Simulation Modeling of UML Software Architectures

Moreno Marzolla

Dipartimento di Informatica
Università “Ca' Foscari” di Venezia
marzolla@dsi.unive.it

Talk Outline

- Motivations and General Principles
- Contribution
- The proposed approach
- Conclusions

Motivations

- Large software systems are complex and expensive artifacts
- Their success depends on whether their functional and non-functional requirements are satisfied
- Performance is one of the most important non-functional requirements
- Two questions:

  How to evaluate performance during the software development process?

  When are performance to be evaluated?

How can performances be evaluated?

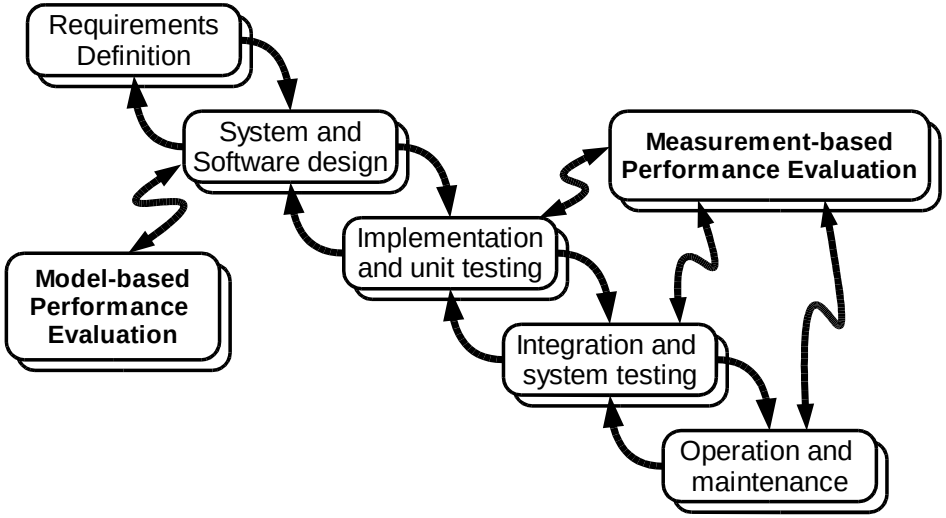
- ***Measurement-based approach***

  Perform direct measures on a running system or a prototype; use these measurements to identify bottlenecks

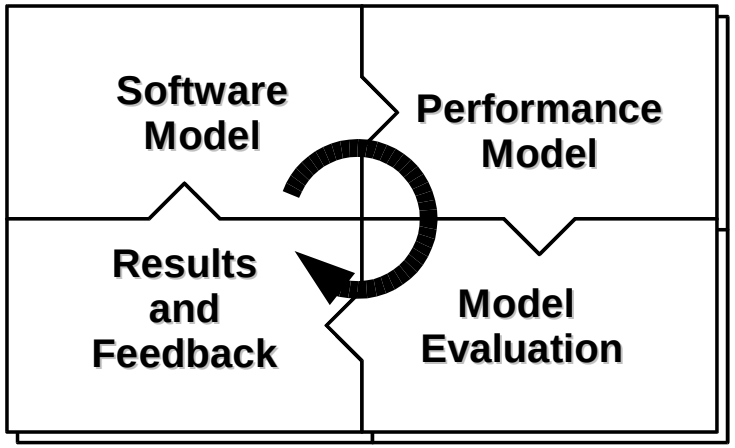
- ***Model-based approach***

  Develope 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?



General Software Performance Modeling



Background

- Many software performance evaluation approaches have been proposed
  - Analytical
  - Simulation-based (only a few)
- They use different software models
  - LTS, Process Algebras (PA), Petri Nets (PN), Message Sequence Charts, UML...
- Many performance models have been considered
  - Queuing Networks, SPN, SPA...

