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
  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
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
- Model-based Performance Evaluation
- Measurement-based Performance Evaluation

General Software Performance Modeling

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

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
  
  Simulation is 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 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
UML is a graphical modeling notation, widely used to describe Object-Oriented software systems.

- Provides several types of diagrams
  - Use Case
  - Deployment
  - Activity
  - State
  - Class / Package
  - Collaboration
  - Sequence
The Performance Model of the UML Profile

Problem 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

Model Generation

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

<<ClosedWorkload>>
PAtoken = 10
PAccessDelay = ["uniform", 10.0, 15.0]

User
PAccess = p1

Use Case 1

PAccess = pn

Use Case N

Modeling Scenarios / 1

• Simple Actions

<<PAstep>>
PAct = 5
PAinterval = ["assm", "dist", ["exponential", 0.1]]
PAdemand = ["msrd", "dist", ["exponential", 0.2]]
PProducer = "Workstation"
PAccess = ["msrd", "dist", ["constant", 0.1]]
PAccessTime = @x@

Modeling Scenarios / 2

• Resource Acquire/Release

<<GRMacquire>> / <<GRMrelease>>
PAresource = "Memory"
PQuantity = ["assm", "dist", ["constant", 2]]
PAccessTime = @x@

Modeling Resources / 1

• Active Resource (processor)

<<PAhost>>
PUtilization = @x@
PThroughput = @x@
PAschedPolicy = "FIFO"
PAccessTime = ["assm", "dist", ["constant", 0.1]]
PAccessRate = 2.0
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
FOR ALL Use Case diagram U
FOR ALL Actor a in U
    IF a is tagged as "Open Workload"
        Ac := new OpenWorkload(a) simulation process
    ELSE IF a is tagged as "Closed Workload"
        Ac := new ClosedWorkload(a) simulation process
    FOR ALL Use Case u associated with a
        Sc := new PSimulation process
        Link Sc to Ac
        A := Activity diagram associated with u
        FOR ALL Activity s in A
            IF s is stereotyped as "PAstep"
                P[a] := new SimpleAction(a) sim. process
            ELSE IF a is stereotyped as "GMAquire"
                P[a] := new AcquireAction(a) simulation process
            ELSE IF a is stereotyped as "GMRrelease"
                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
            ELSE IF n is a node
                Add P[a] to Sc
            FOR ALL a in A such that b is successor of a
                Set P[b] as a successor of P[a]

FOR ALL Deployment diagram D
FOR ALL Node instance n in D
    IF n is tagged as "PAHost"
        p := new PRhost process
    ELSE IF n is tagged as "PResource"
        p := new PRresource process
    ELSE IF n is tagged as "PRAccumulate"
        PAccumulate := "unbounded", ["exponential", 20.0]
    ELSE IF n is stereotyped as "GRAcquire"
        PrAcquire := new AcquireAction(n) sim. process
    ELSE IF n is stereotyped as "GRRlease"
        PrRelease := new ReleaseAction(n) sim. process
    ELSE IF n is stereotyped as "GRLfork"
        PrFork := new ForkAction(n) sim. process
    ELSE IF n is stereotyped as "GRLjoin"
        PrJoin := new JoinAction(n) sim. process
    ELSE IF n is stereotyped as "GRAccumulate"
        PAccumulate := ["exponential", 20.0]
    ELSE IF n is stereotyped as "GRLnode"
        Add PrAccumulate to Sc
    FOR ALL a in A such that b is successor of a
        Set Pr[b] as a successor of Pr[a]

FOR ALL Activity s in A
    IF s is stereotyped as "PAstep"
        P[a] := new SimpleAction(a) sim. process
    ELSE IF a is stereotyped as "GMAquire"
        P[a] := new AcquireAction(a) simulation process
    ELSE IF a is stereotyped as "GMRrelease"
        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
    ELSE IF n is a node
        Add P[a] to Sc
    FOR ALL a in A such that b is successor of a
        Set P[b] as a successor of P[a]
Translating Active Resources

```plaintext
<<PAhost>>

PUtilization = @x@ 
PAvailability = @x@ 
PAvailability = "FIFO" 
PAvailability = ["assm", "dist", "constant", 0.1] 
PAvailability = 2.0
```

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

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

The approach
UML-

- Written in C++
- Parses the XML 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

<<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@

<<PHost>>

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


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

An application to mobile systems

- We applied the approach to performance *and* mobility modeling of UML specifications
  Use Cases represent mobility patterns (e.g., 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

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

Mobility and Performance Modeling with UML