From Domains to Requirements Lecture Notes in Software Engineering Budapest, 11–22 October 2010 1

Dines Bjørner
Fredsvej 11, DK-2840 Holte, Denmark
bjorner@gmail.com – www.imm.dtu.dk/~db

Begun: Tuesday June 22, 2010. Compiled: November 1, 2010: 15:25

Abstract 2

We present "standard" domain description and requirements prescription examples using the RAISE [112] Specification Language, RSL [110]. The illustrated example is that of transportation networks.

These notes shalll serve as lecture notes for my lectures at Uppsala, Nov.8-19, 2010. The present document is the ordinary "book-form"-like notes. A separate document, compiled from the same files, present 11 sets of lecture slides. The "funny" small numbers you see in the present document, in margins and at almost end of display lines refer to slide page numbers of the slides document.

Lecture Notes 3

A Tentative Lecture Schedule

Lecture 1: Introduction Mo.8.11.2010 8–12

Lecture 2: Specification Ontology Mo.8.11.2010 13–25

Entities: Simple Entities, Actions, Events, Behaviours

Lecture 3: Domain Facets I Tu.9.11.2010 26–40

Intrinsics, Support Technologies, Rules & Regulations

Lecture 4: Domain Facets II We.10.11.2010 40–56

Scripts, Management & Organisation, Human Behaviour

Lecture 5: Requirements Facets I Th.11.11.2010 57–65

Domain Requirements I: Projection, Instantiation, Determiniation

Lecture 6: Requirements Facets II Fr.12.11.2010 65–73

Domain Requirements II: Extension, Fitting
Interface Requirements
4 Machine Requirements

Lecture 7: RSL I Mo.15.11.2010 89–97

Types

Lecture 8: RSL II Tu.16.11.2010 97–112

Values and Operations

Lecture 9: RSL III We.17.11.2010 112–128

Logic, Λ-Calculus, Other Applicative Constructs

Lecture 10: RSL IV Th.18.11.2010 128–141

Imperative Constructs, Process Constructs, Specifications

Lecture 11: Conclusion Fr.19.11.2010 74–76

Contents

1 Introduction 8
1.1 The Problem . . . 8
1.2 General Remarks . . . 8
1.2.1 What are Domains . . . 8
1.2.2 What is a Domain Description . . . 9
Description Languages . . . 9
1.2.3 Contributions of These Lecture Notes . . . 9
1.2.4 Relation to Other Engineering Disciplines . . . 10
1.3 The Triptych Approach . . . 10
1.4 On The Structure of These Lecture Notes . . . 11
1.5 The Comparative Methodology Endeavour . . . 11
1.6 Caveat . . . 12

2 An Ontology of Specification Entities 13
2.1 Simple Entities . . . 13
2.1.1 Net, Hubs and Links . . . 13
2.1.2 Unique Hub and Link Identifiers . . . 13
2.1.3 Observability of Hub and Link Identifiers . . . 14
2.1.4 A Theorem . . . 14
Links implies Hubs . . . 14
2.1.5 Hub and Link Attributes . . . 15
2.1.6 Hub and Link Generators . . . 15
2.2 States . . . 16
2.3 Actions . . . 16
2.3.1 Insert Hubs . . . 16
2.3.2 Remove Hubs . . . 17
2.3.3 Insert Links . . . 17
2.3.4 Remove Links . . . 18
2.3.5 Two Theorems . . . 19
Idempotency . . . 19
Reachability . . . 20
2.4 Events . . . 20
2.5 Behaviours . . . 21
2.5.1 Behaviour Prescriptions . . . 22
Construction Plans . . . 22
Wellformedness of Construction Plans . . . 22
2.5.2 Augmented Construction Plans . . . 23
2.5.3 Sequential Construction Behaviours . . . 24

3 An Ontology of Domain Facets 26
3.1 What Can Be Observed . . . 26
3.2 Intrinsics . . . 26
3.2.1 Net Topology Descriptors . . . 26
3.2.2 Link States and Link State Spaces . . . 27
3.2.3 Hub States and Hub State Spaces . . . 28
3.2.4 State and State Space Wellformedness . . . 29
3.2.5 Concrete Types for Simple Entities . . . 30
3.2.6 Example Hub Crossings . . . 30
3.2.7 Actions Continued . . . 31
3.3 Support Technologies . . . 32

