Such engineering effort ultimately yielded a practical Ecolingua which, together with synthesis, enables our model reuse mechanism.

Ecolingua: a prototype ontology for description of ecological data (Chapter 3 and Appendix A).

Ecolingua concepts, for the most part, consist of classes of ecological quantities found in modelling data sets. The axioms of a class characterise its quantities in terms of their physical dimension, which can be captured through the unit of measure they are expressed in. The synthesis mechanisms involve proofs over these axioms in order to validate the metadata descriptions that substantiate the models (Section 7.4.1).

Notably, in this research project the ontology was not an end on itself but was built to serve a purpose in metadata-based model synthesis. As part and parcel of the model reuse mechanism it demonstrates a novel and practical use of ontologies in enabling knowledge reuse.

10.2 Further Work

The work in this thesis has a methodological theme. It articulates a method for construction, application, and evaluation of reuse-oriented automated modelling mechanisms enabled by ontologies. The method is outlined below.

1. Identify a conceptual model design problem in a domain;
2. Identify the kind of information used to substantiate models of systems in the domain;
3. Build an ontology from scratch or through reuse of other ontologies, or simply reuse a suitable one if available, for semantic annotation of such information;
4. Build a mechanism from scratch or through reuse of methods and other mechanisms, or simply reuse a suitable one if available, for construction of models based on the annotated information and reuse of existing models;

5. Apply the mechanism to efficiently obtain models of systems of interest;
6. Evaluate the mechanism's performance, if desired.

This method has been used in research only. It is desirable to make it into a more mature method that is useful and usable to knowledge engineers and across domains, with a suite of tools to accompany. This opens up a number of methodological and technological issues, some of which we discuss below.

10.2.1 Availability, Selection and Modularity of Reusable Models

Reusability of course presupposes availability. A traditional approach to availability is to have reusable pieces of knowledge organised in repositories (e.g., libraries of PSMs — Problem Solving Methods, or model fragments in compositional modelling). Selection of adequate models, methods, components, etc., that are adequate to the task at hand is an issue that immediately follows. In the context of our method, the primary criterion for selection of models for reuse should be their adequacy with respect to the information annotated to support the new model.

In the thesis (Chapter 8) the reused models (so called *reference models*) are entire models. As we argue in Section 9.3, it is unlikely for perfect adequacy to occur between a reference model and given metadata, leading to waste of either structural knowledge in the reference model or metadata evidence. If, however, instead of entire models we try and reuse smaller, more general model modules, the chances are that their adequacy to the metadata will increase. We would then need techniques to determine the right scope (size) of the modules so as to make them small enough to maximise reusability, yet large enough to contain significant structural knowledge. We would also need techniques to appropriately combine modules to obtain fully-formed models. Compositional modelling is a research area to turn to for such techniques.

So, in summary, model reusability may be enhanced through principled modularisation and composition. We can also think of ways to maximise use of annotated information. We suggested in Section 9.5.3 that a model can be reused first to cheaply produce an initial new model which can then be expanded by a non-reuse-oriented mechanism

(Synthesis-0 style) that exhausts, at a higher cost, the modelling potential of the annotated information.

10.2.2 Web-Enabling and Tools

The World Wide Web is the obvious medium where to make repositories of reusable models and reuse tools distributed and publically available. And since the method involves ontology building and reuse, one should consider their specification in semantic web languages (e.g., RDF, DAML+OIL) which are now strong contenders to become standard for ontology specification.

Web tools that are being built around these languages (e.g., Protégé-2000) can be valuable for the application of the method. For instance, there are annotation tools to link information — of the kind that substantiates models — with their descriptional ontology. In Chapter 5 we showed a purely manual, error-prone process of specifying data in Ecolingua to obtain metadata[2] that could be facilitated by one of these tools. Furthermore, agency technology using the semantic web infrastructure is too attractive a promise to be ignored. We are interested in building software agents able to find adequate models for reuse in model repositories and to understand capabilities of automated modelling mechanisms accessible through the Web.

10.2.3 Domain Modelling Languages

Unlike ontology languages, modelling languages are not amenable to standardisation across domains. Each domain has distinct problem solving practices to which the modelling languages are intrinsically related — to model is essentially to transform one problem representation into another that is more suitable for problem solving.

Having said that, it would be interesting to see how well the method applies in the context of other modelling languages and practices, and UML (Unified Modelling Lan-

[2]The term 'metadata' in semantic web jargon refers in general to semantic annotations, i.e. ontology-annotated information.

guage) for software development, would be our first target.

This brings up the question of representativeness of the empirical results in Chapter 9 on efficiency of our model reuse mechanism and applicability of the theory we propose based on those (Section 9.5.4). We saw in Section 9.5.2 that the efficiency of the mechanism is sensitive to the topology of the reused model — which will vary across domain modelling languages — in particular, to the combinatorics involved in searching for structures for the new model that match highly connected nodes in the reused model.

The evaluation step of the method can be revisited to look at application of optimised search algorithms, complexity analysis, and other performance criteria.

10.2.4 Link with Quantitative Modelling

A good thing about constructing conceptual models through the method is that these are more general and thus more widely applicable than quantitative models. However, there are domains, such as the empirical sciences, where conceptual models make more impact when translated to their quantitative form (Section 1.2).

Conceptual system dynamics models, which we synthesise in the thesis, function as frameworks for equation design (Section 2.4). Synthesis mechanisms such as ours, underpinned by higher-level data properties, could be integrated with equation induction methods to reduce their hypothesis space and, hence, the large number of spurious equations they induce (Section 2.3.3).

Equation design is followed by estimation of parameters to calibrate the equations. In (Brilhante, 1999; Brilhante and Robertson, 2001) we present preliminary work on metadata-based composition and decomposition of ecological functions for parameter estimation.

Appendix A

Low-Level Ecolingua

We call low-level Ecolingua the subset of the ontology's concepts which are not descriptional, i.e., not directly employed to describe ecological data instances. These relations are part of the interpretation constraints of the descriptional concepts (Section 3.1).

The axiomatisation of low-level Ecolingua consists of a rendition of the parts of the EngMath ontology that are relevant for our application. The axiomatisation is given in this appendix in Horn clause notation, assuming a resolution-based system.

The EngMath quotations and references in the sequel are from (Gruber and Olsen, 1994). We also make references to the specification of the EngMath ontology available in the Ontolingua Server (Ontolingua Server, 1995).

A.1 Ecolingua **Quantities and their Units and Scales**

Ecolingua **quantities**

Q is an Ecolingua quantity if Q identifies an instance of any of the Ecolingua quantity classes or an instance of a model goal variable.

$$\begin{aligned} eco\_qtty(Q) \leftarrow\ & amt\_of\_mat(Q,Mt,E,U) \lor mat\_dens(Q,Mt,E,U) \lor \\ & amt\_of\_time(Q,U) \lor amt\_of\_money(Q,E,U) \lor \\ & abs\_rate(Q,Mt,E_{from},E_{to},U) \lor spf\_rate(Q,R_{abs},Mt,U) \lor \\ & temperature\_of(Q,E,S) \lor weight\_of(Q,E,U) \lor \\ & number\_of(Q,E,U) \lor percentage(Q,E,U) \lor \\ & (model\_goal\_var(Q_{term}) \land name\_qtty(Q,Q_{term})) \end{aligned}$$

Q_{term} is a quantity term of a class suitable for simulation if Q_{term} is of any of the defined Ecolingua quantity classes except the class Amount of Time.

$$sim\_qtty\_class(Q_{term}) \leftarrow Q_{term} \in \left\{ \begin{array}{l} amt\_of\_mat(A,Mt,E,U), mat\_dens(A,Mt,E,U), \\ amt\_of\_money(A,E,U), abs\_rate(R,Mt,E_{from},E_{to},U), \\ spf\_rate(R,R_{abs},Mt,U), temperature\_of(T,E,S), \\ weight\_of(W,E,U), number\_of(N,E,U), \\ percentage(P,E,U) \end{array} \right\}$$

Q is a dimensionless quantity specified in U if the dimension of unit U is the identity dimension.

$$dimensionless\_qtty(Q,U) \leftarrow unit\_dimension(U,identity\_dimension)$$

Unit of mass

U is a unit of mass if U is a unit of measure of the mass dimension.

$$mass\_unit(U) \leftarrow unit\_of\_measure(U) \land unit\_dimension(U,mass)$$

Unit of power of length

U is a unit of a power of length if U is a unit of measure of the length dimension (power is 1) or, more generally, of the length to a real power dimension.

$$\begin{aligned} length^n\_unit(U) \leftarrow\ & unit\_of\_measure(U) \land \\ & \left(\begin{array}{l} unit\_dimension(U,length) \lor \\ (unit\_dimension(U,length^N) \land realnum\_expr(N)) \end{array} \right) \end{aligned}$$

Unit of material

U is a unit of material if U is a unit of mass or if U is equivalent to an expression Um/Ul, where Um is a unit of mass and Ul is a unit of a power of length.

$$mat\_unit(U) \leftarrow mass\_unit(U)$$
$$mat\_unit(U) \leftarrow eqv\_expr(U, Um/Ul) \wedge mass\_unit(Um) \wedge length^n\_unit(Ul)$$

Unit of money

U is a unit of money if U is a unit of measure of the money dimension.

$$money\_unit(U) \leftarrow unit\_of\_measure(U) \wedge unit\_dimension(U, money)$$

Unit of time

U is a unit of time if U is a unit of measure of the time dimension.

$$time\_unit(U) \leftarrow unit\_of\_measure(U) \wedge unit\_dimension(U, time)$$

Identity unit

Ontolingua Server: `Identity-Unit` is an individual in the Physical-Quantities ontology (which is part of the EngMath family of ontologies), defined in KIF as (= `Identity-Unit` 1).

$$identity\_unit(1)$$

Scale of temperature

S is a temperature scale if S is a scale of the temperature dimension.

$$temperature\_scale(S) \leftarrow scale\_dimension(S, temperature)$$

A.2 Units and Scales of Measure

U is a unit of measure if U is the base unit for a fundamental dimension in the defined system of units (see Section A.4).

$$unit\_of\_measure(U) \leftarrow bu\_fdim(U,D)$$

Units named by convention are units of measure (Massey, 1986).

