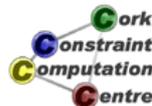


Chapter 12: Systematic Development

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning [Overview](#)



Licence

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.



Outline

- 1 Introduction
- 2 Application Structure
- 3 Documentation
- 4 Data Representation
- 5 Programming Concepts
- 6 Style Guide



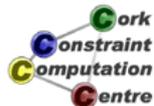
Outline

- 1 Introduction
- 2 Application Structure
- 3 Documentation
- 4 Data Representation
- 5 Programming Concepts
- 6 Style Guide



Overview

- How to develop large applications in ECLiPSe
- Software development issues for Prolog
- This is essential for large applications
 - But it may show benefits already for small programs
- This is not about problem solving, but the *boring bits* of application development



Disclaimer

- This is not *holy writ*
 - But it works!
- This is a team issue
 - People working together must agree
 - Come up with a local style guide
- Consistency is not optional
 - Every shortcut must be paid for later on
- This is an appetizer only
 - The real story is in the tutorial Developing Applications with ECLiPSe (part of the ECLiPSe documentation)



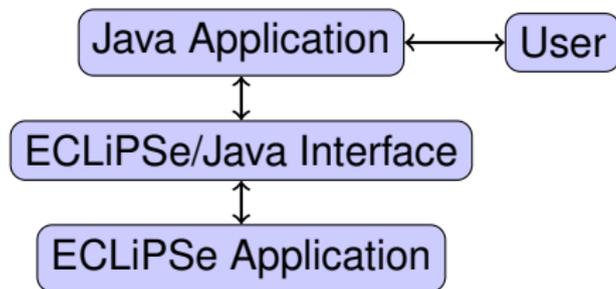
Outline

- 1 Introduction
- 2 Application Structure**
- 3 Documentation
- 4 Data Representation
- 5 Programming Concepts
- 6 Style Guide

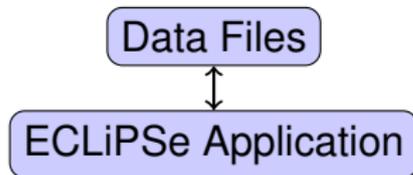


Application Structure

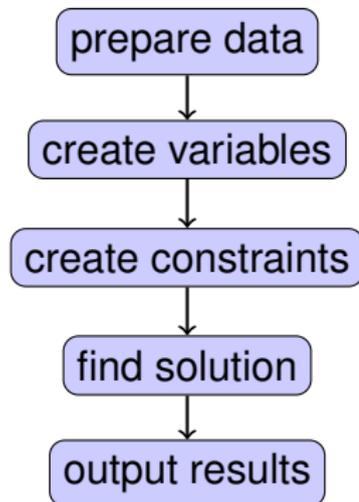
Full Application



Batch Application



LSCO Structure



Top-Down Design

- Design queries
- UML static class diagram (structure definitions)
- API document/test cases
- Top-level structure
- Data flow analysis
- Allocate functionality to modules
- Syntactic test cases
- Module expansion
 - Using programming concepts where possible
 - Incremental changes



Modules

- Grouping of predicates which are related
- Typically in a single file
- Defined external interfaces
 - Which predicates are exported
 - Mode declaration for arguments
 - Intended types for arguments
 - Documentation
- Helps avoid Spaghetti structure of program



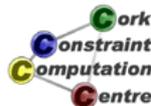
Outline

- 1 Introduction
- 2 Application Structure
- 3 Documentation**
- 4 Data Representation
- 5 Programming Concepts
- 6 Style Guide



Creating Documentation

- Your program can be documented in the same way as ECLIPSe library predicates
- Comment directives in source code
- Tools to extract comments and produce HTML documentation with hyper-links
- Quality depends on effort put into comments
- Every module interface should be documented



Example

```
:- comment(prepare_data/4, [
    summary:"creates the data structures
for the flow analysis",
    amode:prepare_data(+,+,+,-),
    args:[
"Dir":"directory for report output",
"Type":"the type of report to be generated",
"Summary":"a summary term",
"Nodes":"a nodes data structure"],
    desc:html("
```

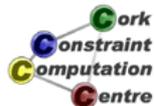
This routine creates the data structures for the flow analysis.

...



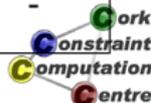
Outline

- 1 Introduction
- 2 Application Structure
- 3 Documentation
- 4 Data Representation**
- 5 Programming Concepts
- 6 Style Guide



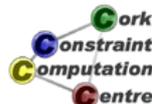
External Data Representation

Property	Argument	Data File	Term File	Facts	EXDR
Multiple runs	++	+	+	-	+
Debugging	-	+	+	++	-
Test generation effort	-	+	+	+	-
Java I/O Effort	-	+	-	-	+
ECLiPSe I/O Effort	++	+	++	++	++
Memory	++	-	-	-	-
Development Effort	+	-	+	+	-



Internal Data Representation

- Named structures
 - Define & document properly
- Lists
 - Do not use for fixed number of elements
- Hash tables, e.g. `lib(hash)`
 - Efficient
 - Extensible
 - Multiple keys possible
- Vectors & arrays
 - Requires that keys are integers (tuples)
- Multi-representation
 - Depending on key use one of multiple representations