Limitations of current approaches

- Software models
  - Many approaches use software models described using non-standard, ad-hoc notations
- Performance models
  - Analytical modeling techniques
    - Can be very complex to analyze
    - Require special assumptions on the model
    - There may be a complex mapping between software and performance model, making it difficult to report feedback at the software level
  - Simulation modeling techniques
    - Special-purpose, make use of a limited software model

The challenge

- Develop an approach for performance evaluation of Software Architectures with the following characteristics:
  - Easy to use, possibly integrated with CASE tools
  - Based on standard notations, when available and appropriate
  - Easy to use for software architects with limited performance analysis skills
  - Can be applied from early design stages
  - Performance results should be easy to interpret at the Software Architectural design level

The contribution

- We propose an approach for performance evaluation of software systems described at a high level of abstraction
  - We use a notation (UML) for the description of the software system
  - We define a (simulation-based) performance model
  - We propose an algorithm for translating software specifications into performance models which allows feedback to be easily reported

Selection of the software model notation / 1

- Many different notations have been proposed
  - Some are only used for special application domains
- We choose UML
  - De facto standard for Software Architecture description, widely used in the Software Engineering community
  - Well supported by CASE tools
- UML alone not enough
  - Does not specify quantitative informations necessary for performance evaluation
  - Lacks a clear notion of time
  - Only informally specified

Selection of the software model notation / 2

- The *UML metamodel* can be extended using a standard mechanism based on *profiles*
  - Profiles can be used to specify additional semantic informations of particular UML elements
- We use the *UML Profile for Schedulability, Performance and Time specification* (OMG standard, 2002) with slight modifications

Choosing the performance model

- We define a process-oriented simulation model of UML Software Architectures
  - Simulationis a very general modeling technique
  - The simulation model has the same structure of the SA model, so reporting feedback is very easy
- We propose an algorithm for automatic derivation of a process-oriented simulation model from UML software specifications

The modeling algorithm

- The modeling algorithm translates an XML-based representation of annotated UML diagrams into a simulation model
  - The simulation model is process-oriented
- The simulation model makes use of a suitable simulation environment in order to be implemented and executed

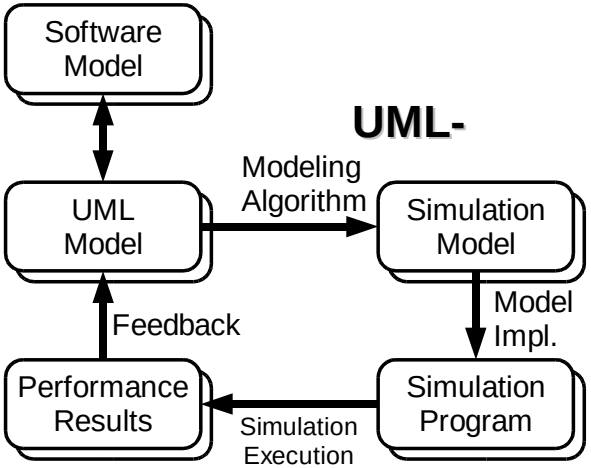
Extension of the approach

- We extended the approach to describe and evaluate performances of mobile systems
  - Both physical and logical mobility is considered
  - Mobility is described in standard UML notation