A conventional unit of mass:

$$unit\_of\_measure(g)$$

Some conventional units of length:

$$unit\_of\_measure(cm)$$
$$unit\_of\_measure(km)$$
$$unit\_of\_measure(inch)$$
$$unit\_of\_measure(foot)$$
$$unit\_of\_measure(mile)$$
$$unit\_of\_measure(ha)$$

Some conventional units of time:

$$unit\_of\_measure(min)$$
$$unit\_of\_measure(hour)$$
$$unit\_of\_measure(day)$$
$$unit\_of\_measure(year)$$

EngMath: Every product of units is also a unit of measure, and the product is commutative.

$$\begin{aligned} unit\_of\_measure(U) \leftarrow\ & eqv\_expr(U, V \times W) \wedge \\ & unit\_of\_measure(V) \wedge unit\_of\_measure(W) \end{aligned}$$

EngMath: Every real-valued exponentiation of a unit is also a unit of measure.

$$\begin{aligned} unit\_of\_measure(U) \leftarrow\ & eqv\_expr(U, V^X) \wedge U \neq V \wedge realnum\_expr(X) \wedge \\ & unit\_of\_measure(V) \end{aligned}$$

EngMath: Units of measure are positive.

$$\begin{aligned} unit\_of\_measure(U) \leftarrow\ & eqv\_expr(U, X \times V) \wedge arithm\_expr(X) \wedge \\ & unit\_of\_measure(V) \wedge eval(X, 1) \end{aligned}$$

Unit identities

Identities of conventional units (the $\equiv$ symbol to mean that, for example, g is not merely equal to or equivalent to $0.001 \times kg$ but that g *is* $0.001 \times kg$) (Massey, 1986).

$$
\begin{aligned}
g &\equiv Expr \leftarrow eqv\_expr(Expr, 0.001 \times kg) \\
cm &\equiv Expr \leftarrow eqv\_expr(Expr, 0.01 \times m) \\
km &\equiv Expr \leftarrow eqv\_expr(Expr, 1000 \times m) \\
inch &\equiv Expr \leftarrow eqv\_expr(Expr, m/39.37) \\
foot &\equiv Expr \leftarrow eqv\_expr(Expr, 12 \times inch) \\
mile &\equiv Expr \leftarrow eqv\_expr(Expr, 5280 \times foot) \\
ha &\equiv Expr \leftarrow eqv\_expr(Expr, 10000 \times m \times m) \\
min &\equiv Expr \leftarrow eqv\_expr(Expr, 60 \times s) \\
hour &\equiv Expr \leftarrow eqv\_expr(Expr, 60 \times min) \\
day &\equiv Expr \leftarrow eqv\_expr(Expr, 24 \times hour) \\
year &\equiv Expr \leftarrow eqv\_expr(Expr, 365 \times day) \vee \\
& eqv\_expr(Expr, 366 \times day)
\end{aligned}
$$

A.2.1 Scales of Measurement

Scales in general are not the same as units of measure. They are not a standard quantity against which other quantities of the same physical dimension can be compared. They "have reference origins and can have negative values. Units are like distances between points on a scale" (Ontolingua Server).

Actually scales subsume units of measure. Units can be seen as scales of type ratio which are characterised by having an absolute zero. For example, K (kelvin) used for measuring temperature is an absolute (or ratio) scale and thus can also be called a unit of measure. ℃ (degree Celsius) and ℉ (degree Fahrenheit), on the other hand, are scales that cannot be called units of measure.

This distinction between scales and units is made in the EngMath ontology (even though the individual `Degree-Celcius` appears as an instance of the `Unit-Of-Measure` class in the ontology's specification in the Ontolingua Server).

In Ecolingua we define K as a unit of measure (Section A.4) and define ℃ and ℉

through scale correspondences in relation to K (Massey, 1986):

$$scale\_corresp('K', \,^{\circ}C, \; X\,^{\circ}C = (273.15 + X)\; 'K')$$
$$scale\_corresp(^{\circ}C, \,^{\circ}F, \; X\,^{\circ}C = (9/5 \times X + 32)\; ^{\circ}F)$$

$$scale\_corresp(S, S', Formula) \;\leftarrow\; scale\_corresp(S', S, Formula)$$

We cannot have identities between the temperature scales (like the ones for units) because the zeros of the scales do not coincide. The left and right-hand sides of the equality in the formulae above represent points in the different scales at the same temperature.

A.3 Dimensions of Units and Scales

EngMath: Units of measure are also quantities and as such are characterised by their dimensions.

Units' dimension

Unit U is of dimension D if, within a defined system of units (Section A.4), U is the base unit of a fundamental dimension D; or if U is a derived unit of a fundamental or composite dimension D; or if U is a non-physical unit of a fundamental or composite dimension D.

$$\begin{aligned} unit\_dimension(U,D) &\;\leftarrow\; bu\_fdim(U,D) \\ unit\_dimension(U,D) &\;\leftarrow\; du\_fdim(U,D) \vee du\_cdim(U,D) \\ unit\_dimension(U,D) &\;\leftarrow\; npu\_fdim(U,D) \vee npu\_cdim(U,D) \end{aligned}$$

Axiom of the Physical-Quantities ontology (Ontolingua Server):

$(= \; (\texttt{Quantity.Dimension Identity-Unit})\; \texttt{Identity-Dimension})$.

$$unit\_dimension(U, identity\_dimension) \;\leftarrow\; identity\_unit(U)$$

Scales of the temperature dimension

Scale S is of the temperature dimension if S is a unit of measure which is the base unit or a derived unit of the fundamental dimension temperature.

$$scale\_dimension(S, temperature) \leftarrow unit\_of\_measure(S) \wedge \begin{pmatrix} bu\_fdim(S, temperature) \vee \\ du\_fdim(S, temperature) \end{pmatrix}$$

Scale S is of the temperature dimension if there is a formula of correspondence between a measure in S and a measure in S' and S' is of the temperature dimension.

$$\begin{aligned} scale\_dimension(S, temperature) \leftarrow\ & scale\_corresp(S, S', Formula) \wedge \\ & scale\_dimension(S', temperature) \end{aligned}$$

A.4 System of Units

EngMath: "A system of units is a class of units defined by composition from a base set of units, such that every instance of the class is the 'standard' unit for a physical dimension and every physical dimension has an associated unit ...

To define a system of units, the model builder chooses a set of fundamental dimensions that are orthogonal (i.e., not composable from each other) ... For each of the fundamental dimensions, the model builder chooses a standard unit of that dimension; these are called the base-units of the system ... For example, the [Système International d'Unités][1] (SI) is a system of units that defines a set of seven fundamental dimensions with the base-units meter [m, for length], kilogram [kg, for mass], second [s, for time], ampere [A, for electric current], Kelvin [K, for temperature], mole [mol, for amount of substance], and candela [cd, for luminous intensity][1]."

Base units of fundamental dimensions

Our chosen fundamental dimensions are mass, length, time, temperature and money. For the base units we borrow kg for mass, m for length, s for time and K for temperature from SI and define the base unit $ for the money dimension.

[1] (Massey, 1986)

$$bu\_fdim(kg, mass)$$
$$bu\_fdim(m, length)$$
$$bu\_fdim(s, time)$$
$$bu\_fdim('K', temperature)$$
$$bu\_fdim(\$, money)$$

EngMath: "The set of fundamental dimensions determines the space of possible quantities that can be described in this system — those whose physical dimensions are some algebraic combination of the fundamental dimensions."

Derived units are part of this space of possible quantities. Direct comparison with a standard (through one of the base units) is feasible for only a limited number of physical quantities. So the units for most quantities must be derived from (or defined in terms of) a few base units (Massey, 1986).

EngMath: Every derived unit in the system is a composition, using multiplication and exponentiation to a real power, of units from the base set.

