Performance Analysis of Domain Specific Models

Antonio Vallecillo
[Javier Troya, Francisco Durán, J.E. Rivera]
Atenea Research Group
IPCE 2011, Karlsruhe, Germany. March 2011
**Preliminaries**

**Model (of a <X>):** A representation or specification of a <X> from a given point of view and with a particular purpose.

**Prototype model:** A functional model of a <X>, where the emphasis is on testing – e.g., to verify the design.

**Domain Specific Model:** A model written in a domain specific language.

**Domain Specific Language:** A language which offers concepts and notations closer to the domain experts, at an appropriate level of abstraction, and with a particular purpose.

**Model Transformation:** An algorithmic specification (declarative or operational) of the relationship between models.

**Performance Analysis:** the process of evaluating how a particular system is functioning (or will work), with the aims to

- ensure that the system is working at optimum efficiency;

- identify and correct issues that may negatively impact that efficiency;

- help the engineer adjust components so that they make the best use of available resources.
Model-Driven Performance Analysis

Palladio Component Model

- Comp. Dev. DSL Instance
- Soft. Arch. DSL Instance
- Sys. Depl. DSL Instance
- Dom. Exp. DSL Instance

Transformations:
- Stochastic Regular Expr.
- SPA with Scheduling
- Queueing Network
- Performance Prototype
- Java Code Skeletons

Analysis
- Analysis + Simulation

Execution + Measurement
- Completion + Compilation

Model-Driven Performance Analysis

Core Scenario Models

Domain Specific Modeling Languages (DSML)

Languages for representing different **views** of a system in terms of models

Higher-level **abstraction** than general purpose languages

Closer to the **problem domain** than to the implementation domain

Closer to the **domain experts**, allowing modelers to perceive themselves as working directly with domain concepts

Domain **rules can be included into the language** as constraints, disallowing the specification of illegal or incorrect models.
An example of a DSM

The design for a conference application intended to run on a Symbian/S60 phone.

Visual DSMLs

VDSMLs tend to offer substantial gains over conventional textual languages
- Formal studies show significant benefits for novices
- Increasing number of VDSMLs being defined

But not a panacea: every notation has advantages and disadvantages
- [Not the subject of this talk]

[Kirsten whitley “Visual programming languages and the empirical evidence for and against”, JVLC 1997]
[R. Navarro-Prieto, J. Cañas “Are visual programming languages better? The role of imagery in program comprehension”, IJHCS 2001]

[Thomas R.G. Green & Marian Petre]
“The impossible equation”

USA (estimates for 2012):
• 90M computer users
• 55M Spreadsheet & DB users
• 13M self-described as programmers
• 2.5M professional programmers

End-user Programming[Modeling]

- Most software creators are not software professionals
  - End users are participants and developers, not passive consumers
  - They do not reason about software like professionals
    - [Mary Shaw, *The Challenge of Pervasive Software to the Conventional Wisdom of Software Engineering, ESEC-FSE09*]

- End users are not “casual,” “novice” or “naive” users; they are people such as chemists, librarians, teachers, architects, and accountants, who have computational needs and want to make serious use of computers, but who are not interested in becoming professional programmers.
End-user (Visual) Modeling
Production systems
DSMLs are starting to proliferate

They allow users to model their systems at an appropriate level of abstraction

Some of them allow more than “documentation”
  - Code generation
  - Animation
  - Simulation
  - ...

Very few allow specification and analysis of the Quality Properties (NFPs) of modeled systems
  - QoS usage and management constraints: performance, reliability, resource consumption and allocation, etc.
How to conduct Performance Analysis on
> High-Level,
> Domain-specific,
> End-user defined
models?
Requirements for PA of DSM

Notations for describing systems must be:
- Simple and intuitive
- Close to the problem domain
- Close to the domain experts’ language

Models must be:
- Abstract, yet precise
- Executable (to, at least, prototype systems)

QoS notations must be:
- Simple and precise, yet expressive

Analysis results and feedback must be:
- Understandable and easy to manage
“Being abstract is something profoundly different from being vague... The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.”