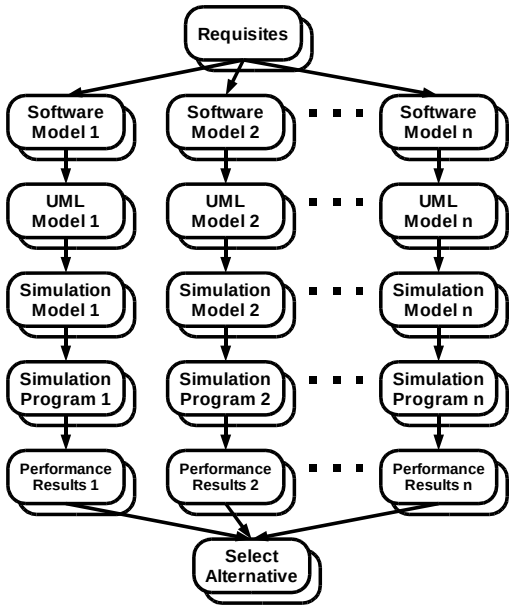
Implementing the approach

- We develope 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

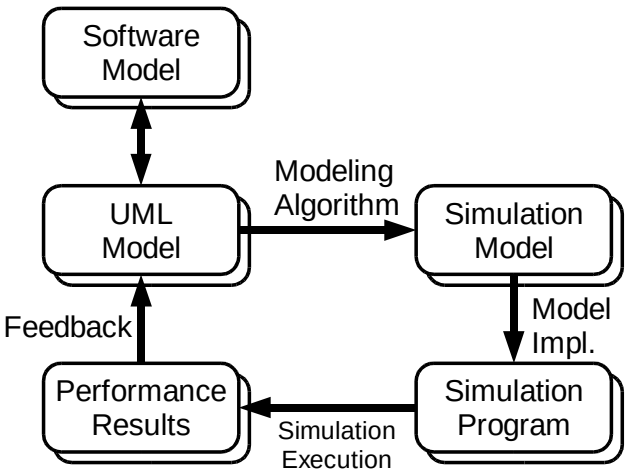
The approach / 1



The approach / 2



The approach

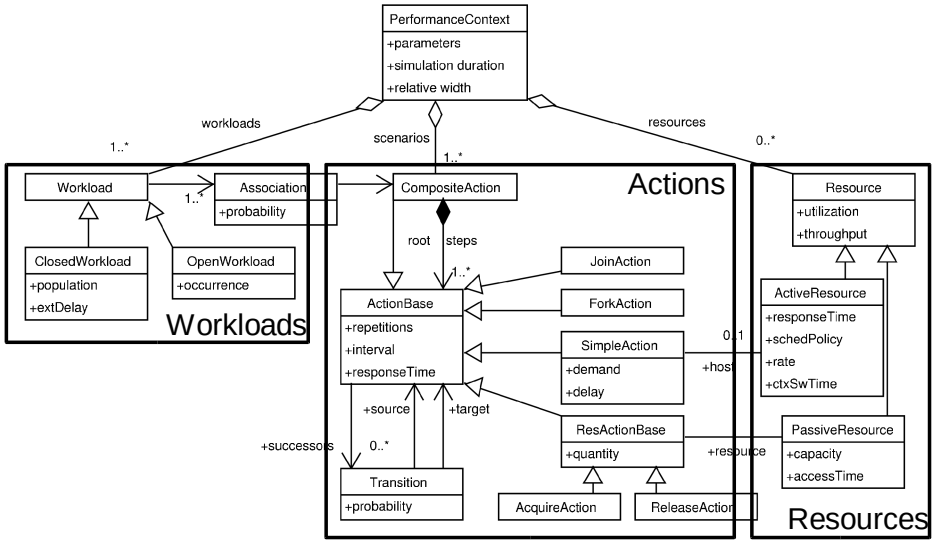


The UML software model

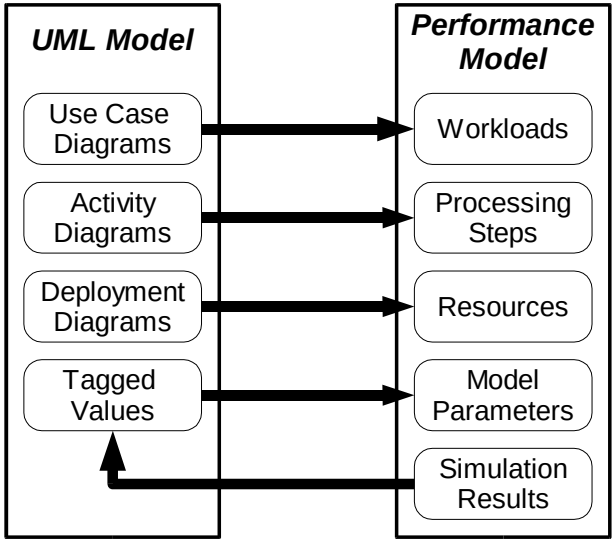
- UML is a graphical modeling notation, widely used to describe Object-Oriented software systems
- Provides several types of diagrams

Use Case	Class / Package
Deployment	Collaboration
Activity	Sequence
State	

The Performance Model of the UML Profile



Model Generation



Problems with UML

- UML is informally specified
- How can any performance model be derived from “informal” SW models?