Derived units of fundamental dimensions

U is a derived unit of a fundamental dimension D if an identity exists between U and an expression *Expr* which is a derived unit of D.

$$du\_fdim(U,D) \leftarrow U \equiv Expr \wedge du\_fdim(Expr,D)$$

U is a derived unit of a fundamental dimension D if U is equivalent to an expression $X \times V$ where X is a positive real number and V is the base unit or a derived unit of D.

$$\begin{aligned} du\_fdim(U,D) \leftarrow\ & eqv\_expr(U, X \times V) \wedge U \neq V \wedge \\ & posrealnum\_expr(X) \wedge \\ & (bu\_fdim(V,D) \vee du\_fdim(V,D)) \end{aligned}$$

Derived units of composite dimensions

U is a derived unit of a composite dimension D if an identity exists between U and an expression *Expr* which is a derived unit of D.

$$du\_cdim(U,D) \leftarrow U \equiv Expr \wedge du\_cdim(Expr,D)$$

The composition of a derived unit accord with the composition of the dimension. Physical dimensions are composed from other dimensions using multiplication and exponentiation to a real power.

$$\begin{aligned} du\_cdim(U,D) \leftarrow\ & eqv\_expr(U, V \times W) \wedge \\ & (eqv\_expr(D, A \times B) \vee D \equiv A \times B) \wedge \\ & unit\_dimension(V,A) \wedge unit\_dimension(W,B) \end{aligned}$$

$$\begin{aligned} du\_cdim(U,D) \leftarrow\ & eqv\_expr(U, V^X) \wedge U \neq V \wedge realnum\_expr(X) \wedge \\ & (eqv\_expr(D, A^X) \vee D \equiv A^X) \wedge D \neq A \wedge \\ & unit\_dimension(V,A) \end{aligned}$$

Non-physical units and dimensions

$ is the unit of the non-physical fundamental dimension money.

$$npu\_fdim(\$, money)$$

U is the unit of a non-physical composite dimension D if U is equivalent to an expression $V \times \$$ and D is equivalent to an expression $A \times money$ where A is the dimension of the unit V.

$$\begin{aligned} npu\_cdim(U,D) \leftarrow\ & eqv\_expr(U, V \times \$) \wedge \\ & eqv\_expr(D, A \times money) \wedge \\ & unit\_dimension(V,A) \end{aligned}$$

Dimension identities

$$\begin{aligned} area \equiv Expr &\leftarrow eqv\_expr(Expr, length^2) \\ volume \equiv Expr &\leftarrow eqv\_expr(Expr, length^3) \\ frequency \equiv Expr &\leftarrow eqv\_expr(Expr, 1/time) \end{aligned}$$

A.5 Reals and Real-Valued Expressions

X is a real-valued expression if X is a real number or if X is an arithmetic expression that evaluates to a real number.

$$
\begin{aligned}
realnum\_expr(X) \;\leftarrow\;& X \in \mathbb{R} \vee \\
& (arithm\_expr(X) \;\wedge\; eval(X,Y) \;\wedge\; Y \in \mathbb{R})
\end{aligned}
$$

Likewise to positive reals:

$$
\begin{aligned}
posrealnum\_expr(X) \;\leftarrow\;& X \in \mathbb{R}^{+} \vee \\
& (arithm\_expr(X) \;\wedge\; eval(X,Y) \;\wedge\; Y \in \mathbb{R}^{+})
\end{aligned}
$$

Appendix B

Metadata Specifications of the Pond Management Example

This appendix contains Prolog-encoded metadata specifications for synthesis of a pond management model. The metadata sources (data tables and statements, model objectives and assumptions statements) are extracts from (Grant et al., 1997), Chapter 9 — 'Annotated Example of Model Development and Use: Simulation of Aquaculture Pond Management'.

The extracts are not exhaustive — they are a sample of data and statements that refer to the objects and relations described. Some descriptions are reiterated by later extracts. Units of measure specified later in the chapter have been added to the extracts, where needed.

```
%%% File:     metadata_pond.pl
%%% Author:   Virginia Brilhante
%%% Purpose:  Metadata specifications for synthesis of the Pond
%%%           Management Model (Grant et al. (1997), Chapter 9)


%%%-------------------------------------------------------------------
%%%                   Metadata from Field Data
%%%-------------------------------------------------------------------
```

```
%%%  From Table B.1

data(mat_dens(aquatic_plants_biomass, biomass, aquatic_plants, kg/ha)).
data(entity(aquatic_plants)).
data(material(biomass)).

data(mat_dens(fish_biomass, biomass, fish, kg/ha)).
data(entity(fish)).

data(weight_of(indiv_fish_weight, indiv_fish, kg)).
data(entity(indiv_fish)).

data(temperature_of(water_temp, water, celsius)).
data(entity(water)).


%%%  From Table B.2

data(spf_rate(plant_production_rate, plant_production, biomass, g/kg/day)).
data(influences(water_temp, plant_production_rate, ?)).

data(spf_rate(consumption_rate, consumption, biomass, kg/kg/day)).
data(influences(water_temp, consumption_rate, ?)).
data(influences(indiv_fish_weight, consumption_rate, ?)).

model_time_unit(day).

%%%  ``It also is known that consumption by fish becomes limited by
%%%    plant biomass when the plant biomass falls below 20,000 kg/ha
%%%    and that the consumption rate is reduced by roughly one-half
%%%    for every 5,000 kg/ha decrease in plant biomass below that
%%%    point.''

data(influences(aquatic_plants_biomass, consumption, ?)).
data(influences(aquatic_plants_biomass, consumption_rate, +)).


%%%-----------------------------------------------------------------
%%%                 Metadata from Model Objectives
%%%-----------------------------------------------------------------

%%%  ``The general objective of the model can be stated quite simply:
%%%    to determine if any of several alternative stocking and
%%%    harvesting schemes will yield higher than current profits.
%%%    Specific questions of interest include the following:
%%%    1. What size harvest is expected in a ``normal'' year under the
%%%    usual scheme of stocking 75 0.227-kg fish on April 15 and
%%%    harvesting them on November 15, and what profit is associated
%%%    with this harvest ... ?
```

```
model_goal_var(amt_of_money(profit, pond_system, $)).


%%%-------------------------------------------------------------------
%%%                Metadata from Model Assumptions
%%%-------------------------------------------------------------------

%%%  ``... the biomass of harvested fish [kg/ha] is converted into
%%%    dollars at the end of the year.''
%%%  ``Profit is calculated based in part on the biomass of fish
%%%    accumulated on the harvest [kg/ha/day] date, which depends on
%%%    the net accumulation of fish biomass in the pond since stocking
%%%    [kg/ha/day]. Therefore, we need to represent fish biomass
%%%    dynamics. Because fish are herbivorous, their growth depends in
%%%    part on the amount of plant biomass accumulated in the pond at
%%%    any given time, so we also need to represent plant biomass
%%%    dynamics.''

data(influences(harv_fish_biomass, profit, +)).
data(mat_dens(harv_fish_biomass, biomass, harv_fish, kg/ha)).
data(entity(harv_fish)).

data(abs_rate(harvest, biomass, fish, harv_fish, kg/ha/day)).
data(abs_rate(stocking, biomass, outside, fish, kg/ha/day)).

model_mat(biomass).


%%%  ``Processes that affect biomass dynamics of aquatic plants
%%%    include net primary production, natural mortality , and
%%%    consumption by fish.''

data(influences(plant_production, aquatic_plants_biomass, +)).
data(influences(plant_nat_mortality, aquatic_plants_biomass, -)).
data(influences(consumption, aquatic_plants_biomass, -)).


%%%  ``Processes that affect biomass dynamics of herbivorous fish
%%%    include consumption of plants, respiration, excretion, stocking,
%%%    and harvest.''

data(influences(consumption, fish_biomass, +)).
data(influences(respiration, fish_biomass, -)).
data(influences(excretion, fish_biomass, -)).
data(influences(stocking, fish_biomass, +)).
data(influences(harvest, fish_biomass, -)).


%%%  ``Net Production of plants is dependent on biomass of plants and
%%%    water temperature.''
```

```
data(abs_rate(plant_production, biomass, outside, aquatic_plants, kg/ha/day)).

data(influences(aquatic_plants_biomass, plant_production, ?)).
data(influences(water_temp, plant_production, ?)).


%%%  ``Natural mortality of plants is a function of plant biomass.''
%%%  ``It is known [from expert opinion] that the natural mortality
%%%    rate of plants [kg/ha/day] is density-dependent.''

data(abs_rate(plant_nat_mortality, biomass, aquatic_plants, outside, kg/ha/day)).
data(influences(aquatic_plants_biomass, plant_nat_mortality, ?)).


%%%  ``Fish are herbivorous and feed solely on aquatic plants that
%%%    grow naturally in the pond.''

data(abs_rate(consumption, biomass, aquatic_plants, fish, kg/ha/day)).


%%%  ``Consumption of plants by fish is a function of biomass of
%%%    plants, biomass of fish, and size of individual fish, as well
%%%    as being temperature-dependent.''

data(influences(fish_biomass, consumption, ?)).
data(influences(indiv_fish_weight, consumption, ?)).
data(influences(water_temp, consumption, ?)).


%%%  ``Fish respiration can be represented as a function of the weight
%%%  of fish and water temperature by [a] generally applicable
%%%  empirical relationship.''

data(abs_rate(respiration, biomass, fish, outside, kg/ha/day)).
data(influences(indiv_fish_weight, respiration, ?)).
data(influences(water_temp, respiration, ?)).


%%%  ``Fish excretion and respiration also are dependent on biomass
%%%    of fish, size of individual fish, and water temperature ...''
%%%  ``... with excretion also dependent on assimilation efficiency of
%%%    fish, which is a constant.''

data(abs_rate(excretion, biomass, fish, outside, kg/ha/day)).

data(influences(fish_biomass, excretion, ?)).
data(influences(indiv_fish_weight, excretion, ?)).
data(influences(water_temp, excretion, ?)).
data(influences(fish_ass_eff, excretion, ?)).
```

```
data(percentage(fish_ass_eff, fish, 1)).
data(constant(fish_ass_eff)).

data(influences(fish_biomass, respiration, ?)).


%%%  ``Stocking is calculated based on ... the number of fish
%%%    stocked, the initial weight of individual fish, which is a
%%%    constant, and the stocking date, which also is a constant.''

data(influences(number_fish_stocked, stocking, ?)).
data(influences(init_indiv_fish_weight, stocking, ?)).
data(weight_of(init_indiv_fish_weight, indiv_fish, kg)).
data(constant(init_indiv_fish_weight)).
data(influences(stocking_date, stocking, ?)).
data(event(stocking)).
data(when(stocking_date)).
data(time_of_event(stocking, stocking_date)).
data(constant(stocking_date)).


%%%  ``Harvest is calculated ... from herbivorous fish and harvest
%%%    date, which is a constant. Note that although we will run
%%%    simulations with different values for the stocking date,
%%%    harvest date, and number of fish stocked to address our
%%%    managemnet questions, during any given simulation these values
%%%    remain constant.''

data(influences(fish_biomass, harvest, ?)).
data(influences(harvest_date, harvest, ?)).
data(event(harvest)).
data(when(harvest_date)).
data(time_of_event(harvest, harvest_date)).
data(constant(harvest_date)).


%%%  ``The weight of indivifual fish is ... calculated based on the
%%%    number of fish stocked, which is a constant representing the
%%%    number of fish in the pond because there is no fish mortality,
%%%    and the biomass of herbivorous fish.''

data(influences(number_fish_stocked, indiv_fish_weight, ?)).
data(number_of(number_fish_stocked, fish_stocked, 1)).
data(entity(fish_stocked)).
data(constant(number_fish_stocked)).

data(influences(fish_biomass, indiv_fish_weight, ?)).

%%%-----------------------------------------------------------------
```