Edsger Dijkstra
Current notations for DSMs

Notations for describing systems must be normally are:
- Simple and intuitive
- Close to the problem domain
- Close to the domain experts’ language
- General purpose

Models must be normally are:
- Abstract, yet precise
- Executable

QoS notations must be normally are:
- Simple and precise, yet expressive
- Too complex and low level (more than needed for most end-user DSMLs)

Analysis results and feedback must be normally are:
- Understandable and easy to manage
- Tough to deal with!
Once upon a time, there was a team leader that was going on holidays. Before leaving, she made the last recommendation to her small team of three young engineers: “For the ongoing project, do not start coding in Java before the UML model is completely finished and you all agree on the model.”

On the Monday morning, as soon as she left, one of the engineers told the others about a wonderful discovery he made while twittering in the weekend: a very powerful tool that generates UML diagrams from code. The decision was rapidly taken and all three started coding the problem in Java.

Some days before the end of the leader’s holidays, all the Java code was used to generate UML diagrams and both the code and the UML diagrams were handled to the group leader.

She was quite impressed about the level of detail of the UML model and the narrow correspondence between the code and the model.

[Borrowed from J. Bezivin]
A. Vallecillo: "Performance Analysis of Domain Specific Models"

ICPE 2011
The *precise* meaning of models
“There are only 10 types of people in the world: Those who understand binary, and those who don't”
Same model for different concepts

There are only 10 types of people in the world: Those who understand binary, and those who don't.

Your meaning goes here...
Different models for the same concept
What does this model means?

$[[M]]_{BPMN} \neq [[M]]_{UML}$

Exclusive semantics

Inclusive semantics
Current notations for expressing QoS

- Annotations to existing (OO) models
- Very detailed and precise
- Provide connections with analysis models and tools (QNM, SPN, SPA)
- Excellent for modeling systems at certain levels of abstraction

Slide courtesy of Sebastien Gerard, CEA-LETI

[Borrowed from Selic, 2007]
However...

At a lower level than needed for most end-user DSMLs
- Complex (to read, write and maintain)
- Tedious, error-prone
- General-purpose
- Object-oriented
However...

- At a lower level than needed for most end-user DSMLs
- Complex (to read, write and maintain)
- Tedious, error-prone
- General-purpose
- Object-oriented
- No “dynamic” management of QoS (contracts)
  - Negotiation of QoS?
  - Adaptation?
  - End-to-end QoS reqs?
Require complex quality models

Fig. 6. The QoS Model for the Speed Regulator example

[Borrowed from Espinoza et al.]
Easy-to-read specifications?

Notations to express QoS are strongly dependent on the notations used to express behavior.

(Probably this is why it gets so complex in UML?)

Fig. 7. Sequence Diagram for updateSpeed annotated with the UML profile for QoS & FT
Requirements on QoS specifications

QoS constraints should be **modular** enough to be attached to individual objects.

It should be possible to derive the QoS of a composition of objects from the QoS of its component objects.

The level of QoS should be **observable** so as to allow the development of monitoring applications.

Through observation, applications become “QoS-aware” and can operate a feedback control loop on the supporting computing or network resources.

QoS should be **guaranteed** at certain periods

The nature of the guarantees can range from deterministic “hard” real-time guarantees, through weaker probabilistic guarantees or “best-efforts” policies.

QoS should be **negotiable** so that, during the life of the system, some users can quit an application whereas others can appear with different needs.

The framework should be flexible enough to allow such QoS management policies as graceful degradation.
How can we specify DSMs?

How do we express in a precise and abstract manner:

- Structure
- Behavior
- Time-dependent functionality
- Quality properties (QoS,...)

Which is the best **notation** for each of those aspects?

