Experience in developing the mCRL2 toolset

J.F. Groote J.J.A. Keiren F.P.M. Stappers J.W. Wesselink
T.A.C. Willemse

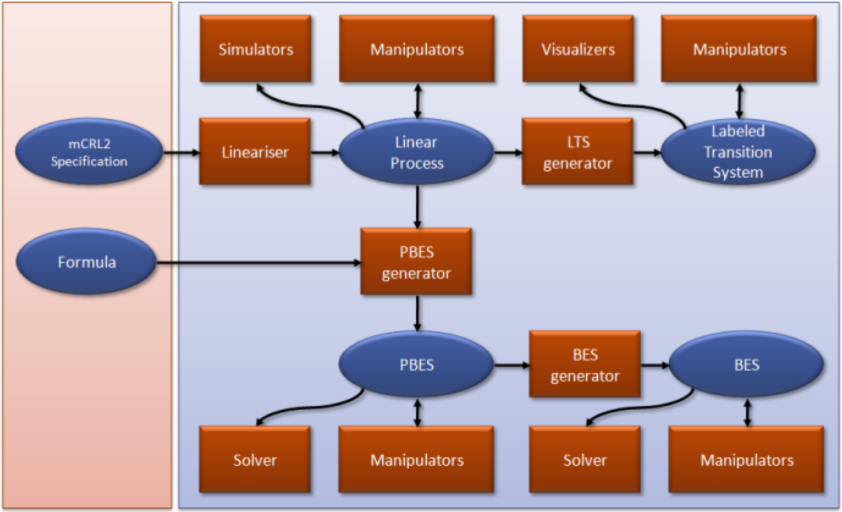
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven

22 February 2010

mCRL2, what is it?

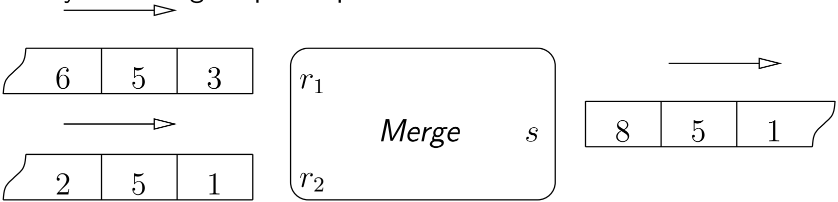
- Toolset for verification of software
- Process algebra based
- Explicit state model checking + symbolic transformations
- Support for modal μ-calculus with data + time

mCRL2, overview

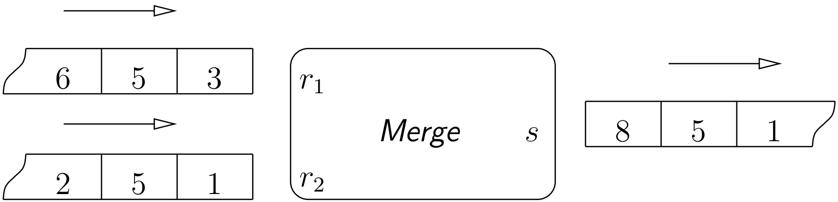


Example

Process that merges two streams of natural numbers, producing a locally ascending output sequence