Table B.1: Data on historical states of the pond system including (a) the standing crop biomass of plants and fish, and the average weight of individual fish on various dates during several years, and (b) the water temperature at selected locations within the pond on various dates during a "normal" year (extract from (Grant et al., 1997)).

(a) Standing Crop Biomass of Plants and Fish, and Average Weight of Individual Fish

Date	Standing Crop Biomass (kg/ha) Plants	Fish	Weight of Individual Fish (kg)
	1975		
Apr. 15	39,973	17	0.23
May 15	39,273	18	0.24

(b) Water Temperature

Year	Date	Temperature (°C) at Location 1	2	3	4	5
1983	Jan. 1	13.9	15.2	16.0	15.2	14.7
	Jan. 15	15.6	14.4	15.1	15.0	14.8

Table B.2: Data on processes occurring within the pond system including (a) the net production rate of plants as a function of water temperature and (b) the rate of consumption of plants by fish as a function of water temperature and the weight of individual fish (extract from (Grant et al., 1997)).

(a) Net Production Rate of Plants (g Produced/kg Plant Biomass-Day)

Water Temperature (°C)	Net Production Rep 1	Rep 2	Rep 3	Rep 4	Rep 5
10	0.225	0.422	0.072	0.358	0.002
15	0.639	0.601	0.578	0.143	0.231

(b) Rate of Consumption of Plants by Fish of Different Sizes (kg Consumed/kg Fish Biomass-Day)

	Weight of Fish (kg)								
	0.227			1.500			3.636		
Water Temperature (°C)	Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3
15	0.001	0.340	0.387	0.043	0.056	0.243	0.376	0.083	0.086
20	0.739	0.534	1.026	0.360	0.166	0.605	0.132	0.278	0.254

Appendix C

Synthesis-$\mathcal{R}$'s Intermediate Algorithms

In this appendix we present the intermediate algorithms in the hierarchy of algorithms of the Synthesis-$\mathcal{R}$ system, as illustrated in Figure 8.5. The algorithms are referred to throughout Chapter 8.

The *Traverse Backbone and Match* Algorithm

Alg. C.1 traverses the reference model's backbone, matching its flows and state variables with the metadata set as it goes along to establish new pairs of matched flows.

The traversal takes off from the matched state variables established by the matching initialisation procedure. Each flow connected to the initial reference state variable is tried out for matching with the metadata set (Alg C.2). On success, a new pair of matched flows is established, as well as, in case of in-between-flows, the corresponding pair of newly matched state variables. A set of these new state variables pairs is formed. The reference state variables in the pairs are visited next, one at a time, their flows matched, state variables to be subsequently visited are established and so on.

The $Flows_{ref} \neq \{\}$ and $SVpairs = \{\}$ situation occurs when the reference state variables

Alg. C.1 $traverse\_bb\_n\_match(\Delta, SVpairs, Flows_{ref}, Mt, FPsf, FlowsPairs)$ — inputs: Δ, *SVpairs*, $Flows_{ref}$, *Mt*, *FPsf*; output: *FlowsPairs*. Starting from the pair(s) of matched state variables in *SVpairs*, traverse the reference model's backbone in $Flows_{ref}$, and match it with Δ to find new pairs of matching flows, which are added to the pair(s) found so far, *FPsf*, giving *FlowsPairs*.

IF $Flows_{ref} \neq \{\}$ **THEN**

  IF $SVpairs = \{\}$ **THEN**

    $\underline{match\_flows}(\Delta, Flows_{ref}, FPsf, FlowsPairs)$ (Alg. 8.2)

  ELSE $SVpairs = \{SVpair \mid SVps1\}$

    $\underline{survey\_SV\_n\_match}(\Delta, SVpair, Flows_{ref}, Fs_{ref}1, Mt, FPsf, FP1, SVps2)$ (Alg. C.2)

    $SVps3 = SVps1 \cup SVps2$

    $traverse\_bb\_n\_match(\Delta, SVps3, Fs_{ref}1, Mt, FP1, FlowsPairs)$

ELSE $FlowsPairs = FPsf$

reachable through the flows matched so far have all been visited, but there are still reference flows to be matched. In this case, Alg. 8.2 is called again to try and initialise a separate chain of flows and state variables in the new model, possibly representing flow of a different material. 'Separate' in the sense that it would not be connected to the previous one by flow arcs. Nevertheless, the separate chains may later become connected if Alg. 8.3, *Match Links*, succeeds in establishing link arcs between them.

The *Survey State Variable and Match* Algorithm

Alg. C.2 is executed to each pair of matched state variables as the reference model's backbone is traversed. Every flow connected to the current reference state variable is matched with the metadata set, according to the flow's form in relation to the state variable. Each successful new match updates the set of pairs of flows and also the set of pairs of state variables where appropriate. Flows which are not connected to the current reference state variable are returned, to be used in other runs of Alg. C.2 for forthcoming pairs of state variables.

In Alg. C.2's implementation we select, per pair of state variables, the best matches for

Alg. C.2 *survey_SV_n_match*$(\Delta, S_{ref}\text{-}S_{new}, Flows_{ref}, Fs_{ref}4, Mt, FPsf, FP1, SVps)$ — inputs: Δ, S_{ref}, S_{new}, $Flows_{ref}$, Mt, $FPsf$; outputs: $Fs_{ref}4$, $FP1$, $SVps$. Find the flows in $Flows_{ref}$ which are connected to the reference state variable S_{ref}, and match them with Δ to find: new pairs of matched flows, which are added to the pairs established so far, $FPsf$, giving $FP1$; and the corresponding new pairs of matched state variables, which form the set $SVps$. $Fs_{ref}4$ is the subset of $Flows_{ref}$ of flows not connected to S_{ref} to be tried out for other pairs of matched state variables.

Find $InFlows_{ref} = \{InF_{ref} : InF_{ref} \in Flows_{ref} \wedge InF_{ref}$ is an in-flow to $S_{ref}\}$

$Fs_{ref}1 = Flows_{ref} \backslash InFlows_{ref}$

FOR Each $InF_{ref} \in InFlows_{ref}$

IF $\underline{match\_in\_flow}(\Delta, InF_{ref}, Mt, S_{ref}\text{-}S_{new}, InF_{new})$ (Alg. 8.4) **AND**

There is no matched new flow in $FPsf$ with the same name as InF_{new} **THEN**

Add $InF_{ref}\text{-}InF_{new}$ to $FPsf$

Find $OutFlows_{ref} = \{OutF_{ref} : OutF_{ref} \in Fs_{ref}1 \wedge OutF_{ref}$ is an out-flow from $S_{ref}\}$

$Fs_{ref}2 = Fs_{ref}1 \backslash OutFlows_{ref}$

FOR Each $OutF_{ref} \in OutFlows_{ref}$

IF $\underline{match\_out\_flow}(\Delta, OutF_{ref}, Mt, S_{ref}\text{-}S_{new}, OutF_{new})$ (Alg 8.5) **AND**

There is no matched new flow in $FPsf$ with the same name as $OutF_{new}$ **THEN**

Add $OutF_{ref}\text{-}OutF_{new}$ to $FPsf$

Find $BtoFlows_{ref} = \{BtF_{ref} : BtF_{ref} \in Fs_{ref}2 \wedge BtF_{ref}$ is an in-between-flow entering $S_{ref}\}$

$Fs_{ref}3 = Fs_{ref}2 \backslash BtoFlows_{ref}$

FOR Each $BtF_{ref} \in BtoFlows_{ref}$

IF $\underline{match\_between_{in}\_flow}(\Delta, BtF_{ref}, Mt, S_{ref}\text{-}S_{new}, S1_{ref}\text{-}S1_{new}, BtF_{new})$ (Alg. 8.6) **AND**