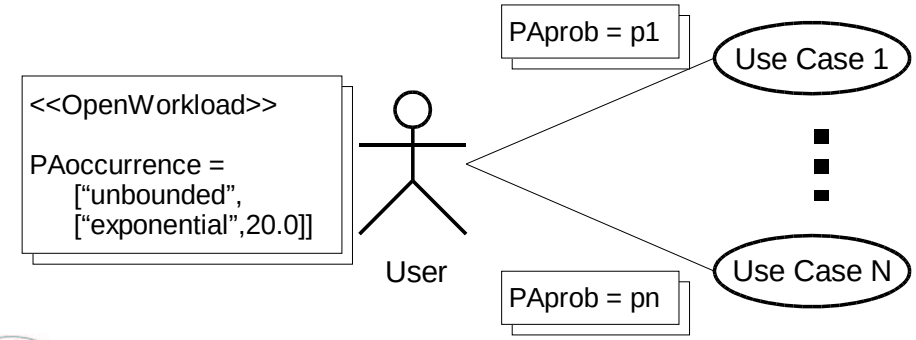
  We conform as much as possible with the informal, intuitive semantics of UML diagrams

  The mapping between SW model and simulation model actually enforces an operational semantics on annotated UML diagrams

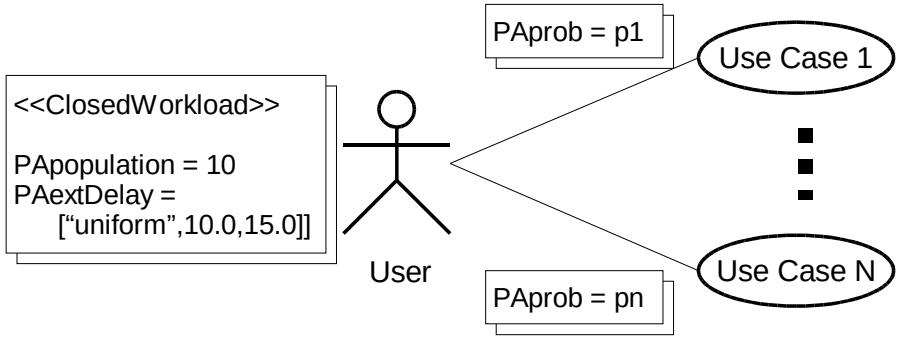
  This semantics is compatible with the intuitive one
- UML 2.0 is being defined more formally

Modeling Workloads / 1

- Workloads are modeled by the UML Profile by stereotyping the first stimulus of an interaction as *<<OpenWorkload>>* or *<<ClosedWorkload>>*
- We chose a different approach, and model workloads using Use Case diagrams

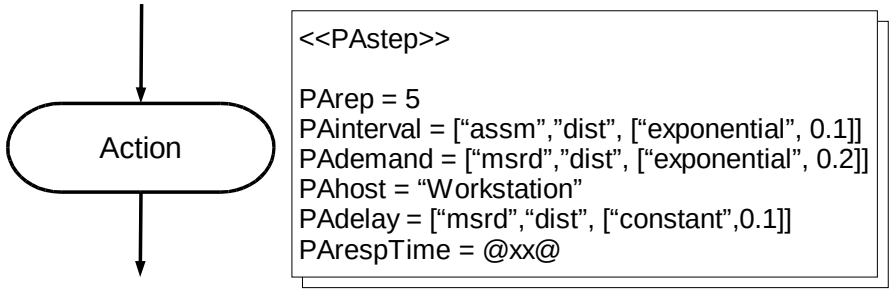


Modeling Workloads / 2



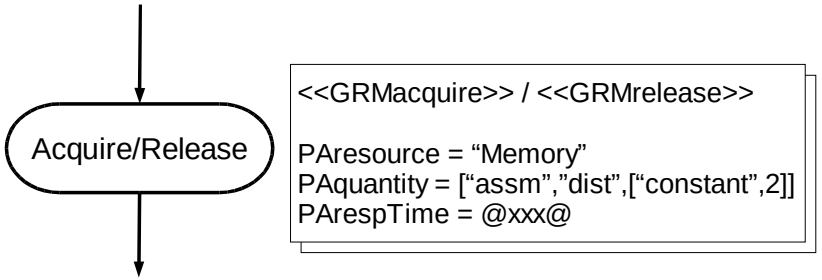
Modeling Scenarios / 1

- Simple Actions



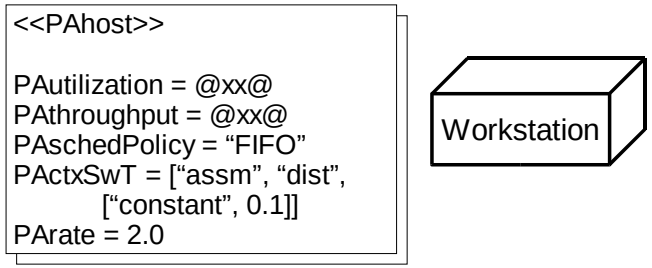
Modeling Scenarios / 2

- Resource Acquire/Release



Modeling Resources / 1

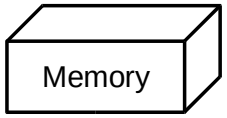
- Active Resource (processor)



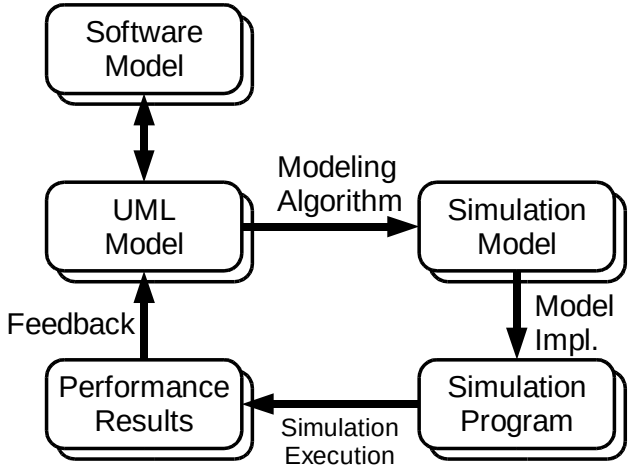
Modeling Resources / 2

- Passive Resource

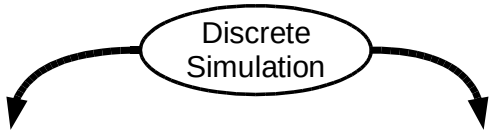
```
<<PAresource>>

PAutilization = @xx@
PAthroughput = @xx@
PAcapacity = 100
PAaxTime = [“assm”, “dist”,
        [“uniform”,10.0, 20.0]]
```



The approach

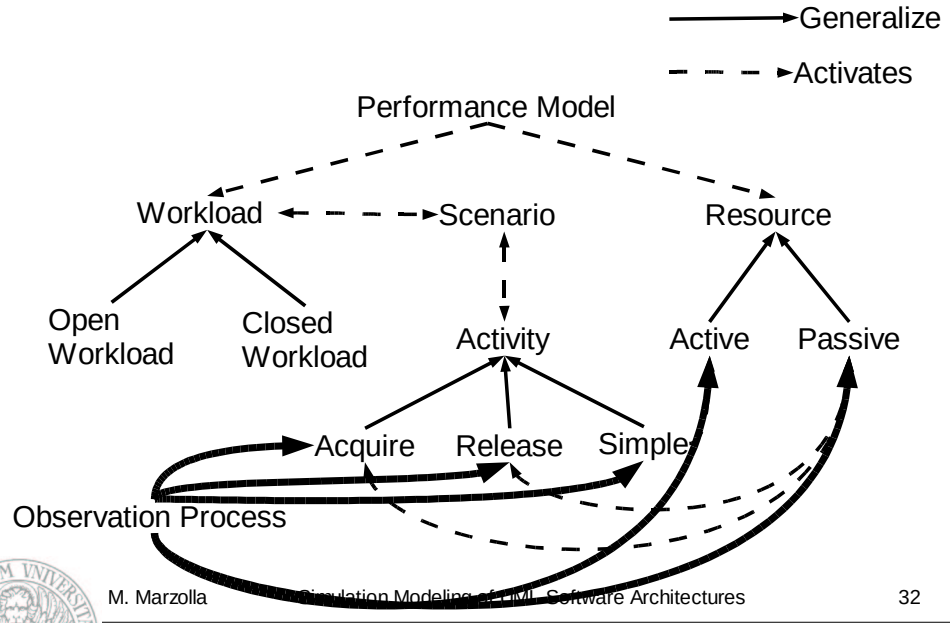


The two kinds of simulation



- Event-oriented
  - Each event is modeled by a procedure
  - Simulation is performed by scheduling procedure execution in the correct order
- Process-oriented
  - Each system component is modeled as a simulation process
  - The simulation model is a collection of concurrent, interacting processes
- Which one?
  - Software systems are based on the notion of *component*
  - It is natural to model components as simulation processes

The Simulation Performance Model

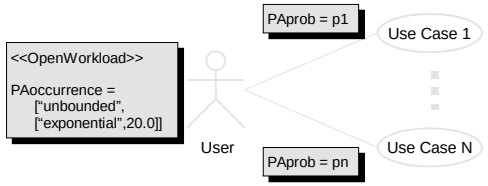


Modeling Algorithm

```
FOR ALL Use Case diagram U
    FOR ALL Actor a in U
        IF a is tagged as "OpenWorkload"
            Ac := new OpenWorkload(a) simulation process
        ELSE IF a is tagged as "ClosedWorkload"
            Ac := new ClosedWorkload(a) simulation process
        FOR ALL Use Case u associated with a
            Sc := new PScenario simulation process
            Link Sc to Ac
            A:=Activity diagram associated with u
            FORALL Activity s in A
                IF s is stereotyped as "PAstep"
                    IF a is a simple action
                        P[a] := new SimpleAction(a) sim. process
                    ELSE
                        P[a] := new CompositeAction(a) sim. process
                ELSE IF a is stereotyped as "GRMacquire"
                    P[a] := new AcquireAction(a) simulation process
                ELSE IF a is stereotyped as "GRMrelease"
                    P[a] := new ReleaseAction(a) simulation process
                ELSE IF a is a fork node
                    P[a] := new ForkAction(a) sim. process
                ELSE IF a is a join node
                    P[a] := new JoinAction(a) sim. process
                Add P[a] to Sc
            FORALL a,b in A such that b is successor of a
                Set P[b] as a successor of P[a]
```

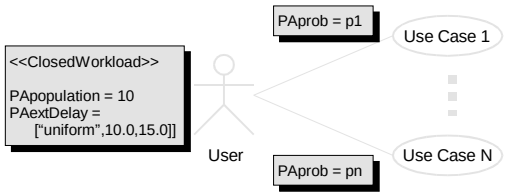
```
FORALL Deployment diagram D
    FORALL Node instance n in D
        IF n is tagged as "PRhost"
            p := new PRhost(n) process
        ELSE IF n is tagged as "PResource"
            p := new PResource(n) process
```


Translating Open Workloads



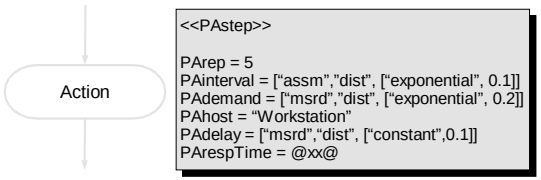

```
process OpenWorkload( Actor A );
var
  u : OpenWorkloadUser;
begin
  while ( true ) do begin
    hold( A.PAoccurrence );
    u := new( OpenWorkloadUser( A ) );
    activate( u );
  end;
end;

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;
```

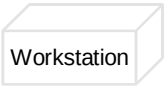

Translating Simple Actions




```
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;
```

Translating Active Resources

```
<<PAhost>>

PAutilization = @xx@
PAthroughput = @xx@
PAschedPolicy = "FIFO"
PActxSwT = ["assm", "dist",
        ["constant", 0.1]]
PArate = 2.0
```




```
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;
```


Simulation results

- We compute steady-state values
  - Mean execution time of actions
  - Utilization of resources
  - Throughput of resources
- Performance results are obtained as confidence intervals from a sequence of observations
- The user can define
  - The confidence level (90% by default)
  - The accuracy (10% of the central value by default)
  - The simulation duration

User interaction

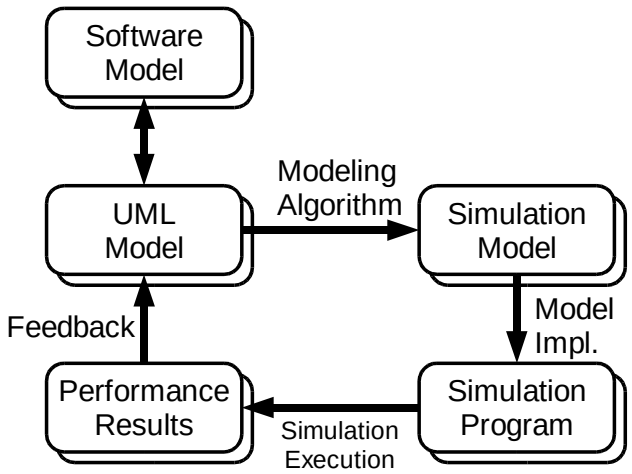
- If the computed confidence width is too large, the user may request to collect more data

  By resuming the simulation from the last saved checkpoint

  By running more replications from different initial conditions

  The more data is collected, the more accurate can be the results

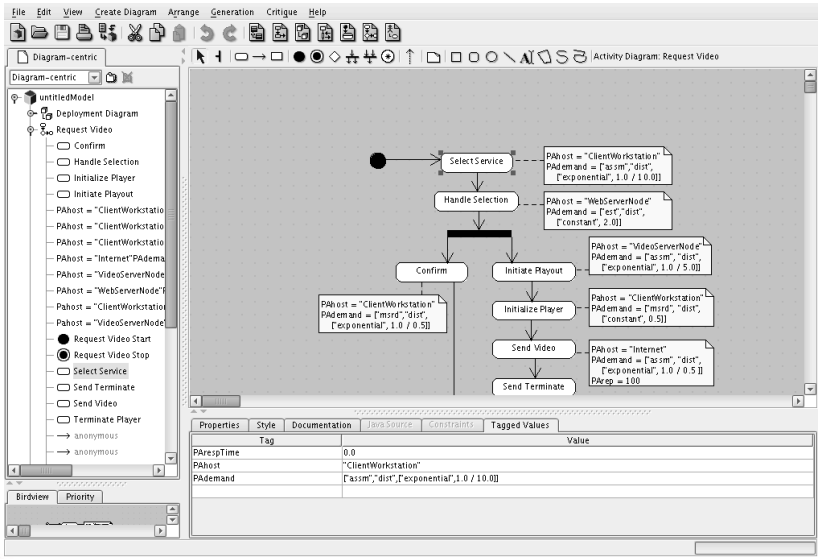
The approach



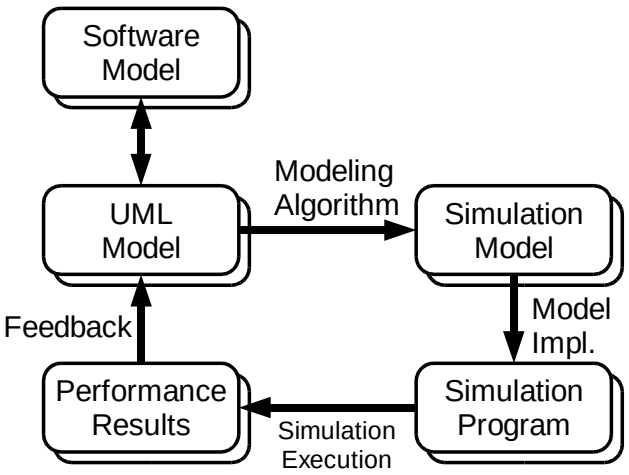
UML-

- 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

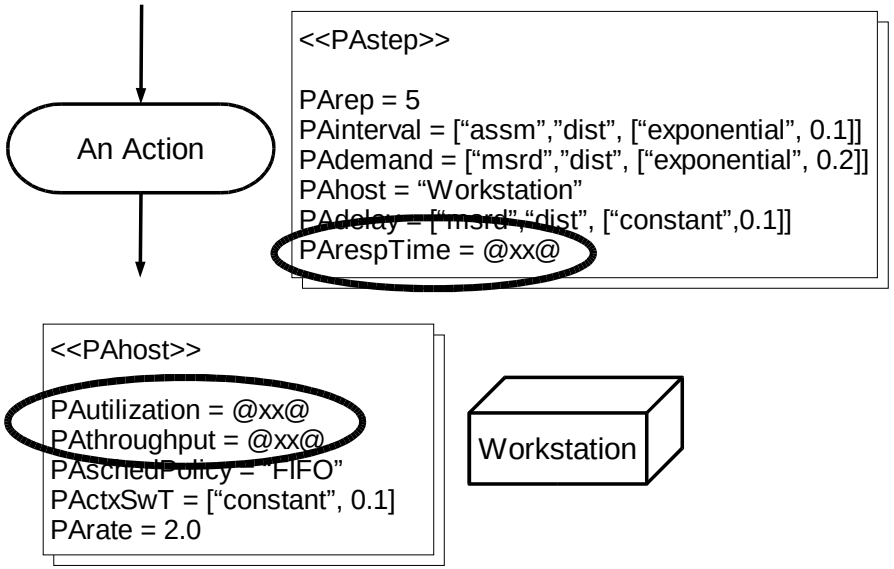
Modeling in action



The approach



Performance Results and Feedback



Conclusions

- We proposed an approach for performance evaluation of software systems at the design level
  - We defined a UML profile for specifying quantitative annotations to UML models
  - We defined a process-oriented simulation model
  - An algorithm for translating the software specification into the performance model has been defined
  - We developed a prototype tool (UML-) to demonstrate the applicability of the approach
  - The tool has been applied to some case studies

Further research

- **UML-related improvements**
  - Include more types of UML diagrams in the software model, such as State Charts and Sequence diagrams
  - UML 2.0
- **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

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

S. Balsamo and M. Marzolla. *Simulation Modeling of UML Software Architectures* Proc. ESM'03, Nottingham (UK), Jun 9—11 2003

S. Balsamo and M. Marzolla. *A Simulation-Based Approach to Software Performance Modeling*. Proc. ESEC/FSE 2003, Helsinki (FI), Sep 1—5 2003

S. Balsamo and M. Marzolla. *Towards Performance Evaluation of Mobile Systems in UML*. Proc. ESMc'03, Napoli, Oct 27—29 2003

S. Balsamo, A. Di Marco, P. Inverardi and M. Marzolla. *Experimenting Different Software Architectures Performance Techniques: A Case Study*. Proc. WOSP 2004, Redwood Shores (CA), Jan 14—16, 2004

S. Balsamo, M. Grosso and M. Marzolla. *Towards Simulation-Based Performance Modeling of UML specifications*, Technical Report CS-2003-2, Jan 2003, Dipartimento di Informatica, Università Ca' Foscari di Venezia.

Work supported by MIUR research project SAHARA and by FIRB research project "Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools"

Modifications to the Performance Profile

- We associate probabilities to transitions, rather than to Actions as proposed in the UML Performance Profile

  The same action may be successor of multiple predecessors, with different probabilities each

- We represent the Action hierarchy using the *composite pattern*

  The Profile states that “a step is a kind of scenario”, which appears not correct

- We define different kinds of Actions

  This makes the performance model derivation easier

Model-Based Performance Evaluation

- **Pro**
  - Predictive analysis is possible (does not require a running system)
  - Hence, it can be applied from the early design stages
  - Measurement-based analysis requires modifications to the observed system
- **Cons**:
  - Accuracy of the performance prediction depends on the accuracy of the software model (very high accuracy not needed anyway)
  - May be difficult to report feedback due to different structure of the performance model wrt the software model

An application to mobile systems

- We applied the approach to performance *and* mobility modeling of UML specifications

  Use Cases represent mobility patterns (eg, a “user moving fast”, “user moving slowly”)

  Use Cases are expanded as *high-level activity diagrams*, each action representing a particular Configuration of the system which is triggered as the user moves

  Each Configurations is detailed as low-level Activity diagrams describing the interactions happening while that particular configuration is active

Mobility and Performance Modeling with UML