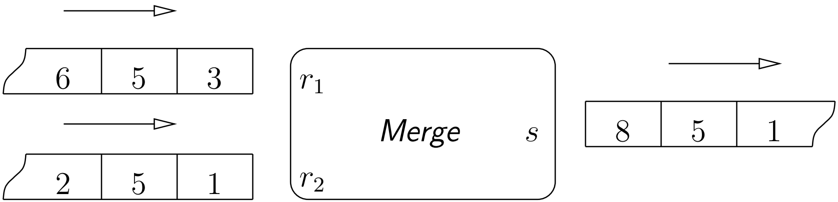


Example: specification



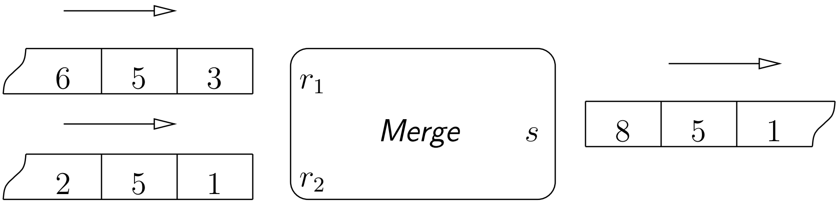
$$
\begin{array}{lcl}
Merge & = & \sum m : \mathbb{N}.\ (r_1(m) \cdot Merge_1(m) \\
 & & +r_2(m) \cdot Merge_2(m)) \\
Merge_1(n : \mathbb{N}) & = & \sum m : \mathbb{N}.\ r_2(m) \cdot Merge_3(n, m) \\
Merge_2(m : \mathbb{N}) & = & \sum n : \mathbb{N}.\ r_1(n) \cdot Merge_3(n, m) \\
Merge_3(n, m : \mathbb{N}) & = & n \leqslant m \rightarrow s(n) \cdot Merge_2(m) \\
 & & +m \leqslant n \rightarrow s(m) \cdot Merge_1(n)
\end{array}
$$

Example: linear process



$$
\begin{array}{rl}
 & Merge(\sigma, i_1, i_2 : \mathbb{N}) \\
= & \\
 & \sum m : \mathbb{N}.\ \sigma = 0 \rightarrow r_1(m) \cdot Merge(2, m, i_2) \\
+ & \sum m : \mathbb{N}.\ \sigma = 0 \rightarrow r_2(m) \cdot Merge(1, i_1, m) \\
+ & \sum m : \mathbb{N}.\ \sigma = 1 \rightarrow r_1(m) \cdot Merge(3, m, i_2) \\
+ & \sum m : \mathbb{N}.\ \sigma = 2 \rightarrow r_2(m) \cdot Merge(3, i_1, m) \\
+ & \sum m : \mathbb{N}.\ \sigma = 3 \wedge i_1 \leqslant i_2 \rightarrow s(i_1) \cdot Merge(1, i_1, i_2) \\
+ & \sum m : \mathbb{N}.\ \sigma = 3 \wedge i_2 \leqslant i_1 \rightarrow s(i_2) \cdot Merge(2, i_1, i_2)
\end{array}
$$

Example: property



If the input streams are ascending, then the output stream is ascending.
Modal formula:

$$
\begin{aligned}
(\nu X(j_1, j_2, o{:}\mathbb{N} := 0,0,0).\forall l{:}\mathbb{N}.\quad & ([r_1(l)](l \geq j_1 \Rightarrow X(l, j_2, o)) \wedge \\
& [r_2(l)](l \geq j_2 \Rightarrow X(j_1, l, o)) \wedge \\
& [s(l)](l \geq o \wedge X(j_1, j_2, l))))
\end{aligned}
$$

Example: parameterised Boolean equation system

Combining LPS and formula, and applying simplifications, we get

$$
\begin{aligned}
&\nu X(\sigma, i_1, i_2, j_1, j_2, o{:}\mathbb{N}) = \forall l{:}\mathbb{N} \\
&(\sigma = 0 \Rightarrow l \geq j_1 \Rightarrow X(2, l, i_2, l, j_2, o)) \\
&\wedge(\sigma = 0 \Rightarrow l \geq j_2 \Rightarrow X(1, i_1, l, j_1, l, o)) \\
&\wedge(\sigma = 1 \Rightarrow l \geq j_2 \Rightarrow X(3, i_1, l, j_1, l, o)) \\
&\wedge(\sigma = 2 \Rightarrow l \geq j_1 \Rightarrow X(3, l, i_2, l, j_2, o)) \\
&\wedge((\sigma = 3 \wedge i_1 \leq i_2) \Rightarrow (i_1 \geq o \wedge X(1, i_1, i_2, j_1, j_2, i_1))) \\
&\wedge((\sigma = 3 \wedge i_2 \leq i_1) \Rightarrow (i_2 \geq o \wedge X(2, i_1, i_2, j_1, j_2, i_2))))
\end{aligned}
$$

Observe that $i_1 = j_1 \wedge i_2 = j_2 \wedge o \leq \min(i_1, i_2)$ is an invariant, now this can automatically be solved, and the property holds for $Merge(0, 0, 0)$

History of mCRL2

- Based on μCRL
- Development started in 2004, using code from μCRL
- 22 developers with commit access
- Currently 240949 Lines of code (obtained using SLOCcount)