There is no matched new flow in $FPsf$ with the same name as BtF_{new} **AND**

None of $S1_{ref}$ and $S1_{new}$ are matched with other state variables in $FPsf$ **THEN**

Add $BtF_{ref}\text{-}BtF_{new}$ to $FPsf$ and $S1_{ref}\text{-}S1_{new}$ to $SVps$

Find $BfrFlows_{ref} = \{BfF_{ref} : BfF_{ref} \in Fs_{ref}3 \wedge BfF_{ref}$ is an in-between-flow leaving $S_{ref}\}$

$Fs_{ref}4 = Fs_{ref}3 \backslash BfrFlows_{ref}$

FOR Each $BfF_{ref} \in BfrFlows_{ref}$

IF $\underline{match\_between_{out}\_flow}(\Delta, BfF_{ref}, Mt, S_{ref}\text{-}S_{new}, S2_{ref}\text{-}S2_{new}, BfF_{new})$ (Alg. 8.7) **AND**

There is no matched new flow in $FPsf$ with the same name as BfF_{new} **AND**

None of $S2_{ref}$ and $S2_{new}$ are matched with other state variables in $FPsf$ **THEN**

Add $BfF_{ref}\text{-}BfF_{new}$ to $FPsf$ and $S2_{ref}\text{-}S2_{new}$ to $SVps$

$FP1$ is the current $FPsf$

each of the four flow types (in-flow, out-flow, inward and outward in-between-flow), with the criterion being the greatest number of matches. Section 8.4 discusses what motivates such selection of local solutions as well as the effects of doing so.

The *Match Immediate Fringe* Algorithm

Alg. C.3 examines each pair of matched flows and appropriately starts up algorithms — Algs. C.5 and C.6 — which will find links connected to the reference flow and its one or two state variables and match them with the metadata. In examining each and every pair of matched flows in this way, the algorithm ultimately covers and matches the whole reference model's immediate fringe.

The *Match Far Fringe* Algorithm

While the *match immediate fringe* algorithm examines each pair of flows, Alg. C.4 below traverses and matches the far fringe by examining each pair of converters.

For each pair, Alg. C.6 is started up to firstly find links leaving the reference converter in the pair, as well as entering it if an intermediate variable. The links found are then matched with the metadata in order to synthesise links of the same form in relation to the new model's converter in the pair. A successful match may establish a new pair of converters (the converters at the other end of the matched links) to be examined subsequently. In this way, the reference model's far fringe is eventually fully traversed and the new model's far fringe expanded where matching metadata hold. The reuse-synthesis process ends here, when no further traversal and expansion are possible.

Alg. C.3 $match\_immediate\_fringe(\Delta, FlowsPairs, Links_{ref}, Links'_{ref}, Msf, M, CvsPairs, CvsPairs')$ — inputs: Δ, $FlowsPairs$, $Links_{ref}$, Msf, $CvsPairs$; outputs: $Links'_{ref}$, M, $CvsPairs'$. Find the links in $Links_{ref}$ composing the reference model's immediate fringe and match them with Δ according to the form of each reference link, with a view to synthesising the new model's immediate fringe, expanding the new model so far Msf into M, and giving a set of pairs of matched converters $CvsPairs'$. Links outside of the reference model's immediate fringe are returned in the set $Links'_{ref}$.

IF $FlowsPairs = \{\}$ **THEN** $Links'_{ref} = Links_{ref} \quad M = Msf \quad CvsPairs' = CvsPairs$

ELSE $FlowsPairs = \{Fpair \mid FP1\}$

IF The flows in *Fpair* have initial state variables, matched in *SViPair*, which have not yet been reached **THEN**

$\underline{find\_SVFlLk\_n\_match}(\Delta, SVi_{ref}, Fpair, Links_{ref}, Ls_{ref}1, Msf, M1)$ (Alg. C.5), find in $Links_{ref}$ and match with Δ a link of form (l1) of Fig. 8.4, where the flow is the reference flow in the current *Fpair*, giving the updated sets $Ls_{ref}1$ and $M1$.

$\underline{find\_ElLks\_n\_match}(\Delta, sv_c \rightarrow iv, SViPair, Ls_{ref}1, Ls_{ref}2, M1, M2, CvsPairs, CP1)$ (Alg. C.6), find in $Ls_{ref}1$ and match with Δ links of form (l4) of Fig. 8.4, where the state variable is the initial state variable of the reference flow in the current *Fpair*, giving the updated sets $Ls_{ref}2$, $M2$ and $CP1$.

ELSE $Ls_{ref}2 = Links_{ref} \quad M2 = Msf \quad CP1 = CvsPairs$

IF The flows in *Fpair* have terminal state variables, matched in *SVtPair*, which have not yet been reached **THEN**

$\underline{find\_SVFlLk\_n\_match}(\Delta, SVt_{ref}, Fpair, Ls_{ref}2, Ls_{ref}3, M2, M3)$ (Alg. C.5), find in $Ls_{ref}2$ and match with Δ a link of form (l2) of Fig. 8.4, where the flow is the reference flow in the current *Fpair*, giving the updated sets $Ls_{ref}3$ and $M3$.

$\underline{find\_ElLks\_n\_match}(\Delta, sv_c \rightarrow iv, SVtPair, Ls_{ref}3, Ls_{ref}4, M3, M4, CP1, CP2)$ (Alg. C.6), find in $Ls_{ref}3$ and match with Δ links of form (l4) of Fig. 8.4, where the state variable is the terminal state variable of the reference flow in the current *Fpair*, giving the updated sets $Ls_{ref}4$, $M4$ and $CP2$.

ELSE $Ls_{ref}4 = Ls_{ref}2 \quad M4 = M2 \quad CP2 = CP1$

$\underline{find\_ElLks\_n\_match}(\Delta, cv \rightarrow flow_c, Fpair, Ls_{ref}4, Ls_{ref}5, M4, M5, CP2, CP3)$ (Alg. C.6), find in $Ls_{ref}4$ and match with Δ links of form (l3) of Fig. 8.4, where the flow is the reference flow in the current *Fpair*, giving the updated sets $Ls_{ref}5$, $M5$ and $CP3$.

$match\_immediate\_fringe(\Delta, FP1, Ls_{ref}5, Links'_{ref}, M5, M, CP3, CvsPairs')$

Alg. C.4 $match\_far\_fringe(\Delta, CvsPairs', CPlog, CPtoExpd, Links'_{ref}, M, Model)$ — inputs: Δ, $CvsPairs'$, $CPlog$, $CPtoExpd$, $Links'_{ref}$, M; output: $Model$. Find the links in $Links'_{ref}$ composing the reference model's far fringe and match them with Δ according to the form of each reference link, with a view to synthesising an outer network of links, which is added to the new model so far M to give a final model $Model$. The synthesis process ends when all the reachable converters have been expanded.

IF $CPtoExpd = \{\}$, the current set of converters pairs to expand became empty **THEN**

> **IF** $CPlog = CvsPairs'$, new pairs of converters have not been established **THEN** $Model = M$
>
> **ELSE** $CPnew = CvsPairs' \setminus CPlog$, find the pairs of converters which are new.
>
> > $match\_far\_fringe(\Delta, CvsPairs', CvsPairs', CPnew, Links'_{ref}, M, Model)$

ELSE $CPtoExpd = \{CPair \mid CPrest\}$

> $\underline{find\_ElLks\_n\_match}(\Delta, cv_c \rightarrow iv, CPair, Links'_{ref}, Ls_{ref}6, M, M6, CvsPairs', CP4)$ (Alg. C.6), find in $Links'_{ref}$ and match with Δ links of form (l5) of Fig. 8.4, where the converter is the reference converter in the current *Cpair*, giving the updated sets $Ls_{ref}6$, $M6$ and $CP4$.
>
> **IF** The converters in *Cpair* are of type intermediate variable **THEN**
>
> > $\underline{find\_ElLks\_n\_match}(\Delta, cv \rightarrow iv_c, CPair, Ls_{ref}6, Ls_{ref}7, M6, M7, CP4, CP5)$ (Alg. C.6), find in $Ls_{ref}6$ and match with Δ links of form (l5) of Fig. 8.4, where the intermediate variable is the reference intermediate variable in the current *Cpair*, giving the updated sets $Ls_{ref}7$, $M7$ and $CP5$.
>
> **ELSE** $Ls_{ref}7 = Ls_{ref}6 \quad M7 = M6 \quad CP5 = CP4$
>
> $match\_far\_fringe(\Delta, CP5, CPlog, CPrest, Ls_{ref}7, M7, Model)$

The *Find State Variable-to-Flow Link and Match* Algorithm

Alg. C.5 establishes links between state variables and flows connected to them (link forms (l1) and (l2) of Figure 8.4).

There can only be a single link between a flow and each of its state variables. The link is sought for and if found matched with the new model's metadata set. At the point of the synthesis process where this algorithm is executed, both nodes (state variable and flow) the new link is to connect are already established and are given in F_{new} — the flow's specification includes its state variable(s) (Section 6.3.3.1). Therefore, the matching consists simply of finding evidence in the metadata set that supports a link between the established new flow and its state variable.

