Practical applications of process calculi in industrial projects

Hubert Garavel

INRIA Rhône-Alpes / VASY

http://www.inrialpes.fr/vasy

Research at INRIA / VASY

- Study of asynchronous concurrent systems
- Compilers for process calculi
- Model checking tools
- Software: CADP verification toolbox
- Industrial projects
  - Airbus, Bull, CEA/Leti, ST Microelectronics
- SENVA team = CWI/SEN2 + VASY

A challenging question

... raised during a evaluation meeting in Grenoble (January 2006)

- *Why using process calculi as programming languages ? Process calculi are formalisms to study theoretical aspects of concurrency, not languages to be compiled*

A recurrent debate

- Not only in computer science
- "Pure" mathematics
  - Driven by abstract "beauty", not practical usefulness
  - Example: Bourbaki group
- Applied mathematics
  - Inspired from the modeling of real-world problems
  - Example: Lions' school (INRIA, X, ...)
- Both may coexist and strengthen each other
- Theory benefits from applications to establish its relevance and identify new research directions

Towards "Applied concurrency"?

- From the beginning, concurrency theory has been rooted in concrete examples:
  - 80's: scheduler example in Milner's book
  - 80's: LOTOS standard to specify OSI protocols
- Some active teams in "applied concurrency"
  - INRIA / VASY team
  - CWI / SEN2 team (Jaco van de Pol)
  - Eindhoven Univ. of Technology (Jan Friso Groote)
  - Oxford Univ. / Formal Methods Europe
  - etc.

Applied concurrency in action

- Find companies that need to design **reliable** systems involving asychronous concurrency
  - embedded software
  - hardware architectures
  - *(more companies than research teams)*
- Model these systems formally:
  - Dutch style: "*we model it for you*"
  - French style: "*learn and model it yourself*"
- Reuse models for verification, prototyping, testing, performance evaluation, etc.
- Get feedback for research on languages, algorithms, and tools

Permanent fight against complexity

- Complexity of industrial problems increases (or do we model more details?)
- LOTOS examples handled by CADP:
  - 80's : < 100 lines
  - 90's : < 1,000 lines
  - 2000's : < 10,000 lines
- Tool capabilities increase too (probably not enough...)

Three issues faced when applying process calculi in industry

Issue #1: Lack of expressiveness

- Not so much an issue in practice…
- Standard process calculi (LOTOS, FDR2, mCRL) seem to be sufficient for many problems
- So far, little demand for real-time and mobility extensions
- Stochastic extensions have a big potential: Merging verification and performance evaluation would reduce costs significantly

Issue #2: Fragmentation

- Incompatible languages: LOTOS, mCRL, FDR2 are similar and differ only by details
- Too many process calculi $\Rightarrow$ confusion for industrial users. Which language to choose?
- Modelling is a service => proximity criterion
  - UK companies use CSP
  - Dutch companies use mCRL
  - VASY partners use LOTOS, etc.

Issue #2: Remedies to fragmentation

- Source-to-source translators (VASY):
  - FSP → LOTOS, CHP → LOTOS, FDR → LOTOS
- Software gateways (SENVA = SEN2 + VASY)
  - mCRL compiler → CADP model-checkers
  - LOTOS compiler → mCRL minimization tool

- On the long term, a unique language will probably emerge (such as Java for object-oriented languages)

Issue #3: Lack of user-friendliness

- A key issue in the "*model it yourself*" approach
- Economical factors to be considered:
  - Modeling is a time consuming activity
  - Not enough experts trained to formal methods
  - Industry engineers know programming languages not process calculi
  - Process calculi have a steep learning curve
  - They are too different from mainstream programming languages

Issue #3: Remedies

- To ease industrial adoption, we need better languages than today
- Goals:
  - Reduce learning time
  - Allow faster modeling
- Ideas:
  - In large specifications, 80% is standard sequential code and 20% only relates to concurrency
  - Merge process calculi with programming languages
  - Do not let SOS rules dictate the shape of the language

A simple example

read 10 integers and output their sum

normally expected:

```
process SUM [GET, PUT]
   var N, X : int
   N := 0 ;
   for I = 1 to 10 loop
      GET ?X ;
      N := N + X
   end loop  ;
   PUT !N
end process
```

current:

```
process SUM [GET, PUT] : exit :=
SUM2 [GET, PUT] (1, 0)
where
   process SUM2 [GET, PUT] (I, N : nat)
    : exit :=
      [N < 10] ->
        GET ?X:nat ;
          SUM2 [GET, PUT] (I+1, N +X)
      []
      [N = 10] ->
          PUT !N;
            exit
    endproc
endproc
```

The E-LOTOS project (1992-2001)

- Enhanced-LOTOS = complete redesign of LOTOS
  - data types
  - imperative / functional style
  - quantitative time
  - modules

  $\Rightarrow$ International standard ISO 15437:2001
- But:
  - Probably too complex
  - Too many SOS rules
  - Never implemented entirely
- A interesting step towards better process calculi

Two new opportunities in a fast evolving context

Opportunity #1 : Models everywhere

- New methodologies for software development
- MDA (*Model Driven Architecture*) / MDE (*Model Driven Engineering*)
- Emphasis on "models" (seen as first-class entities)
- Familiar concepts:
  - Models, as abstractions of real systems
  - Transformation between models (refinement)
  - Specification languages (often graphical)
  - Custom languages ("*domain specific*")

Opportunity #1 : Models everywhere

- Limitations of MDA/MDE for concurrent systems:
  - No formal language to describe concurrent behaviours
  - Only architectural aspects are captured correctly
  - Transformations are mostly syntax-driven
- Concurrency theory has much to teach:
  - Proper description of behaviours
  - Formal semantics (SOS rules, LTS model…)
  - Link with verification and proof techniques (bisimulation, congruence, etc.)

Opportunity #2: Parallel machines everywhere

- Sequential processors have reached their limits
- No more MHz, but several cores (2, 4, 8, …) in the CPU
- Soon: Dual core (since 2005), multicore (in 2007)
- Concurrency everywhere: from laptops to clusters/grids
- Advantages:
  - Simultaneous execution of sofware complex applications
  - Evolution from time-sharing to "true" multi-tasking
- A turning point in software development:
  - Sequential applications won't run faster than today
  - Only concurrent programs will benefit from new processors
  - Software industry is not ready yet

Conclusion

A promising future

- The present context is ideal
- Concurrency everywhere:
  - Hardware: new machines are parallel machines (multicore CPU, clusters, grids…)
  - Software: must shift to concurrency to benefit from new hardware capabilities
- Industry becomes aware:
  - Increasing need for hardware and software reliability
  - Models (even non-formal) become standard practice
- "Applied concurrency" starts being effective

Concurrency theory has a role to play

- Improve dissemination of theoretical results
  - (Try to) influence model-driven approaches
  - (Try to) replace non-formal models by formal ones
- Provide "better" process calculi
  - Industry needs simpler languages
  - Avoid unecessary fragmentation
  - Focus on user-friendliness, not simply expressiveness
  - Merge process calculi into mainstream programming languages
- Enhance verification tools
  - Process calculi fit well with automated verification
  - Major challenge: state explosion problem