- It depends on the **purpose** of the model...
- ...must have a **precise** meaning
- ...and must allow the **analysis** of the models
Each notation is more apt for a task

\[
\begin{align*}
MCMLXVII & + DLXXIX \\
\text{???} & + 579 \\
\hline
1.967
\end{align*}
\]
Each notation is more apt for a task

\[ \text{MCMLXVII} + \text{DLXXIX} = ? + 579 \]

\[ ? + 579 = 2.546 \]
Each notation is more apt for a task

\[
\begin{array}{c}
MCMLXVII \\
+ \quad DLXXIX \\
\checkmark \quad MMDXLVI
\end{array}
\]

\[
\begin{array}{c}
1.967 \\
+ \quad 579 \\
\checkmark \quad 2.546
\end{array}
\]
How do you solve this problem?

A 40-years-old man has a daughter and a son. If the difference of age between the kids is 4 years, and the sum of their ages is half of the age of the father, how old are they?

\[
\begin{align*}
    x - y &= 4 \\
    x + y &= 20 \\
    2x &= 24
\end{align*}
\]

\[
\begin{align*}
    x &= 12 \\
    y &= 8
\end{align*}
\]

Solution: the older is 12 and the younger is 8
An invariant through the history of mature disciplines is the search for notations that allow formulating problems in a language that allows their easy solution.

\[
\frac{\partial f}{\partial x_i}(a_1, \ldots, a_n) = \lim_{h \to 0} \frac{f(a_1, \ldots, a_i + h, \ldots, a_n) - f(a_1, \ldots, a_n)}{h}.
\]

\[
\ddot{F} = \frac{d}{dt}(mv) \quad \int_{-N}^{N} f(x) \, dx \quad \sum_{n=1}^{\infty} \frac{1}{n^2} \quad \square(p \rightarrow q) \rightarrow (\square p \rightarrow \square q).
\]

\[
\rho \left( \frac{\partial v}{\partial t} + v \cdot \nabla v \right) = -\nabla p + \nabla \cdot T + f,
\]

http://en.wikipedia.org/wiki/Temporal_logic
Our current software modeling notations
The UML way...
Sauron’s approach to metamodeling
(e.g., OMG’s UML metamodel)

The lord of the Metamodels
(obviously, adapted)

Three notations for the Structure modelers under the sky,
Seven for the Behavior modelers in their halls of stone,
Tree for mortal Packagers doomed to die,
**One for the Designer of the Whole system on his dark throne**
    In the Land of Mof where the shadows lie.

One Metamodel to rule them all, One Metamodel to find them,
One Metamodel to bring them all and in the darkness bind them
    In the Land of Mof where the shadows lie.
No general purpose language can express all different semantics without becoming a monster

Especially under the presence of antagonist semantics (Discrete & continuous; synchronous & asynchronous;...)

“More general does not mean Better. Heterogeneity may be better than generality.
...Useful semantics imply constraints on designers.”

Edward A. Lee
The village metaphor
Semantic (or “Meaningful”) Domains
(vs. “meaningless” Models)

The meaning of a model $M$ is defined by its interpretation in a meaningful semantic domain $D$.

Each Semantic Domain has
- Precise semantics
- A set of (equivalent) notations
- A set of analysis tools
- Underlying logic

Semantic Bridges connect Semantic Domains

- The Prolog village
- The QNM village
- The Petri Net village
- The Coq village
- The Process Alg village
- The Maude village
- The Z village
- The Modelica village
- The B village
- ...
Expressing semantic bridges

As Model Transformations

- Possible if correspondences can be expressed as functions
- Pairwise consistency can be formally studied

- One form of consistency involves a set of correspondence rules to steer a transformation from one language to another. Thus given a specification $S_1$ in viewpoint language $L_1$ and specification $S_2$ in viewpoint language $L_2$, a transformation $T$ can be applied to $S_1$ resulting in a new specification $T(S_1)$ in viewpoint language $L_2$ which can be compared directly to $S_2$ to check, for example, for behavioral compatibility between allegedly equivalent objects or configurations of objects [RM-ODP, Part 3]

As Weaving Models

- Possible if correspondences are just mappings
Semantic Mappings as Model Transformations

Types
- Domestic
- Horizontal
- Vertical
- Abstracting
- Refining
- Pruning
- Forgetful
- ...