Alg. C.5 $find\_SVFlLk\_n\_match(\Delta, SV_{ref}, F_{ref}\text{-}F_{new}, Ls_{ref}, Ls_{ref}1, M, M1)$ — inputs: Δ, SV_{ref}, F_{ref}, F_{new}, Ls_{ref}, M; outputs: $Ls_{ref}1$, $M1$. Given the pair of matched flows F_{ref}-F_{new}, find in the set of reference links Ls_{ref} and match with Δ a link of form (l1) or (l2) of Fig. 8.4, where the flow is F_{ref} and the state variable is SV_{ref}; L_{ref} is removed from Ls_{ref} giving the updated set of reference links $Ls_{ref}1$, and the matching new link is added to M giving the new model so far $M1$.

IF $\exists L_{ref} \,.\, L_{ref} \in Ls_{ref} \,\wedge\, L_{ref}$ is a link from SV_{ref} to its flow F_{ref} **THEN**

$Ls_{ref} = \{L_{ref} \mid Ls_{ref}1\}$

IF $\underline{establish\_SVFlLk}(\Delta, SV_{new}, Fid_{new}, L_{new})$ (Alg. 8.8), a link L_{new} is established from the matched new state variable SV_{new} to its flow named Fid_{new} **THEN**

$M1 = \{L_{new} \mid M\}$

ELSE $M1 = M$

ELSE $Ls_{ref}1 = Ls_{ref} \quad M1 = M$

The *Find Element's Links and Match* Algorithm

Alg. C.6 finds links from and to the current reference model element (link forms (l3), (l4) and (l5) of Figure 8.4), according to the link form given, and proceeds to match the links found with the metadata set.

Alg. C.6 $find\_ElLks\_n\_match(\Delta, Lform, El_{ref}\text{-}El_{new}, Ls_{ref}, Ls_{ref}1, M, M1, CP, CP1)$ — inputs: Δ, $Lform$, El_{ref}, El_{new}, Ls_{ref}, M, CP; outputs: $Ls_{ref}1$, $M1$, $CP1$. Find in the set of reference links Ls_{ref}, the subset of links of form $Lform$ connected to the reference element El_{ref}, leaving the complement subset $Ls_{ref}1$. Match the links found with the metadata set Δ to give new links to update the new model so far giving $M1$ as well as the set of pairs of matched converters giving $CP1$.

CASE $Lform \in \{sv_c \rightarrow iv, cv_c \rightarrow iv\}$ **THEN**

Find $LsEl_{ref} \subset Ls_{ref}$, the subset of all links from the current matched reference element El_{ref} (a state variable or a converter) into intermediate variables.

$Lform \in \{cv \rightarrow flow_c, cv \rightarrow iv_c\}$ **THEN**

Find $LsEl_{ref} \subset Ls_{ref}$, the subset of all links from converters to the current matched reference element El_{ref} (a flow or an intermediate variable).

IF $LsEl_{ref} \neq \{\}$ **THEN**

<u>match_ElLinks</u>$(\Delta, Lform, LsEl_{ref}, El_{new}, M, M1, CP, CP1)$ (Alg. C.7)

$Ls_{ref}1 = Ls_{ref} \backslash LsEl_{ref}$

ELSE $Ls_{ref}1 = Ls_{ref}$ $M1 = M$ $CP1 = CP$

The *Match Element's Links* Algorithm

Alg. C.7 does the matching of all links connected to the current reference model element, calling the appropriate procedure according to the form of each link, and upon successful matches, updates the new model and the set of pairs of matched converters. For updates to occur, the matching links must be new to the current new model, as well as non-conflicting with previously established components in it. There are two situations that allow such updates:

- The link is new to the new model, but the converters pair is not. This means that the converter linked to the current new element has already been matched and established in the new model. Of course, the only update needed is the addition of the link to the new model.

- The link is new to the model and the converters pair is new and non-conflicting, i.e., the reference converter is unmatched and the new converter introduces a new quantity into the model.

Alg. C.7 $match\_ElLinks(\Delta, Lform, LsEl_{ref}, El_{new}, M, M2, CP, CP2)$ — inputs: Δ, *Lform*, $LsEl_{ref}$, El_{new}, *M*, *CP*; outputs: *M2*, *CP2*. Match with Δ the links $LsEl_{ref}$ of form *Lform* connected to the reference element matched with the new element El_{new}, to give new links which are added to the new model so far *M* giving *M2*, and new pairs of matched converters which are added to their set so far *CP* giving *CP2*.

IF $LsEl_{ref} = \{\}$ **THEN** $M2 = M \quad CP2 = CP$

ELSE $LsEl_{ref} = \{L_{ref} \mid LsEl_{ref}1\}$

IF $\left(\begin{array}{l} Lform = sv_c \rightarrow iv \quad \textbf{AND} \\ \underline{match\_frSVLk}(\Delta, El_{new}, L_{ref}, L_{new}, CV_{ref}\text{-}CV_{new}) \text{ (Alg. 8.10)} \end{array}\right)$ **OR**

$\left(\begin{array}{l} Lform = cv \rightarrow flow_c \quad \textbf{AND} \\ \underline{match\_toFlLk}(\Delta, El_{new}, L_{ref}, L_{new}, CV_{ref}\text{-}CV_{new}) \text{ (Alg. 8.9)} \end{array}\right)$ **OR**

$\left(\begin{array}{l} Lform = cv_c \rightarrow iv \quad \textbf{AND} \\ \underline{match\_frCVLk}(\Delta, El_{new}, L_{ref}, L_{new}, CV_{ref}\text{-}CV_{new}) \text{ (Alg. 8.11)} \end{array}\right)$ **OR**

$\left(\begin{array}{l} Lform = cv \rightarrow iv_c \quad \textbf{AND} \\ \underline{match\_toIVLk}(\Delta, El_{new}, L_{ref}, L_{new}, CV_{ref}\text{-}CV_{new}) \text{ (Alg. 8.12)} \end{array}\right)$ **THEN**

IF $L_{new} \notin M$ **AND** $CV_{ref}\text{-}CV_{new} \in CP$ **THEN**

$M1 = \{L_{new} \mid M\}$

$match\_ElLinks(\Delta, Lform, LsEl_{ref}1, El_{new}, M1, M2, CP, CP2)$

ELSE

IF CV_{ref} is not matched (it is not the ref. converter of one of the pairs in *CP*) **AND** The quantity CV_{new} holds is not held by any other model element in *M* **THEN**

$M1 = \{L_{new} \mid M\}$

$CP1 = \{CV_{ref}\text{-}CV_{new} \mid CP\}$

$match\_ElLinks(\Delta, Lform, LsEl_{ref}1, El_{new}, M1, M2, CP1, CP2)$

ELSE $match\_ElLinks(\Delta, Lform, LsEl_{ref}1, El_{new}, M, M2, CP, CP2)$

ELSE $match\_ElLinks(\Delta, Lform, LsEl_{ref}1, El_{new}, M, M2, CP, CP2)$

Appendix D

Input and Output Settings for Artificial Metadata Generation

D.1 Input Settings

The settings for describing the sample models, shown in Table D.1, comprise identifiers and units of measure (where appropriate) for each type of model component. The identifiers index *i* corresponds to a counter for components of that type in the model. The two possibilities of unit of measure for parameters are intended simply to introduce an element of variability into the metadata sets generated.

D.2 Output Settings

As we saw in Chapter 5, a metadata set includes descriptions of model requirements and of ecological data. Given a sample model described according to the input settings in Table D.1, the metadata generation program outputs the model requirements descriptions as in Table D.2, and the data descriptions for each type of model component as in Table D.3.

Model component type	Id	Unit of measure
flow	f_i	$kg/year$
state variable	a_i	kg
intermediate variable	i_i	1 (the identity unit)
parameter	p_i	$kg/kg/year$, if connected to a flow; 1 (the identity unit), otherwise
driving variable	d_i	day
source/sink	$outside$	

Table D.1: Input settings for generation of artificial metadata.

Model requirements
– $model\_att(m)$
– $model\_time\_unit(year)$
– $model\_goal\_var(amt\_of\_mat(A,m,E,Ua))$, for each $sv(A,Ua)$ model component

Table D.2: Output settings for generation of artificial metadata: model requirements descriptions.

Model component	Data descriptions
$flow(F, outside, sv(A, Ua), Ua/Ut)$	– $data(abs\_rate(F, m, outside, E, Ua/Ut))$
	– $\langle sv(A, Ua)$ data description$\rangle$
$flow(F, sv(A, Ua), outside, Ua/Ut)$	– $data(abs\_rate(F, m, E, outside, Ua/Ut))$
	– $\langle sv(A, Ua)$ data description$\rangle$
$flow(F, sv(A_1, Ua), sv(A_2, Ua), Ua/Ut)$	– $data(abs\_rate(F, m, E_1, E_2, Ua/Ut))$
	– $\langle sv(A_1, Ua)$ data description$\rangle$
	– $\langle sv(A_2, Ua)$ data description$\rangle$
$link(sv(A, Ua), flow_{id}(F))$	– $data(influences(A, F, ?))$
$link(iv(I, Ui), flow_{id}(F))$	– $data(influences(I, F, ?))$
	– $\langle iv(I, Ui)$ data description$\rangle$
$link(param(P, Up), flow_{id}(F))$	– $\langle$flow-connected $param(P, Up)$ data description$\rangle$
$link(dv(D, Ud), flow_{id}(F))$	– $data(influences(D, F, ?))$
	– $\langle dv(D, Ud)$ data description$\rangle$
$link(sv(A, Ua), iv(I, Ui))$	– $data(influences(A, I, ?))$
	– $\langle iv(I, Ui)$ data description$\rangle$
$link(iv(I_1, Ui_1), iv(I_2, Ui_2))$	– $data(influences(I_1, I_2, ?))$
	– $\langle iv(I_1, Ui_1)$ data description$\rangle$
	– $\langle iv(I_2, Ui_2)$ data description$\rangle$
$link(dv(D, Ud), iv(I, Ui))$	– $data(influences(D, I, ?))$
	– $\langle dv(D, Ud)$ data description$\rangle$
	– $\langle iv(I, Ui)$ data description$\rangle$
$link(param(P, Up), iv(I, Ui))$	– $data(influences(P, I, ?))$
	– $\langle param(P, Up)$ data description$\rangle$
	– $\langle iv(I, Ui)$ data description$\rangle$
$iv(I, Ui)$	– $data(proportion(I, Ui))$
$param(P, Up)$	– $data(spf\_rate(P, F, m, Up))$, if connected to a flow;
	$data(number\_of(P, Up))$, otherwise
	– $data(constant(P))$
$dv(D, Ud)$	– $data(time(D, Ud))$

Table D.3: Output settings for generation of artificial metadata: data descriptions.