3.3.1 Traffic Signals . . . 32
3.3.2 Traffic "Control" . . . 34
3.4 Rules and Regulations . . . 34
3.4.1 Vehicles . . . 35
3.4.2 Traffic . . . 36
Wellformedness of Traffic . . . 36
• Static Wellformedness . . . 36
• Dynamic Wellformedness . . . 37
3.4.3 Traffic Rules (I of II) . . . 39
3.4.4 Another Traffic Regulator . . . 39
3.4.5 Traffic Rules (II of II) . . . 39
3.5 Scripts . . . 40
3.5.1 Routes as Scripts . . . 40
Paths . . . 40
Routes . . . 41
3.5.2 Bus Timetables as Scripts . . . 42
Buses . . . 42
Bus Stops . . . 42
Bus Routes . . . 43
Bus Schedule . . . 44
Timetable . . . 44
3.5.3 Route and Bus Timetable Denotations . . . 45
3.5.4 Licenses and Contracts . . . 46
Contracts . . . 46
Contractual Actions . . . 48
Wellformedness of Contractual Actions . . . 49
3.6 Management and Organisation . . . 51
3.6.1 Transport System Examples . . . 51
3.7 Human Behaviour . . . 52
3.8 Towards Theories of Domain Facets . . . 52
3.8.1 A Theory of Intrinsics . . . 52
3.8.2 Theories of Support Technologies . . . 52
An Example . . . 52
General . . . 53
3.8.3 A Theory of Rules & Regulations . . . 54
3.8.4 A Theory of Management & Organisation . . . 55
3.8.5 A Theory of Human Behaviour . . . 55

4 An Ontology of Requirements Constructions 57
4.1 Business Process Re-engineering . . . 57
4.1.1 The Kinds of Requirements . . . 57
4.1.2 Goals Versus Requirements . . . 58
Goals of a Toll Road System . . . 58
Goals of Toll Road System Software . . . 58
Arguing Goal-satisfaction of a Toll Road System . . . 58
Arguing Goal-satisfaction of Toll Road System Software . . . 59
4.1.3 Re-engineered Nets . . . 59
4.2 Domain Requirements . . . 61
4.2.1 Projection . . . 61
4.2.2 Instantiation . . . 62
Example . . . 62

Abstraction: From Concrete Toll Road Nets to Abstract Nets . . . 63
Theorem . . . 64
4.2.3 Determination . . . 64
Example . . . 64
4.2.4 Extension . . . 65
Intuition . . . 65
Descriptions . . . 66
• A RAISE/CSP Model . . . 66
Toll Booth Plazas . . . 66
Cars . . . 67
Entry Booths . . . 67
Gates . . . 67
The Entry Plaza System . . . 68
• A Duration Calculus Model . . . 69
• A Timed Automata Model . . . 70
4.2.5 Fitting . . . 70
Examples . . . 71
4.3 Interface Requirements . . . 71
4.3.1 But First: On Shared Phenomena and Concepts . . . 72
4.3.2 Shared Simple Entities . . . 72
Example . . . 72
4.3.3 Shared Actions . . . 72
Example . . . 72
4.3.4 Shared Events . . . 72
Examples . . . 72
4.3.5 Shared Behaviours . . . 73
Example . . . 73
4.4 Machine Requirements . . . 73
4.4.1 An Enumeration of Classes of Machine Requirements . . . 73

5 Conclusion 74
5.1 What Have We Omitted . . . 74
5.2 Domain Descriptions Are Not Normative . . . 74
5.3 “Requirements Always Change” . . . 74
5.4 What Can Be Described and Prescribed . . . 74
5.5 What Have We Achieved – and What Not . . . 75
5.6 Relation to Other Work . . . 75
5.7 “Ideal” Versus Real Developments . . . 75
5.8 Description Languages . . . 75
5.9 Entailments . . . 75
5.10 Domain Versus Ontology Engineering . . . 76

6 Bibliographical Notes 76
6.1 Description Languages . . . 76
6.2 References . . . 76

A An RSL Primer 89
A.1 Types . . . 89
A.1.1 Type Expressions . . . 89
Atomic Types . . . 89
Example 1: *Basic Net Attributes* . . . 89
Composite Types . . . 90