Process related issues

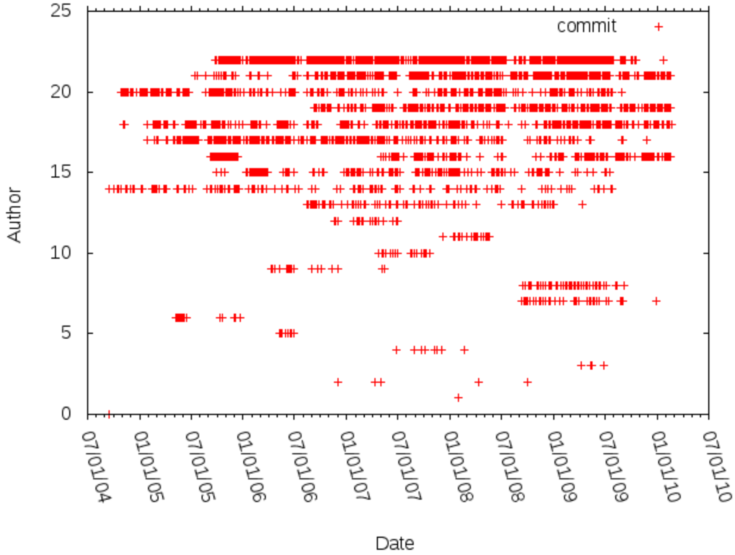
1. Rapid turnover of resources
2. Programming not 1^{st} priority

Three kinds of developers:

- Scientific programmers: long period of time (3+ years), dedicated time available
- (Bachelor + Master) students: short period of time (3-6 months), little time for programming
- Other staff: long period of time (3+ years), little time for programming, not 1^{st} priority

Leads to single components that have been modified by 10 people in disjoint time spans

Commits made per developer



Consequences

- Poorly documented code
- Lack of user documentation
- Most code not maintained by original author(s)
- Multiple ad-hoc solution attaining same goal; code duplication
- Little attention for refactoring/restructuring code

Code base related issues

1. Adopting an existing code base
   - Fixes major design choices
   - Based on ATerm library (having untyped interfaces)
   - Low level of abstraction
   - Inheriting other people's choices/bugs
   - Poorly programmed (heavy use of unsafe type casts)

Code base related issues

2. Poor documentation of code
   - Hardly any explicit documentation; choices exists in developer's minds
   - Big trouble when developers leave!
3. Supporting multiple platforms
   - Was problematic with legacy code

Lessons learned

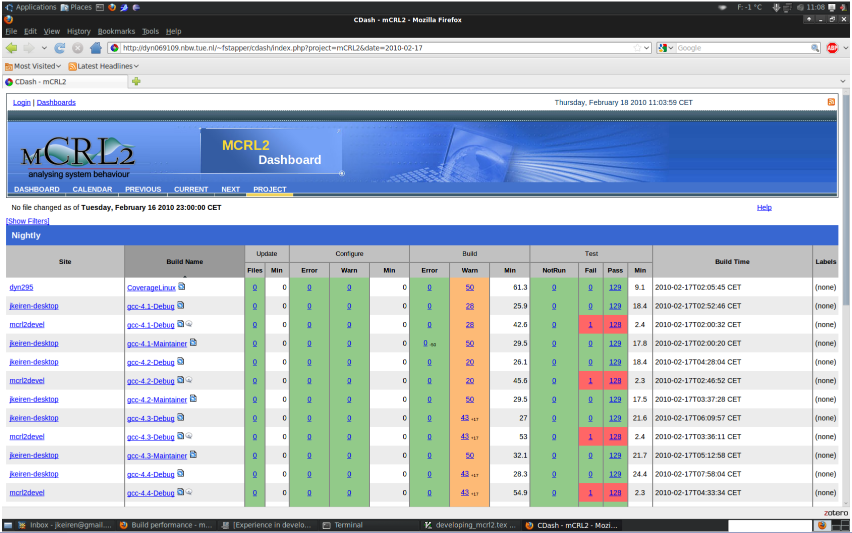
1. Document properly
   - Four kinds of documentation:
     1. User documentation (aimed at tool users)
     2. User documentation (aimed at library users)
     3. Source code documentation (using Doxygen)
     4. Algorithm documentation (detailed algorithm descriptions)
   - Documentation is reviewed by another developer

   Results:
   - Decrease learning curve for new developers
   - Decrease source code duplication