Internal Representation Comparison

	Named Structures	Lists	Hash Tables	Vectors Arrays	Multi-representation
hold disparate data	++	-	-	-	-
access specific info	+	-	+	+	+
add new entries	-	+	++	-	-
do loops	+	++	-	++	++
sort entries	-	++	-	-	++
index calculations	-	-	-	++	+ 

Getting it to work

- Early testing `lib(test_util)`
 - Define what a piece of code should do by example
 - May help to define behaviour
- Stubs
- Line coverage `lib(coverage)`
 - Check that tests cover code base
- Heading warnings of compiler, `lib(lint)`
 - Eliminate all causes of warnings
 - Singleton warnings typically hide more serious problems
- Small, incremental changes
 - Matter of style
 - Works for most people



Outline

- 1 Introduction
- 2 Application Structure
- 3 Documentation
- 4 Data Representation
- 5 Programming Concepts**
- 6 Style Guide



Programming Concepts

- Many programming tasks are similar
 - Finding the right information
 - Putting things together in the right sequence
- We don't need the fastest program, but the easiest to maintain
 - Squeezing the last 10% improvement normally does not pay
- Avoid unnecessary inefficiency
 - `lib(profile), lib(port_profiler)`



List of concepts

- Alternatives
- Iteration (list, terms, arrays)
- Transformation
- Filtering
- Combine
- Minimum/Best and rest
- Sum
- Merge
- Group
- Lookup
- Cartesian
- Ordered pairs



Example: Cartesian

```
:-mode cartesian(+,+,-).  
cartesian(L,K,Res):-  
    (foreach(X,L),  
     fromto([],In,Out,Res),  
     param(K) do  
         (foreach(Y,K),  
          fromto(In,In1,[pair(X,Y)|In1],Out),  
          param(X) do  
              true  
          )  
        )  
    ).
```

Input/Output

- Section on DCG use
 - Grammars for parsing and generating text formats
- XML parser in ECLiPSe
 - `lib(xml)`
- EXDR format to avoid quoting/escaping problems
- Tip:
 - Generate hyper-linked HTML/SVG output to present data/results as development aid

If it doesn't work

- Understand what happens
 - Which program point should be reached with which information?
 - Why do we not reach this point?
 - Which data is wrong/missing?
- Do not trace through program!
- Debugging is like solving puzzles
 - Pick up clues
 - Deduce what is going on
 - Do not simulate program behaviour!

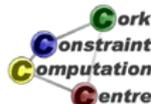
Correctness and Performance

- Testing
- Profiling
- Code Reviews
 - Makes sure things are up to a certain standard
 - Don't expect reviewer to find bugs
- Things to watch out for
 - Unwanted choice points
 - Open streams
 - Modified global state
 - Delayed goals



Did I mention testing?

- Single most important/neglected activity
- Re-test directly after every change
 - Identifies faulty modification
 - Avoids lengthy debugging session after making 100s of changes
- Independent verification
 - Check results by hand (?)
 - By other program (??)
 - Use constraint solver as checker

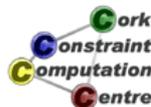


Outline

- 1 Introduction
- 2 Application Structure
- 3 Documentation
- 4 Data Representation
- 5 Programming Concepts
- 6 Style Guide

Style Guide

- Rules that should be satisfied by finished program
- Things may be relaxed during prototyping
- Often, choice among valid alternatives is made arbitrarily, so that a consistent way is defined
- If you don't like it, change it!
 - But: better a bad rule than no rule at all!



Style Guide Examples

- There is one directory containing all code and its documentation (using sub-directories).
- Filenames are of form `[a-z][a-z_]+` with extension `.ecl`.
- One file per module, one module per file.
- Each module is documented with comment directives.
- ...
- Don't use `' , ' / 2` to make tuples.
- Don't use lists to make tuples.
- Avoid `append/3` where possible, use accumulators instead.



Layout rules

- How to format ECLiPSe programs
- Pretty-printer format
- Eases
 - Exchange of programs
 - Code reviews
 - Bug fixes
 - Avoids extra reformatting work



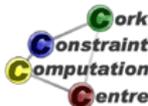
Core Predicates List

- Alphabetical predicate index lists 2940 entries
 - You can't possibly learn all of them
 - Do you really want to know what `set_typed_pool_constraints/3` does?
- List of Prolog predicates you need to know
 - 69 entries, more manageable
- Ignores all solver libraries
- If you don't know what an entry does, find out about it
 - what does `write_exdr/2` do?
- If you use something not on the list, start to wonder...



Other Sources

- Developing Applications with ECLiPSe
 - H. Simonis
 - <http://www.eclipse-clp.org>
- Constraint Logic Programming Using ECLiPSe
 - K. Apt, M. Wallace
 - Cambridge University Press
- The Craft of Prolog
 - R.O'Keefe, MIT Press



Conclusions

- Large scale applications can be built with ECLiPSe
- Software engineering is not that different for Prolog
- Many tasks are similar regardless of solver used
- Correctness of program is useful even for research work