Example 2: *Composite Net Type Expressions* . . . 90
A.1.2 Type Definitions . . . 91
Concrete Types . . . 91
Example 3: *Composite Net Types* . . . 91
Example 4: *Net Record Types: Insert Links* . . . 93
Subtypes . . . 95
Example 5: *Net Subtypes* . . . 96
Sorts — Abstract Types . . . 97
Example 6: *Net Sorts* . . . 97
A.2 Concrete RSL Types: Values and Operations . . . 97
A.2.1 Arithmetic . . . 97
A.2.2 Set Expressions . . . 98
Set Enumerations . . . 98
Example 7: *Set Expressions over Nets* . . . 98
Set Comprehension . . . 99
Example 8: *Set Comprehensions* . . . 99
A.2.3 Cartesian Expressions . . . 100
Cartesian Enumerations . . . 100
Example 9: *Cartesian Net Types* . . . 100
A.2.4 List Expressions . . . 101
List Enumerations . . . 101
List Comprehension . . . 101
Example 10: *Routes in Nets* . . . 101
A.2.5 Map Expressions . . . 103
Map Enumerations . . . 103
Map Comprehension . . . 103
Example 11: *Concrete Net Type Construction* . . . 104
A.2.6 Set Operations . . . 105
Set Operator Signatures . . . 105
Set Examples . . . 105
Informal Explication . . . 106
Set Operator Definitions . . . 106
A.2.7 Cartesian Operations . . . 107
A.2.8 List Operations . . . 107
List Operator Signatures . . . 107
List Operation Examples . . . 107
Informal Explication . . . 108
List Operator "Definitions" . . . 108
A.2.9 Map Operations . . . 109
Map Operator Signatures and Map Operation Examples . . . 109
Map Operation Explication . . . 110
Example 12: *Miscellaneous Net Expressions: Maps* . . . 111
Map Operation "Redefinitions" . . . 111
A.3 The RSL Predicate Calculus . . . 112
A.3.1 Propositional Expressions . . . 112
A.3.2 Simple Predicate Expressions . . . 112
A.3.3 Quantified Expressions . . . 113
Example 13: *Predicates Over Net Quantities* . . . 113
A.4 λ-Calculus + Functions . . . 114
A.4.1 The λ-Calculus Syntax . . . 114
A.4.2 Free and Bound Variables . . . 114

A.4.3 Substitution . . . 115
A.4.4 α-Renaming and β-Reduction . . . 115
Example 14: *Network Traffic* . . . 115
A.4.5 Function Signatures . . . 118
Example 15: *Hub and Link Observers* . . . 118
A.4.6 Function Definitions . . . 119
Example 16: *Axioms over Hubs, Links and Their Observers* . . . 120
A.5 Other Applicative Expressions . . . 120
A.5.1 Simple let Expressions . . . 120
A.5.2 Recursive let Expressions . . . 120
A.5.3 Non-deterministic let Clause . . . 120
A.5.4 Pattern and "Wild Card" let Expressions . . . 121
A.5.5 Conditionals . . . 121
Example 17: *Choice Pattern Case Expressions: Insert Links* . . . 122
A.5.6 Operator/Operand Expressions . . . 128
A.6 Imperative Constructs . . . 128
A.6.1 Statements and State Changes . . . 128
A.6.2 Variables and Assignment . . . 129
A.6.3 Statement Sequences and skip . . . 129
A.6.4 Imperative Conditionals . . . 129
A.6.5 Iterative Conditionals . . . 130
A.6.6 Iterative Sequencing . . . 130
A.7 Process Constructs . . . 130
A.7.1 Process Channels . . . 130
Example 18: *Modelling Connected Links and Hubs* . . . 130
A.7.2 Process Definitions . . . 132
Example 19: *Communicating Hubs, Links and Vehicles* . . . 132
A.7.3 Process Composition . . . 133
Example 20: *Modelling Transport Nets* . . . 133
A.7.4 Input/Output Events . . . 135
Example 21: *Modelling Vehicle Movements* . . . 135
A.8 Simple RSL Specifications . . . 137
Example 22: *A Neat Little "System"* . . . 139

B Terminology **142**
B.1 Term Table of Contents . . . 142
B.2 Terms . . . 142
Last page . . . 231

1 Introduction 5

1.1 The Problem

The problem to be solved by this technical note is to present in one specific formal specification language, RSL [112], a domain description and a requirements prescription developed according to the "triptych approach" [34].

1.2 General Remarks

Before we can design software we must have a robust understanding of its requirements. And before we can prescribe requirements we must have a robust understanding of the environment, or, as we shall call it, the domain in which the software is to serve – and as it is at the time such software is first being contemplated.

In consequence we suggest that software, "ideally"[1], be developed in three phases.

First a phase of **domain engineering.** In this phase a reasonably comprehensive description is constructed from an analysis of the domain. That description, as it evolves, is analysed with respect to inconsistencies, conflicts and completeness on one hand, and, on the other hand, in order to achieve pleasing concepts in terms of which to abstractly model the domain (Sect. 3).