The relationship between domains $D$ and $D'$ is defined by a model transformation $T:D\rightarrow D'$.

$$[[M]]_{D'} := [[T(M)]]_{D'}$$
How do we analyse models?

Crossing the bridges!!!
Some initial experiments...
A Production System Example

![Diagram of a production system example]
Behavioral semantics

Using in-place model transformations (Graph Transf.)

$l: [NAC] \times LHS \rightarrow RHS$
Some essential additions

**Time**
- Rule duration
- Periodicity, soft scheduling
- Ongoing rules
- Access to the Global Time Elapse
- Time stamps, scheduled actions

**Specification of action executions**
- Without the need of unnaturally modify the metamodel

**OCL** for attribute calculations and rule conditions

\[ l : [NAC] \times LHS \rightarrow RHS \]
Precise Semantics

Defined by a Semantic Mapping to Real-Time Maude

This makes models amenable to simulation and to formal analysis using the Real-Time toolkit!

Implementable (by a set of ATL model transformations)
Representing Models with Maude

ProductionSystem {
    < 'p : Plant | els : 'heg 'hag 'c1 'c2 't1 'a 'c3 't2 'u >
    < 'hag : HandleGen | in : null, out : 'c2, xPos : 1, yPos : 1 >
    < 'heg : HeadGen | in : null, out : 'c1, xPos : 1, yPos : 3 >
    < 'c1 : Conveyor | outParts : nil, out : 't1, xPos : 2, yPos : 3 >
    < 'c2 : Conveyor | outParts : nil, out : 't1, xPos : 2, yPos : 1 >
    < 't1 : Tray | parts : nil, capacity : 4, xPos : 3, yPos : 2 >
    < 'a : Assembler | in : 't1, out : 'c3, xPos : 4, yPos : 2 >
    < 'c3 : Conveyor | outParts : nil, out : 't2, xPos : 5, yPos : 2 >
    < 't2 : Tray | parts : nil, capacity : 4, xPos : 6, yPos : 2 >
    < 'u : User | parts : nil, xPos : 6, yPos : 3 >
}
Representing Metamodels with Maude

op ProductionSystem : -> @Metamodel.
op PS : -> @Package.

sort PositionedEl.
subsort PositionedEl < @Class.
op PositionedEl : -> PositionedEl.

op xPos : -> @Attribute.
op yPos : -> @Attribute.

sort Container.
subsort Container < PositionedEl.
op Container : -> Container.
op parts : -> @Reference.

op in : -> @Reference.
op out : -> @Reference.

...
Representing **Behavior** with Maude

\[
\text{rl [Transfer] : } \\\n\text{ProductionSystem} \{ \\\n\quad \langle p : \text{Part} | \text{xPos : XPOS, yPos : YPOS, SFS} \rangle \\\n\quad \langle c : \text{Conveyor} | \text{OutParts : (p PARTS), out : t, SFS'} \rangle \\\n\quad \langle t : \text{Tray} | \text{xPos : XPOS', yPos : YPOS', parts : PARTS', SFS''} \rangle \}
\]

\[
\Rightarrow \\\n\text{ProductionSystem}\{ \\\n\quad \langle p : \text{Part} | \text{xPos : XPOS', yPos : YPOS',SFS} \rangle \\\n\quad \langle c : \text{Conveyor} | \text{outParts : PARTS, out : t, SFS'} \rangle \\\n\quad \langle t : \text{Tray} | \text{xPos : XPOS', yPos : YPOS', parts : (p PARTS'), SFS''} \rangle \}
\]

A. Vallecillo: "Performance Analysis of Domain Specific Models"
Model Simulation and Analysis
[Simulation 2009]

- Simulation/Execution of specifications
  (tre<initModel in time <= 20 .)

- Reachability Analysis
  - Deadlock
    search initModel =>*
    ProductionSystem {
      < O : Tray | capacity : CAP, parts : PARTS, SFS >
    OBJSET }

  - Invariants
    \[\text{find earliest } \{\text{initModel}\} \Rightarrow^* \{\text{ProductionSystem}\} \{\text{< T : ActionExec | rule : "Collect", value : null, SFS@T > OBJSET }\} \).