Bibliography

Abel, D. E. and Niven, B. S. (1990). Application of a formal specification language to animal ecology. *Ecological Modelling*, 50:205–212.

Apt, K. R. (1997). *From logic programming to Prolog*. Prenctice Hall.

Arpírez, J. C., Gómez-Pérez, A., Lozano-Tello, A., and Pinto, H. S. (2000). Reference ontology and $(\mathrm{ONTO})^2$ agent: the ontology yellow pages. *Knowledge and Information Systems*, 2(4):387–412.

Baker, P., Goble, C., Bechhofer, S., Paton, N., Stevens, R., and Brass, A. (1999). An ontology for bioinformatics applications. *Bioinformatics*, 15(6):510–520.

Bayardo, R. J., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., Rashid, M., Rusinkiewicz, M., Shea, R., Unnikrishnan, C., Unruh, A., and Woelk, D. (1997). InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 195–206, New York. ACM Press. Also in M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, Morgan Kaufman, 1997.

Benjamins, R. and Fensel, D. (1998). The ontological engineering initiative $(\mathrm{KA})^2$. In Guarino, N., editor, *Proceedings of the First International Conference on Formal Ontology in Information Systems - FOIS'98*, pages 287–301, Trento, Italy. IOS Press.

Berlin, B. (1978). *Cognition and Categorization*, chapter Ethnobiological Classification, pages 9–26. Lawrence Erlbaum, Hillsdale, N.J. Edited by E. Rosch and B. Loyd.

Biot, Y. (1995). Data survey report - BIONTE Project. Technical report, INPA - Brazil and ODA - UK, Manaus, Brazil.

Biot, Y., Higuchi, N., Brilhante, V., de Freitas, J., Ferraz, J., Leal, N., Ferreira, S., and Desjardins, T. (1996). INFORM: the INpa FORest Model. Technical report, Project BIONTE, INPA - Brazil and ODA - UK, Manaus, Brazil.

Borst, P., Akkermans, H., and Top, J. (1997). Engineering ontologies. *International Journal of Human-Computer Studies*, 46:365–406.

Borst, P., Benjamin, J., Wielinga, B., and Akkermans, H. (1996). An application of ontology construction. In *Proceedings of the ECAI'96 Workshop on Ontological Engineering*, pages 5–16, Budapest.

Brilhante, V. (1996). Inform-logic: a system for representing uncertainty in ecological models. M.Sc. thesis MT9605, Department of Artificial Intelligence, University of Edinburgh.

Brilhante, V. (1999). Using formal meta-data descriptions for automated ecological modeling. Papers from the AAAI-99 Workshop on Environmental Decision Support Systems and Artificial Intelligence WS-99-07, Orlando, Florida. AAAI Press.

Brilhante, V. and Robertson, D. (2001). *Environmental Information Systems in Industry and Public Administration*, chapter Metadata-supported Automated Ecological Modelling. Idea Group Publishing, Hershey, PA. Edited by Claus Rautenstrauch and Susanne Patig.

Bundy, A. (2000). Criteria for assessing AI research. Lecture Notes of the Artificial Intelligence Research Methodologies module, Division of Informatics, University of Edinburgh.

Bundy, A. and Uschold, M. (1989). The use of typed lambada calculus for requirements capture in the domain of ecological modelling. Technical Report DAI Research Paper 446, Department of Artificial Intelligence, University of Edinburgh.

Castro, A. (1999). *A Techniques-based Framework for Domain-Specific Synthesis of Simulation Models*. PhD thesis, University of Edinburgh.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., and Rice, J. (1998a). OKBC: A programmatic foundation for knowledge base interoperability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence - AAAI-98*, pages 600–607. AAAI Press.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., and Rice, J. P. (1998b). Open knowledge base connectivity 2.0. Technical Report KS-98-06, Knowledge Systems Laboratory, Stanford University, Stanford, California.

Chikofsky, E. J. and Cross II, J. H. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17.

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3(4):261–283.

Clark, P. and Porter, B. (1997). Building concept representations from reusable components. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence - AAAI-97*, pages 369–376, Providence, RI. AAAI Press.

Cohen, P. (1985). *Heuristic Reasoning About Uncertainty: an Artificial Intelligence Approach*. Pitman, London.

Cohen, P., Chaudhri, V., Pease, A., and Schrag, R. (1999). Does prior knowledge facilitate the development of knowledge-based systems? In *Proceedings of the Sixteenth International Conference on Artificial Intelligence - AAAI-99*, pages 221–226. AAAI Press.

Cohen, P., Schrag, R., Jones, E., Pease, A., Lin, A., Starr, B., Gunning, D., and Burke, M. (1998). The DARPA high performance knowledge bases project. *AI Magazine*, 19(9):25–49.

Cyc (1995). Cyc Knowledge Base. http://www.cyc.com. Cycorp.

da Silva, F. S. C., Vasconcelos, W. W., Robertson, D. S., Brilhante, V., de Melo, A. C. V., Finger, M., and Agustí, J. (2002). On the insufficiency of ontologies: Problems in knowledge sharing and alternative solutions. *Knowledge-Based Systems*, 15(3):147–167.

DAML (2002). DAML Ontology Library. http://daml.org/ontologies. The DARPA Agent Markup Language Program.

Davis, E. (1990). *Representations of Commonsense Knowledge*. Morgan Kaufmann Publishers, Inc.

Džeroski, S., Grbović, J., Walley, W. J., and Kompari, B. (1997). Using machine learning techniques in the construction of models. II. Data analysis with rule induction. *Ecological Modelling*, 95:95–111.

Džeroski, S. and Todorovski, L. (1993). Discovering dynamics. In Utgoff, P., editor, *Proceedings of the Tenth International Conference on Machine Learning*, pages 97–103, San Mateo, CA. Morgan Kaufmann.

Džeroski, S. and Todorovski, L. (1995). Discovering dynamics: from inductive logic programming to machine discovery. *Journal of Intelligent Information Systems*, (4):89–108.

Džeroski, S., Todorovski, L., Bratko, I., Kompare, B., and Križman, V. (1999). *Machine Learning Methods for Ecological Applications*, chapter Equation Discovery with Ecological Applications, pages 185–207. Kluwer Academic Publishers, Boston. Edited by Alan Fielding.

Ellis, B. (1966). *Basic Concepts of Measurement*. Cambridge University Press, London.

Falkenhainer, B. and Forbus, K. (1991). Compositional modelling. *Artificial Intelligence*, 51:95–143.

Farquhar, A. (1995). Ontolingua to prolog syntax translation. http://www.ksl.Stanford.EDU/people/axf/ol-to-prolog.txt.

Farquhar, A., Fikes, R., and Rice, J. (1996). The ontolingua server: a tool for collaborative ontology construction. Technical Report KSL-96-26, Computer Science Department, Stanford University.

Ferguson, G. and Allen, J. F. (1994). Arguing about plans: Plan representation and reasoning for mixed-initiative planning. In *Proceedings of the Second International Conference on AI Planning Systems - AIPS-94*, pages 43–48, Chicago.

Fernández, M., Gómez-Pérez, A., and Juristo, N. (1997). METHONTOLOGY. In *AAAI-97, Spring Symposium Series, Ontological Engineering*, pages 33–40.

Fernández, M., Gómez-Pérez, A., Sierra, A. P., and Sierra, J. P. (1999). Building a chemical ontology using METHONTOLOGY and the Ontology Design Environment. *IEEE Intelligent Systems*, 14(1):37–46.

Flach, P. (1994). *Simply Logical*. John Wiley & Sons.

Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, 24:85–168.

Ford, A. (1999). *Modeling the Environment*. Island Press.

Forrester, J. W. (1961). *Industrial Dynamics*. The M.I.T. Press, Massachusetts.

Genesereth, M. R. and Fikes, R. E. (1992). Knowledge interchange format, version 3.0, reference manual. Technical Report Logic-92-1, Logic Group, Computer Science Department, Stanford University, Stanford, California.

Gómez-Pérez, A. and Rojas-Amaya, M. D. (1999). Ontological reengineering for reuse. In Fensel, D. and Studer, R., editors, *Knowledge Acquisition, Modeling and Management*, number 1621 in Lecture Notes in Artificial Intelligence, pages 139–156. Springer, Berlin; New York. Proceedings of the Eleventh European Knowledge Acquisition Workshop - EKAW-99, Dagstuhl Castle, Germany.