Then a phase of **requirements engineering.** This phase is strongly based, as we shall see (in Sect. 4), on an available, necessary and sufficient domain description. Guided by the domain and requirements engineers the *requirements stakeholders* point out which domain description parts are to be left (*projected*) out of the *domain requirements*, and of those left what forms of *instantiations*, *determinations* and *extensions* are required. Similarly the requirements stakeholders, guided by the domain and requirements engineers, inform as to which domain *entities*, *actions*, *events* and *behaviours* are *shared* between the domain and the *machine*, that is, the *hardware* and the *software* being required. In these notes we shall only very briefly cover aspects of *machine requirements*.

And finally a phase of **software design.** We shall not cover this phase in these notes.

Methodology

> These notes focus on methodology – where a method is seen as a set of principles (applied by engineers, not machines) for selecting and applying (often with some tool support) techniques (and tools) for the efficient construction of some artifact – here software.

1.2.1 What are Domains

By a domain we shall thus understand a universe of discourse, an area of nature subject to laws of physics and studies by physicists, or an area of human activity (subject to its interfaces with nature). There are other domains which we shall ignore. We shall focus on the human-made domains. "Large scale" examples are *the financial service industry:*

[1] Section 5.7 will discuss practical renditions of "idealism"!

banking, insurance, securities trading, portfolio management, etc., *health care: hospitals, clinics, patients, medical staff, etc.*, *transportation:* road, rail/train, sea, and air transport (vehicles, transport nets, etc.); *oil and gas systems:* pumps, pipes, valves, refineries, distribution, etc. "Intermediate scale" examples are *automobiles:* manufacturing or monitoring and control, etc.; and *heating systems*.

The above explication was "randomised": for some domains, to wit, *the financial service industry*, we mentioned major functionalities, for others, to wit, *health care*, we mentioned major entities. An objection can be raised, namely that the above characterisation – of what a domain is – is not sufficiently precise. We shall try, in the next section, to partially meet this objection.

1.2.2 What is a Domain Description

By a *domain description* we understand a description of the *entities*, the *actions*, the *events* and the *behaviours* of the domain, including its *interfaces* to other domains. A domain description describes the domain **as it is**. A domain description does not contain requirements let alone references to any software. Michael Jackson, in [139], refers to domain descriptions as *indicative* (stating objective fact), requirements prescriptions as *optative* (expressing wish or hope) and software specifications as *imperative* ("do it!"). A description is *syntax*. The meaning (*semantics*) of a domain description is usually a set of *domain models*. We shall take domain models to be *mathematical structures (theories)*. The form of domain descriptions that we shall advocate "come in pairs": precise, say, English, i.e., narrated text (narratives) alternates with clearly related formula text.

Description Languages Besides using as precise a subset of a national language, as here English, as possible, and in enumerated expressions and statements, we "pair" such narrative elements with corresponding enumerated clauses of a formal specification language. We shall be using the `RAISE` Specification Language, `RSL`, [112], in our formal texts. But any of the model-oriented approaches and languages offered by `Alloy` [138], `Event B` [3], `VDM` [107] and `Z` [234], should work as well. No single one of the above-mentioned formal specification languages, however, suffices. Often one has to carefully combine the above with elements of `Petri Nets` [200], `CSP: Communicating Sequential Processes` [128], `MSC: Message Sequence Charts` [137], `Statecharts` [120], and some temporal logic, for example either `DC: Duration Calculus` [236] or `TLA+` [148]. Research into how such diverse textual and diagrammatic languages can be meaningfully and proof-theoretically combined is ongoing [9].

1.2.3 Contributions of These Lecture Notes

We claim that the major contributions of the triptych approach to software engineering as presented in these notes are the following: (1) the clear *identification* of domain engineering, or, for some, its clear *separation* from requirements engineering (Sects. 3 and 4); (2) the *identification* and *'elaboration'* of the pragmatically determined domain *facets* of *(a) intrinsics, (b) support technologies, (c) rules and regulations, (d) scripts (licenses and contracts)*, (e) management and organisation, and *(f) human behaviour* whereby 'elaboration' we mean that we provide principles and techniques for the construction of these facet description parts (Sects. 3.2–3.7); (3) the *re-identification* and *'elaboration'* of the concept of *business process reengineering* (Sect. 4.1); (4) the *identification* and *'elaboration'* of the technically determined *domain requirements facets* of *(g) projection, (h) instantiation, (i) determination, (j) extension* and *(k) fitting* requirements principles and techniques – and, in particular the *"discovery"* that these requirements engineering stages are strongly dependent on necessary and sufficient domain descriptions (Sects. 4.2.1–4.2.5); and (5) the *identification* and *'elaboration'* of the technically determined *interface requirements facets* of *(l) shared simple entity, (m) shared action, (n) shared event* and *(o) shared behaviour* requirements principles and techniques (Sects. 4.3.2–4.3.5). We claim that the facets of (2, 3, 4) and (5) are all relatively new.