Lessons learned

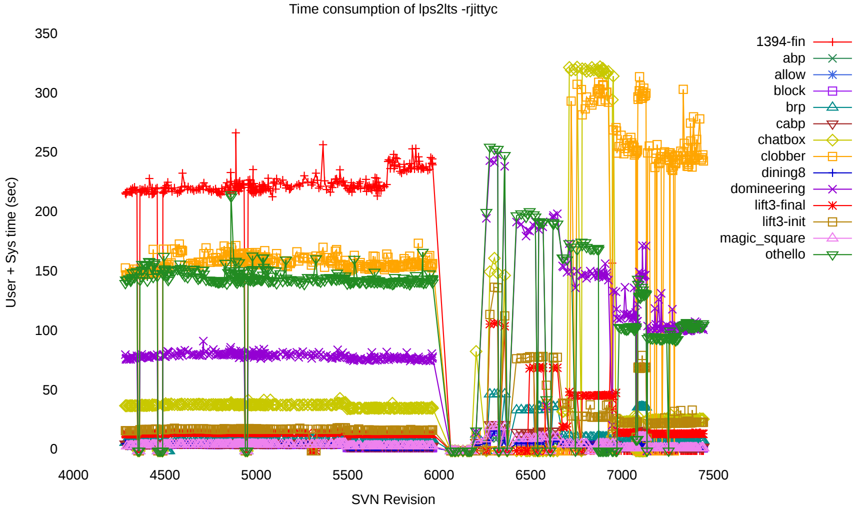
2. Perform systematic tests
   - New components are equipped with unit tests
   - Unit test added for each bug
   - Performance of tools monitored using real-life examples
   - Tests and performance measurements run automatically on a daily basis, with web reports available
   - For performance measurements: historic data, observe trends

Online regression test results



Site	Build Name	Update		Configure			Build			Test				Build Time	Labels
		Files	Min	Error	Warn	Min	Error	Warn	Min	NotRun	Fail	Pass	Min		
dyn295	CoverageLinux	0	0	0	0	0	0	50	61.3	0	0	129	9.1	2010-02-17T02:05:45 CET	(none)
jkeiren-desktop	gcc-4.1-Debug	0	0	0	0	0	0	28	25.9	0	0	129	18.4	2010-02-17T02:52:46 CET	(none)
mcrl2devel	gcc-4.1-Debug	0	0	0	0	0	0	28	42.6	0	1	128	2.4	2010-02-17T02:00:32 CET	(none)
jkeiren-desktop	gcc-4.1-Maintainer	0	0	0	0	0	0 -50	50	29.5	0	0	129	17.8	2010-02-17T02:00:20 CET	(none)
jkeiren-desktop	gcc-4.2-Debug	0	0	0	0	0	0	20	26.1	0	0	129	18.4	2010-02-17T04:28:04 CET	(none)
mcrl2devel	gcc-4.2-Debug	0	0	0	0	0	0	20	45.6	0	1	128	2.3	2010-02-17T02:46:52 CET	(none)
jkeiren-desktop	gcc-4.2-Maintainer	0	0	0	0	0	0	50	29.5	0	0	129	17.5	2010-02-17T03:37:28 CET	(none)
jkeiren-desktop	gcc-4.3-Debug	0	0	0	0	0	0	43 +17	27	0	0	129	21.6	2010-02-17T06:09:57 CET	(none)
mcrl2devel	gcc-4.3-Debug	0	0	0	0	0	0	43 +17	53	0	1	128	2.4	2010-02-17T03:36:11 CET	(none)
jkeiren-desktop	gcc-4.3-Maintainer	0	0	0	0	0	0	50	32.1	0	0	129	21.7	2010-02-17T05:12:58 CET	(none)
jkeiren-desktop	gcc-4.4-Debug	0	0	0	0	0	0	43 +17	28.3	0	0	129	24.4	2010-02-17T07:58:04 CET	(none)
mcrl2devel	gcc-4.4-Debug	0	0	0	0	0	0	43 +17	54.9	0	1	128	2.3	2010-02-17T04:33:34 CET	(none)

Online performance measurements



Lessons learned

3. Enhance reusability of code
    - Generate code from smaller specifications
    - Use high level of abstraction:
        - Transition legacy code from C to C++
        - Use object oriented techniques
        - Use generic programming techniques (now + future)
4. Use generic, cross-platform libraries (BOOST, wxWidgets)
5. Strictly adhere to C++ standard

Future directions

- Systematically generating code from specifications (started)
  - Allow flexibility in supported input language
  - Avoid tedious handwriting of uniform sections of code
  - Prevent typo's and copy-paste mistakes
- Perform testing with random inputs
- Increase use of generic programming techniques