Grant, W. E., Pedersen, E. K., and Marín, S. L. (1997). *Ecology and Natural Resource Management*. John Wiley & Sons, Inc.

Grosso, W. E., Gennari, J. H., Fergerson, R. W., and Musen, M. A. (1998). When knowledge models collide (how it happens and what to do). In *Proceedings of the Eleventh Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Canada.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5/6):907–928.

Gruber, T. R. and Olsen, G. R. (1994). An ontology for engineering mathematics. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, Germany. Morgan Kaufmann.

Haefner, J. (1996). *Modeling Biological Systems*. Chapman & Hall.

Hafner, C. and Fridman, N. (1996). Ontological foundations for biology knowledge models. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology - ISMB-96*, pages 78–87, St. Louis, MO. AAAI Press.

Hilborn, R. and Mangel, M. (1997). *The Ecological Detective*. Princeton University Press.

Kakas, A. C., Kowalski, R. A., and Toni, F. (1998). The role of abduction in logic programming. In D.M. Gabbay, C. H. and Robinson, J., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming 5*, pages 235–324. Oxford University Press.

Kalfoglou, Y. and Schorlemmer, M. (2002). Information-flow-based ontology mapping. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, number 2519 in Lecture Notes in Computer Science, pages 1132–1151. Springer.

Karalič, A. (1992). *RETIS: A Knowledge Acquisition System*. Ljubljana. User's manual. Software version: 2.07/nf.

Karp, P., Riley, M., Paley, S., and Pellegrini-Toole, A. (1996). Ecocyc: Electronic encyclopedia of e. coli genes and metabolism. *Nucleic Acids Research*, 24(1):32–40.

Kashyap, V. (1999). Design and creation of ontologies for environmental information retrieval. In *Proceedings of the Twelfth International Conference on Knowledge Acquisition, Modeling and Management*, Banff, Canada.

Keppens, J. (2002). *Compositional Ecological Modelling via Dynamic Constraint Satisfaction with Order-of-Magnitude Preferences*. PhD thesis, University of Edinburgh.

Keppens, J. and Shen, Q. (2001). On compositional modelling. *Knowledge Engineering Review*, 16(2):157–200.

Knight, K. and Luk, S. (1994). Building a large knowledge base for machine translation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence - AAAI-94*, Seattle, WA. AAAI Press.

Kompare, B., Bratko, I., Steinman, F., and Džeroski, S. (1994). Using machine learning techniques in the construction of models. I. Introduction. *Ecological Modelling*, 75/76:617–628.

Križman, V., Džeroski, S., and Kompare, B. (1995). Discovering dynamics from measured data. *Electrotechnical Review*, (62):191–198.

Kubat, M., Bratko, I., and Michalski, R. (1997). *Machine Learning and Data Mining: Methods and Applications*, chapter A review of Machine Learning Methods, pages 3–69. John Wiley & Sons, New York. Edited by R. Michalski, I. Bratko and M. Kubat.

Kuipers, B. (1994). *Qualitative Reasoning*. MIT Press, Cambridge, MA.

Lenat, D. B. and Guha, R. V. (1990). *Building Large Knowledge Based Systems*. Addison Wesley, Reading, Massachusetts.

Lloyd, J. W. (1993). *Foundations of Logic Programming*. Springer-Verlag, second extended edition.

Lorenz, G., Uhrmacher, A., Simon, K.-H., and Bossel, H. (1989). Application of artificial intelligence methods to the representation and modelling of tree growth. In *Proceedings of the IUFRO Conference, Artificial Intelligence and Growth Models for Forest Management*, Vienna, Austria.

Luger, G. F. and Stubblefield, W. A. (1993). *Artificial Intelligence*. The Benjamin/Cummings Publishing Company, Inc., California, second edition.

MacGregor, R. (1991). Inside the LOOM classifier. *Sigart Bulletin*, 2(3):70–76.

Massey, B. S. (1986). *Measures in Science and Engineering*. Ellis Horwood Limited.

Meadows, D., Meadows, D., Randers, J., and Behrens, W. (1972). *The Limits to Growth*. Universe Books.

Miguel, I. (2001). *Dynamic Flexible Constraint Satisfaction and Its Application to AI Planning*. PhD thesis, University of Edinburgh.

Mittal, S. and Falkenhainer, B. (1990). Dynamic constraint satisfaction problems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 25–32.

Muetzelfeldt, R. I., Robertson, D., Bundy, A., and Uschold, M. (1989). The use of prolog for improving the rigour and accessibility of ecological modelling. *Ecological Modelling*, 46:9–34.

Neches, R., Fikes, R. E., Finin, T., Gruber, T. R., Senator, T., and Swartout, W. R. (1991). Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56.

Niven, B. S. (1982). Formalization of the basic concepts of animal ecology. *Erkenntnis*, 17:307–320.

Niven, B. S. (1992). Formalization of some concepts of plant ecology. *Coenoses*, 7(2):103–113.

Noy, N. and Hafner, C. (2000). Ontological foundations for experimental science knowledge bases. *Applied Artificial Intelligence*, 14(6):565–618.

Noy, N. F. (1997). *Knowledge Representation for Intelligent Information Retrieval in Experimental Sciences*. PhD thesis, Northeastern University, Boston.

O'Keefe, R. A. (1990). *The craft of Prolog*. The MIT Press.

Ontolingua Server (1995). Ontolingua Server. http://ontolingua.stanford.edu. Knowledge Systems Laboratory, Department of Computer Science, Stanford University.

Pinto, H. S. (1999). Towards ontology reuse. Papers from the AAAI-99 Workshop on Ontology Management WS-99-13, Orlando, Florida. AAAI Press.

Pinto, H. S. and Martins, J. P. (2000). Reusing ontologies. Proceedings of the AAAI-00 Spring Symposium Series, Workshop on Bringing Knowledge to Business Processes SS-00-03. AAAI Press.

Pinto, H. S. and Martins, J. P. (2001). A methodology for ontology integration. In *Proceedings of the First International Conference on Knowledge Capture - K-CAP 2001*, Victoria, B.C., Canada. ACM Press.

Protégé (2000). Protégé Ontologies Library. http://protege.stanford.edu/ontologies.html. Stanford Medical Informatics, Stanford University.

Rickel, J. and Porter, B. (1997). Automated modeling of complex systems to answer prediction questions. *Artificial Intelligence*, 93(1-2):201–260.

Rickel, J. W. (1995). *Automated Modelling of Complex Systems to Answer Prediction Questions*. PhD thesis, The University of Texas at Austin.

Robertson, D. and Agustí, J. (1999). *Software Blueprints*. Addison Wesley, ACM Press.

Robertson, D., Bundy, A., Muetzelfeldt, R., Haggith, M., and Uschold, M. (1991). *Eco-Logic: logic-based approaches to ecological modelling*. The MIT Press.

Robertson, D., Haggith, M., Kendon, G., Agusti, J., and Goldsborough, D. (1995). Application of logic programming to decision support systems in ecology. *AI Applications*, 9(3):23–38.

Sowa, J. F. (2000). *Knowledge Representation*. Brooks/Cole.

Spivey, J. M. (1988). *Understanding z*. Cambridge University Press, Cambridge.

Sterling, L. and Shapiro, E. (1994). *The art of Prolog*. The MIT Press, second edition.

Swartout, B., Patil, R., Knight, K., and Russ, T. (1996). Toward distributed use of large-scale ontologies. In *Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop - KAW-96*, Banff, Canada.

Todorovski, L. and Džeroski, S. (1997). Declarative bias in equation discovery. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 376–384, San Mateo, CA. Morgan Kaufmann.

Uhrmacher, A. (1995). Reasoning about changing structure: a modeling concept for ecological systems. *Applied Artificial Intelligence*, 9:157–180.

Uschold, M. (1990). *The use of typed lambda calculus for comprehension and construction of simulation models in the domain of ecology*. PhD thesis, University of Edinburgh.

Uschold, M. (1991). The use of domain information for comprehension and construction of simulation models. Technical Report DAI Research Paper 534, Department of Artificial Intelligence, University of Edinburgh.

Uschold, M. (1998). Where are the killer apps? In *Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods - ECAI-98*, Brighton, UK.

Uschold, M. and Gruninger, M. (1996). Ontologies: principles, methods and applications. *The Knowledge Engineering Review*, 11(2):93–136.

Uschold, M., Harding, N., Muetzelfeldt, R., and Bundy, A. (1984). An intelligent front end for ecological modeling. Technical Report DAI Research Paper 223, Department of Artificial Intelligence, University of Edinburgh.

Uschold, M., Healy, M., Williamson, K., Clark, P., and Woods, S. (1998). Ontology reuse and application. In *Proceedings of the First International Conference on Formal Ontology in Information Systems - FOIS-98*, pages 179–192, Trento, Italy. IOS Press.

Uschold, M., Jasper, R., and Clark, P. (1999). Three approaches for knowledge sharing: a comparative analysis. In *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Canada.

Valente, A., Russ, T., MacGregor, R., and Swartout, W. (1999). Building and (re)using an ontology of air campaign planning. *IEEE Intelligent Systems*, 14(1):27–36.

Xia, S. and Smith, N. (1996). Automated modelling. *The Knowledge Engineering Review*, 11(2):137–160.