Motivations

- Large software systems are complex and expensive artifacts
- Their success depends on whether they deliver a satisfactory performance level
- Two questions:
  - How to evaluate performances during the software development process?
  - When are performances to be evaluated?

How can performances be evaluated?

- Measurement-based approach
  - Perform direct measures on a running system or a prototype; use these measurements to identify bottlenecks
- Model-based approach
  - Develop a performance model of the software system; use the model to mimic the behavior of the system and predict its performances
Where performances can be evaluated?

- Requirements Definition
- System and Software design
- Model-based Performance Evaluation
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- Measurement-based Performance Evaluation

Model-Based Performance Evaluation

**Pro**
- Does not require a running system
- Hence, can be applied from the early design stages

**Cons:**
- Accuracy of the performance prediction depends on the accuracy of the software model (100% accuracy not needed/reasonable anyway)
- May be difficult to report feedback due to different structure of the performance model wrt the software model

General Software Performance Modeling

- Software Model
- Performance Model
- Model Evaluation
- Results and Feedback

Background / 1

**SPE approach by Smith**
- C. U. Smith, Performance Engineering of Software Systems, Addison-Wesley 1990

**Many approaches deriving analytical models**
Background / 2

• Some works deriving simulation models
  • P. Kähkipuro. UML-based performance modeling framework for component-based distributed systems. Performance Engineering, vol 2047 of LNCS.
  • M. De Miguel et al. UML extensions for the specifications and evaluation of latency constraints in architectural models. Proc. WOSP 2000
  • A. Hennig et al. Performance prototyping - generating and simulating a distributed IT-system from UML models. Proc. ESM'03

Critique

• Non-standard software description notations
• Analytical modeling techniques:
  Often hard/impossible to solve analytically
  Complex mapping between sw model and perf model
  Difficult to report feedback at the SA level
• Simulation modeling techniques:
  Limited applicability
  Special-purpose

The challenge

• Develop an approach for performance evaluation of Software Architectures
  Based on standard technologies and notations, when available and appropriate
  Easy to use for users with little or no performance analysis background
  Applicable from early design stages

Our contribution / 1

• We choose UML as a software specification notation
  De facto standard for SA description, widely used in the Software Engineering community
  Well supported by CASE tools
  Extension mechanisms can be used to extend the UML metamodel
  UML Profile for Schedulability, Performance and Time specification released in 2002 as an OMG standard
Our contribution / 2

• We develop a UML profile for specifying performance-oriented annotations on UML models
  Based on the UML Performance Profile, with some modifications
  Different way to model workloads
  We modified the performance model in some points

Our contribution / 3

• We define a process-oriented simulation model of UML SA
  General modeling technique
  Allows unconstrained model representation (fork/join, simultaneous resource possession, general scheduling policies, general time distributions...)
  Simulation model has the same structure of the SA model...
  ...and so reporting feedback is very easy

Our contribution / 4

• We develop a prototype tool (UML-)
  Accepts XMI representations of annotated UML models produced by the ArgoUML CASE Tool
  Automatically derives the simulation model
  Creates a simulation program using a C++ process-oriented simulation library
  Executes the simulation and returns the results into the UML model as tagged values
• We extend the approach to unified UML-based performance and mobility modeling

The approach / 1

- Software Model
- UML Model
- Performance Results
- Simulation Model
- Simulation Program
- Feedback
- Model Impl.
- Modeling Algorithm
The approach / 2

Introduction to UML

- UML is a graphical modeling notation
- Widely used to describe Object-Oriented software systems
- Informally specified
- Provides several types of diagrams
  - Use Case
  - Deployment
  - Activity
  - State
  - Class / Package
  - Collaboration
  - Sequence

Use Case Diagram

- Contains Actors and Use Cases
  - Actors represent entities (physical or logical) which may interact with the system
  - Use Cases represent usage scenarios of the system

- Bank Customer
- Generic Customer
- Withdraw Cash
- Check Balance

Activity Diagram

- Describe the computations performed by elements of the system as a set of activities
- Flow-chart like notation

- Insert Card
- Insert PIN
- Validate PIN
- Check Card
- Operate
- OK
- Invalid
- Eat Card
**Deployment Diagram**

- Describe the resources available in the system

```plaintext
ATM 1  
    ▼
    |  
    |  
WAN

ATM 2

DB Server
```

---

**The approach**

1. Software Model
2. UML Model
3. Simulation Model
4. Simulation Impl.
5. Performance Results
6. Feedback

**Modeling Workloads / 1**

```plaintext
<<OpenWorkload>>
PAoccurrence = ["unbounded", "exponential", 20.0]

User

PAprob = 0.2
Use Case 1

PAprob = 0.8
Use Case 2
```

**Modeling Workloads / 2**

```plaintext
<<ClosedWorkload>>
PApopulation = 10
PAdelay = ["uniform", 10.0, 15.0]

User

PAprob = 0.2
Use Case 1

PAprob = 0.8
Use Case 2
```
Modeling Scenarios / 1

• Simple Actions

<<PStep>>
PArep = 5
PAinterval = ["assm", "dist", ["exponential", 0.1]]
PAdemand = ["msrd", "dist", ["exponential", 0.2]]
PAhost = "Workstation"
PAdelay = ["msrd", "dist", ["constant", 0.1]]
PArespTime = @xx@

An Action

Resource Acquire/Release

<<GRAcquire>> / <<GRLrelease>>
PAresource = "Memory"
PAquantity = ["assm", "dist", ["constant", 2]]

Modeling Resources / 1

• Active Resource (processor)

<<PAhost>>
PAutilization = @xx@
PAthroughput = @xx@
PAschedPolicy = "FIFO"
PActxSWT = ["assm", "dist", ["constant", 0.1]]
PArate = 2.0

Workstation

Modeling Resources / 2

• Passive Resource

<<PAresource>>
PAutilization = @xx@
PAthroughput = @xx@
PAcapacity = 100
PAaxTime = ["assm", "dist", ["uniform", 10.0, 20.0]]

Memory
The approach

Software Model

UML Model

Modeling Algorithm

Simulation Model

Model Impl.

