Including CORBA performance details into MDA System models

Tom Verdickt, Bart Dhoedt, Frank Gielen, Piet Demeester
Overview

- Software Performance Engineering
- SPE + MDA
- Middleware
- Transformation
- Conclusion
Traditional software design

Performance requirements

Functional requirements

design

Architecture

coding

Working program

simulation

Performance parameters

Fine-tuning

Finished program
Problem

- Traditional software design: functionality
- Performance only checked in finished product

Result:
- changes and fine-tuning in final stages
  - expensive (over budget)
  - time-consuming (missed deadlines)
- poor performance in final system (bad software)

- Software quality \(\approx\) performance
  - many projects fail because of poor performance
    - 1/3 fails completely
    - 40% too late or over budget
    - many of those because of poor performance
Software Performance Engineering (1)

- Performance analysis during complete design, starting as early as possible
- Methods:
  - performance models (queueing networks, Petri nets)
  - quantitative methods (MVA) and simulations
- Build performance into design (don’t try to add it later)
- Current status: accurate modeling of non-distributed systems
Software Performance Engineering (2)

- Performance requirements
- Functional requirements
- Design
- Architecture
- Coding
- Performance estimates
- Refined estimates
- Working program
- Simulation
- Performance parameters
- Fine-tuning
- Finished program
• Performance models use dedicated modelling languages
  - Designers need to learn a new modelling language
  - Extra models needed => time-consuming
  - Synchronize changes
• General-purpose modelling languages lack performance features
• Solutions:
  - performance extensions (e.g. UML profiles)
  - automatic transformations
• Software Performance Engineering
• SPE + MDA
• Middleware
• Transformation
• Conclusion
SPE + MDA

• Many performance details needed
  - platform
  - middleware
  - third-party components
  - etc.

• Goal: (semi-) automatic performance evaluation

• MDA/UML provides necessary features:
  - solid modelling formalism
  - different abstraction levels
  - include component models
  - model refinement

\{ PIM-to-PSM transformation \}
Ideal performance modelling process

High-level (UML) model

Component information

Low-level (UML) model

performance model

performance estimations
Overview

• Software Performance Engineering
• SPE + MDA
• Middleware
• Transformation
• Conclusion
Growing system complexity

- Single computer
Growing system complexity

- Single computer
- Distributed system
Growing system complexity

- Single computer
- Distributed system
- Distributed system using middleware
• Goal: provide interoperability between various components of a distributed system
• Examples: CORBA, JINI, RMI
• CORBA benefits:
  - independence from programming language, architecture, platform
  - event handling
  - location transparency (naming service)
CORBA

naming service

client

ORB interface

stub

client ORB

network

server

ORB interface

skeleton

server ORB
Overview

• Software Performance Engineering
• SPE + MDA
• Middleware
• Transformation
• Conclusion
• High-level PSM, using UML
  - collaboration diagram
  - deployment diagram
  - activity diagram

• Description of middleware details
  - type (e.g. CORBA)
  - performance information
  - deployment

• Naming convention to link the different models together

• A library of detailed middleware models (can be included implicitly in the transformation algorithm)
• Low-level PSM, using UML
  - several collaboration diagrams
  - deployment diagram
  - activity diagram
• Diagrams contain performance information using the UML Profile for schedulability, performance and time
Transformation: component names

<middleware>
  <link id="link1" type="CORBA" NSref="NS1">  
    <call cref="G.9" />
  </link>

  <NS id="NS1" host="xmi.17" />
</middleware>
Transformation: deployment

<middleware>
  <link id="link1" type="CORBA" NSref="NS1">
    <call cref="G.9" />
  </link>

  <NS id="NS1" host="xmi.17" />
</middleware>
Transformation: deployment

```xml
<middleware>
  <link id="link1" type="CORBA" NSref="NS1">
    <call cref="G.9"/>
  </link>
  <NS id="NS1" host="xmi.17"/>
</middleware>
```
Transformation: collaboration

```
<middleware>
  <link id="link1" type="CORBA" NSref="NS1">  
    <call cref="G.9" />  
  </link>
</middleware>
```

```
<NS id="NS1" host="xmi.17" />  
</middleware>
```
Transformation: collaboration

```
<middleware>
  <link id="link1" type="CORBA" NSref="NS1">
    <call cref="G.9" />
  </link>
  <NS id="NS1" host="xmi.17" />
</middleware>
```
Transformation: activity, input

```xml
<middleware>
  <link id="link1" type="CORBA" NSref="NS1">
    <call cref="G.9" />
  </link>

  <NS id="NS1" host="xmi.17" />
</middleware>
```
• initialization
• get server reference (Naming Service)
• call server
  - stub
  - skeleton
• destroy orb and clean up
Transformation: activity, initialization

- client
- corba_client
  - call_init
  - initialize
  - call_resolve
  - call_client
  - request
- orb
- naming context
  - do_resolve
  - resolve
  - reply
- NS
Transformation: activity, call

Diagram:

- Client request
- Stub request
- Stub reply
- Dummy
- Skeleton request
- Skeleton reply
- Server waiting
- Process
- Undefined
Transformation: activity, destroy

- client
  - dummy
  - continue
- corba_client
  - call_destroy
  - return_client
- orb
  - destroy
Transformation: performance

- Performance modelling using performance profile
- Actions get the `<<<Pastep>>` stereotype
- Execution times: Pademand tagged value

```xml
<middleware>
  <link id="link1" type="CORBA" NSref="NS1" inittime="36.55"
nscalltime="0.45" destroytime="3.36">
    <call cref="G.9" stubtime="1.84" skeletonstime="0.10" />
  </link>
</middleware>
```
## Transformation: activity final

<table>
<thead>
<tr>
<th>client</th>
<th>corba_client</th>
<th>orb</th>
<th>naming context</th>
<th>NS</th>
<th>stub</th>
<th>skeleton</th>
<th>server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>waiting</td>
</tr>
<tr>
<td></td>
<td>call_init</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>initialize</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>call_resolve</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>request</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>call_client</td>
<td></td>
<td>stub_request</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>do_resolve</td>
<td></td>
<td></td>
<td>skeleton_reply</td>
<td>undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>resolve</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>reply</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dummy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>call_destroy</td>
<td></td>
<td>destroy</td>
<td></td>
<td></td>
<td>skeleton_reply</td>
<td>undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>continue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<<PStep>> {PADemand="('assm', 'mean', (36.549, 'ms'))"}

<<PStep>> {PADemand="('assm', 'mean', (0.4597, 'ms'))"}

<<PStep>> {PADemand="('assm', 'mean', (0.53, 'ms'))"}

<<PStep>> {PADemand="('assm', 'mean', (1.8413, 'ms'))"}

<<PStep>> {PADemand="('assm', 'mean', (0.1021, 'ms'))"}

<<PStep>> {PADemand="('assm', 'mean', (3.3646, 'ms'))"}

<<PStep>> {PADemand="('assm', 'mean', (0.53, 'ms'))"}

<<PStep>> {PADemand="('assm', 'mean', (1.8413, 'ms'))"}

<<PStep>> {PADemand="('assm', 'mean', (0.1021, 'ms'))"}

<<PStep>> {PADemand="('assm', 'mean', (3.3646, 'ms'))"}
The way forward
Test results: response time

![Graph showing response time vs number of clients]

- **Response time**
- **Number of clients**
- **Model**
- **Measurements**
Overview

• Software Performance Engineering
• SPE + MDA
• Middleware
• Transformation
• Conclusion
• Automatic inclusion of CORBA performance information during PIM-to-PSM transformation
• Allows early assessment of the performance impact of using CORBA, without knowledge of the CORBA internals
• Easily adaptable to other middleware types (allowing the designers to compare several types)