1.2.4 Relation to Other Engineering Disciplines

An aeronautics engineer – to be hired by `Boeing` to their design team for a next generation aircraft – must be pretty well versed in applied mathematics and in aerodynamics. A radio communications engineer – to be hired by `Ericsson` to their design team for a next generation mobile telephony antennas – must be likewise pretty well versed in applied mathematics and in the physics of electromagnetic wave propagation in matter. And so forth. Software engineers hired for the development of software for hospitals, or for railways, know little, if anything, about health care, respectively rail transportation (scheduliming, rostering, etc.). The `Ericsson` radio communications engineer can be expected to understand Maxwell's Equations, and to base the design of antenna characteristics on the transformation and instantiation of these equations. It is therefore quite reasonable to expect the domain-specific software engineer to understand formalisation of their domains, to wit: *railways:* `www.railwaydomain.org`, and *pipelines:* `pipelines.pdf`, *logistics:* `logistics.pdf`, *transport nets:* `comet1.pdf`, *stock exchanges:* `tse-2.pdf` and *container lines:* `container-paper.pdf` – these latter five at `www.imm.dtu.dk/~db/`.

1.3 The Triptych Approach 6

The "triptych approach" calls for a thorough description (cum analysis) of the domain before one attempts prescribing requirements for specific software.

As part of the triptych approach to domain engineering one starts by exploring the description ontology of specification entities: *simple entities, actions, events* and *behaviours*
7 (Sect. 2) before delving into the description ontology of facets: *intrinsics, support technologies, rules & regulations, scripts (licenses and contracts), management & organisation*
8 and *human behaviour* (Sect. 3).

And, as part of the triptych approach to requirements engineering one starts by exploring the reengineering of business processes before delving into *domain requirements* concepts of *projection, instantiation, determination, extension* and *fitting* – followed by a number of *interface requirements* stages. The *terms in slanted script* are defined in Appendix B.

For a more pedagogic and didactical introduction to these terms we refer to either of [36, 50, 49, 45, 46, 47] or to [34, 39, 44].

1.4 On The Structure of These Lecture Notes

The presentation (i.e., structuring) of the technical material of these lecture nptes is not meant to suggest that all domain descriptions and requirements prescriptions follow this mold. As mentioned just above our presentation follows the structure of *simple entity, action, event,* and *behaviour* specification ontology (Sect. 2), then the structure of the *domain facets: intrinsics, support technology, rules & regulations, scripts (licenses, contracts), management & organisation* and *human behaviour* (Sect. 3), and finally the structure of the *business re-engineering* (Sect. 4.1.3), the *domain requirements* concepts of *projection, instantiation, determination, extension* and *fitting* (Sect. 4.2), and a number of *interface requirements* facets (Sect. 4.3).

I expect such students who might be pursuing specifications based on the example of this document to do so, either, as here, in `RSL` [110], or according to approaches embodied in `Alloy` [138], `CafeOBJ` [109], `Event B` [3], `VDM` [107] and `Z` [234]. But I do not expect them to follow exactly the order used in this document – although it migt well be a good idea, pedagogically and didactically.

Two remarks are in order:

- Rather I expect `Alloy, CafeOBJ, Event B, VDM-SL` and `Z` specifications to follow a "most natural order" appropriate for their approaches.
- The order in which I have chosen to present the current material reflects a both pedagogic and didactic views.[2] In a commercial project I might very well choose another decomposition of the material — being guided, however, by the need to cover all the footnoted (Footnote 2) facets.

1.5 The Comparative Methodology Endeavour

These notes are intended to replace:

- http://www.imm.dtu.dk/˜db/bjorner-8jan2010.pdf
- http://www.complang.tuwien.ac.at/bjorner/book.pdf

which were first suggested as a basis for the Comparative Methodology endeavour, cf.

[2]The sequence of the *simple entity, action, event, behaviour, domain facets: intrinsics, support technology, rules & regulations, scripts (licenses, contracts), management & organisation, human behaviour, domain requirements: projection, instantiation, determination, extension, fitting* and the *interface requirements* facets reflect these views.

- http://www2.imm.dtu.dk/˜db/comet/
- http://formalmethods.wikia.com/wiki/CoMet .

Rewriting the above referenced earlier notes into the present notes were begun after Kokichi Futatsugi's CafeOBJ lectures. I am happy to acknowledge being thus challenged.

1.6 Caveat

The many examples of Sect. A, the RSL Primer, stem from an earlier version of this attempt to give a 'model' presentation of domains and requirements. They have yet to coordinated with the the present rewrite of Sects. 2–4.