Performance Results

Feedback

Simulation Program

The Simulation Performance Model

Performance Model

Workload

Scenario

Resource

Open Workload

Closed Workload

Activity

Acquire

Release

Simple

Active

Passive

Performance Model in UML

Model Generation
The Modeling Algorithm

- For each Use Case diagram $U$
  - Make a simulation process of type $\text{Open\_Workload}$ or $\text{Closed\_Workload}$, depending on the stereotype of $U$
- For each Action $A$
  - Make a simulation process of the appropriate type $(\text{Simple\_Action}, \text{Acquire\_Action}, \text{Release\_Action})$ according with the stereotype of $A$
  - Link the action in the same predecessor-successor relationship as in the UML model
- For each Deployment diagram node $N$
  - Make a simulation process of type $\text{Active\_Resource}$ or $\text{Passive\_Resource}$, depending on the stereotype of $N$

Translating Open Workloads

```
process OpenWorkload( Actor A )
var
  u : OpenWorkloadUser;
begin
  while ( true ) do begin
    hold( A.PAoccurrence );
    u := new( OpenWorkloadUser( A ) );
    activate( u );
  end;
end;
```

Translating Closed Workloads

```
process ClosedWorkload( A: Actor )
var
  i : integer;
  u : ClosedWorkloadUser;
begin
  for i:=1 to A.pop do begin
    u := new( ClosedWorkloadUser( A ) );
    activate( u );
  end;
end;
```

Translating Simple Actions

```
process SimpleAction( A: Action )
var
  i : integer;
  next : ActionBase;
begin
  while ( true ) do begin
    hold( A.PAprepare );
    for i:=1 to A.PArep do begin
      if ( i < A.PArepl )
        hold( A.PAinterval );
      else
        hold( A.PAprepare );
      end;
      if ( i < A.PArepl )
        hold( A.PAinterval );
      else
        hold( A.PAprepare );
      end;
      uc := new( CompositeAction( UC[i] ) );
      activate( uc );
      { Wait for the next execution }
      passivate( );
    end;
end;
```

Translating Simple Actions (continued)

```
process SimpleAction( A: Action )
var
  i : integer;
  next : ActionBase;
begin
  while ( true ) do begin
    hold( A.PAprepare );
    for i:=1 to A.PArep do begin
      if ( i < A.PArepl )
        hold( A.PAinterval );
      else
        hold( A.PAprepare );
      end;
      if ( i < A.PArepl )
        hold( A.PAinterval );
      else
        hold( A.PAprepare );
      end;
      uc := new( CompositeAction( UC[i] ) );
      activate( uc );
      { Wait for the next execution }
      passivate( );
    end;
end;
```
Translating Active Resources

```
<<.Mult>>
PAutilization = @x@ PAwaitingInput = @x@ PAscheduledPolicy = "FIFO" PAActSwT = \"constant\", 0.1\nPArate = 2.0
```

The Simulation Program

- Based on a process-oriented discrete-event simulation library in C++
- The library provides basic (simulation) process scheduling facilities (SIMULA-like)
- Basic statistical functions for output data analysis are available
  Mean, standard deviation, quantile, histogram
- Initialization bias removal

The approach

```
process ActiveResource( N ModelInstance )
    var j : Action;
    jobq : queue of Action; ( Queue of jobs )
    begin
        while ( true ) do begin
            while ( not jobq.empty() ) do begin
                j := jobq.first();
                jobq.remove_first();
                hold j.Pdemand / N.PRate;
                activate j;
            end;
            passivate j;
        end;
end;
```

UML-

- Written in C++
- Parses the XMI representations of annotated UML models produced by ArgoUML
- Tag values can be written in Perl
- Automatically derives the simulation model
- Creates the simulation program using a C++ process-oriented simulation library
- Executes the simulation and returns the results into the UML model as tagged values
Modeling in action

Performance Results

The approach

Mobility and Performance Modeling with UML
Conclusions

- We described how performance analysis can be done at the SA design level
- A UML profile has been defined for adding quantitative annotations to UML models
- We define a process-oriented simulation model of a SA
- A prototype tool (UML-) has been developed
  - Parses annotated UML diagrams saved in XMI format
  - Generates simulation model
  - Executes simulation and reports feedback

Ongoing Work

- **UML-related improvements**
  - Include more types of UML diagrams in the software model, such as State Charts and Sequence diagrams
- **Simulation-related improvements**
  - Compute more performance indices
- **Further improvements**
  - Integrate the approach into a general framework including different kinds of functional and non functional analyses

Relevant publications


All available on http://www.dsi.unive.it/~marzolla/publications.html