  - Others

- LTL Model checking
  - Liveness properties
    (mc \{\text{initModel}\} \models \[\text{enssembled('he10.ha10) -> collected('he10.ha10) in time <= 100 .}])
Adding NFPs to DSMLs

A. Vallecillo: "Performance Analysis of Domain Specific Models"  
ICPE 2011
Adding NFPs to DSMLs

Use of “observers”

Adding NFPs to DSMLs

Observers capture the state of the NFPs and monitor their progress.

Making use of the Observers

The system can self-adapt under certain conditions
Making use of the Observers

With the new configuration, the system transmits sounds in a faster way.

\[
\text{SoundFlowFast} \quad (\text{\#\{m.xPos-sp.xPos, yPos\}})  \quad \text{abs'(\#\{m.xPos-sp.yPos\})} \quad \text{abs'} - 1/2 \quad \text{\text{\#\{\text{\textquoteleft\textquoteleft eager\textquoteright\textquoteright\}}}}
\]
A more complex example
A more complex example
Packet Processing
T in \([2 \cdot n.\text{packets} \rightarrow \text{size} + 1, 2 \cdot n.\text{packets} \rightarrow \text{size} + 1]\)

Activate Support
T in [0, 0]

Deactivate
T in [0, 0]
Analysing the results

http://atenea.lcc.uma.es/index.php/Main_Page/Resources/E-motions/PacketSwitchingExample/Results
Pros and Cons

Advantages
- Addition of observers independently of the system
- Simple modelling of QoS properties
- Ability to monitor QoS properties
- Results obtained in easy-to-manipulate format
- More expressive than other notations (SPA, SPN, QNM,...) (generalized distributions, OCL expressiveness, dynamic topologies, action executions as 1st class citizens,...)

Limitations
- Efficiency of simulations
- Difficult to debug
- Not for every problem or domain
- More expressive than other notations (SPA, SPN, QNM,...) (Difficulties for defining semantic bridges due to large gaps/chasms)

We are not alone...

**LIGHTWEIGHT MODELING**

**DEFINITION**
- constructing a very abstract model of the core concepts of a system
- using an analysis tool based on exhaustive enumeration to explore its properties

**WHAT IS ITS VALUE?**
- it is a design tool that reveals conceptual errors early
  
  decades of research on software engineering proves that the cost of fixing a bug rises exponentially with the delay in its discovery

- it is a documentation tool that provides complete, consistent, and unambiguous information to implementors and users

- it is easy (at least to get started) and surprising (you get the result of scenarios you would **never** expect)
  
  "If you like surprises, you will love lightweight modeling."
  —Pamela Zave

- EASY + SURPRISING = FUN

**WHY IS IT "LIGHTWEIGHT"?**
- because the model is very abstract in comparison to a real implementation, and focuses only on core concepts, it is small and can be constructed quickly

- because the analysis tool is "push-button", it yields results with little effort

  *in contrast, theorem proving is not "push-button"*

---

A. Vallecillo: "Performance Analysis of Domain Specific Models"  
ICPE 2011
One challenge for software engineers now is to provide **end-users** with **Modeling Languages** (and associated tools) that allow them to model their systems in a cheap, quick and useful way, and to analyse them using **push-button** approaches.

Current widely-used general-purpose modeling notations (especially behavioural and QoS) do not seem to be really up to the job.

*Integrating heterogeneous notations and their associated tools using model transformations seems to be one promising way to go.*
Challenges

- Definition of new languages for behavioural descriptions, which allow easy specification of Quality Properties and their analysis

- Improved languages for QoS specification

- Semantic bridges to other domains
  - Better connection with analysis tools

- Improved traceability mechanisms
  - Improve understandability of results

- Better feedback to users
  - E.g., Performance anti-patterns (!)
Thanks!

Acknowledgements: