Talk Outline

• Motivations and General Principles
• Contribution of this work
• Introduction to UML
• The Software Performance Model
• The Simulation Model
• Conclusions
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 performances can 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
- Measurement-based Performance Evaluation
- Integration and system testing
- Operation and maintenance
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
- Results and Feedback
- Model Evaluation

Feedback and Evaluation
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


  • 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
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...)
  - Model derivation is very easy...
  - ...and so reporting feedback
Our contribution / 4

• We develop a prototype tool (UML-Ψ)
  ♦ 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

Software Model → UML Model → Performance Results → Simulation Model

Feedback → Modeling Algorithm

UML-Ψ

Simulation Program → Model Impl.
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

![Use Case Diagram](Image)

- 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

Diagram:

1. Insert Card
2. Insert PIN
3. Validate PIN
4. Check Card
5. Operate
6. Eat Card
7. OK
8. Invalid
Deployment Diagram

- Describe the **resources** available in the system

![Deployment Diagram]

- ATM 1
- ATM 2
- WAN
- DB Server
The approach

- Software Model
- UML Model
  - Modeling Algorithm
  - Feedback
- Performance Results
- Simulation Model
  - Model Impl.
- Simulation Program
Modeling Workloads / 1

<<OpenWorkload>>

PAoccurrence =
[“unbounded”,
[“exponential”,20.0]]

User

PAprob = 0.2
Use Case 1

PAprob = 0.8
Use Case 2
<<ClosedWorkload>>

PApopulation = 10
PAextDelay = ["uniform", 10.0, 15.0]

User

PApob = 0.2
Use Case 1

PApob = 0.8
Use Case 2
Modeling Scenarios / 1

- Simple Actions

```
<<PAstep>>

PArep = 5
PAinterval = ["assm", "dist", ["exponential", 0.1]]
PAdemand = ["msrd", "dist", ["exponential", 0.2]]
PAhost = "Workstation"
PAdelay = ["msrd", "dist", ["constant", 0.1]]
PArespTime = @xx@
```
Modeling Scenarios / 2

• Resource Acquire/Release

<<GRMacquire>> / <<GRMrelease>>
PAresource = “Memory”
PAquantity = [“assm”, ”dist”, [“constant”, 2]]
Modeling Resources / 1

• Active Resource (processor)

<<P主持>>

PA利用化 = @xx@
P平均通过量 = @xx@
P调度策略 = “FIFO”
PContextSwT = (“constant”, 0.1)
P到达率 = 2.0
• Passive Resource

```plaintext
<<PAresource>>
PAuthentication = @xx@
PAThroughput = @xx@
PACapacity = 100
PAaxTime = ["uniform", 10.0, 20.0]
```
The approach

- Software Model
- UML Model
- Performance Results
- Simulation Model
- Simulation Program

Modeling Algorithm
Feedback
Model Impl.
The Performance Model

- Workloads
  - Open Workload
  - Closed Workload

- Scenarios

- Resources
  - Active
  - Passive

- Activities
  - Acquire
  - Release
  - Simple
Performance Model in UML

- **PerformancContext**
  - +parameters
  - +simulation duration
  - +relative width

- **Workloads**
  - **Workload**
    - ClosedWorkload
      - +population
      - +extDelay
    - OpenWorkload
      - +occurrence

- **Scenarios**
  - 1..*

- **Resources**
  - 0..*

- **Actions**
  - **CompositeAction**
    - root
    - steps
  - **ActionBase**
    - +repetitions
    - +interval
    - +responseTime
    - +source
    - +target
    - +successors
  - **JoinAction**
  - **ForkAction**
  - **SimpleAction**
    - +demand
    - +delay
  - **ResActionBase**
    - +quantity
  - **Transition**
    - +probability
  - **AcquireAction**
  - **ReleaseAction**

- **ActiveResource**
  - +responseTime
  - +schedPolicy
  - +rate
  - +ctxSwTime

- **PassiveResource**
  - +capacity
  - +accessTime
Model Generation

**UML Model**
- Use Case Diagrams
- Activity Diagrams
- Deployment Diagrams
- Tagged Values

**Performance Model**
- Workloads
- Processing Steps
- Resources
- Model Parameters
- Simulation Results
The Modeling Algorithm

- For each Use Case diagram $U$
  - Make a simulation process of type `Open_Workload` or `Closed_Workload`, depending on the stereotype of $U$

- For each Action $A$
  - Make a simulation process of the appropriate type (`Simple_Action`, `Acquire_Action`, `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 `Active_Resource` or `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;

process OpenWorkloadUser( Actor A );
var
  i  : integer;
  uc : UseCase;
begin
  { choose i in [1..n] with probability [A.p[1]..A.p[n]] }
  uc := new( CompositeAction( UC[i] ) );
  activate( uc );
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;

process ClosedWorkloadUser( A: Actor )
var
   i : integer;
   uc : UseCase;
begin
   while ( true ) do begin
      hold( A.PAextDelay );
      { choose i in [1..n] with probability [ A.p[1]..A.p[n] ] }
      uc := new( CompositeAction( UC[i] ) );
      activate( uc );
      { Wait for uc to terminate }
   end;
end;

<<ClosedWorkload>>
PApopulation = 10
PAextDelay = ["uniform",10.0,15.0]

User

Use Case 1

Use Case 2

PApob = 0.2
PApob = 0.8
Translating Simple Actions

```pascal
process SimpleAction( A: Action )
var
  i    : integer;
  next : ActionBase;
begin
  while ( true ) do begin
    hold( A.PAdelay );
    for i:=1 to A.PArep do begin
      if ( A.PAhost <> nil ) then
        { Request service to A.PAhost }
      else
        hold( A.PAdemand );
      if ( i < A.PArep )
        hold( A.PAinterval );
    end;
    { Activate next step }
    next := choose_next_action( );
    activate( next );
    { Wait for the next execution }
    passivate( );
  end;
end;
```

An Action

```
<<PAstep>>
PArEp = 5
PAinterval = ["assm","dist", ["exponential", 0.1]]
PAdemand = ["msrd","dist", ["exponential", 0.2]]
PAhost = "Workstation"
PAdelay = ["msrd","dist", ["constant",0.1]]
PArEpTime = @x@```

M. Marzolla Simulation Modeling of UML Software Architectures
Translating Active Resources

process ActiveResource( N: NodeInstance )
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.Pademand / N.PArate );
      activate( j );
    end;
    passivate( );
  end;
end;

<<Pghost>>
PAutilization = @xx@
PAthroughput = @xx@
PAshedPolicy = “FIFO”
PActxSwT = [“constant”, 0.1]
PArate = 2.0
The approach

Software Model

UML Model

Performance Results

Simulation Program

Simulation Model

Model Impl.

Feedback

Modeling Algorithm
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
UML-$\Psi$

- 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
The approach

Software Model

UML Model

Simulation Model

Simulation Program

Performance Results

Modeling Algorithm

Feedback

Model Impl.
Performance Results

An Action

<<PAstep>>

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

<<PAhost>>

PAutilization = @xx@
PAthroughput = @xx@
PAschedPolicy = “FIFO”
PActxSwT = ["constant", 0.1]
PARate = 2.0

Workstation
Mobility and Performance Modeling with UML

[Diagram showing mobility and performance modeling with UML, including scenarios and transitions.]
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-Ψ) has been developed
  - Parses annotated UML diagrams saved in XMI format
  - Generates simulation model
  - Executes simulation and reports feedback
Open Problems

• 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
Related publications


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