Autonomic management of non-functional concerns in distributed & parallel application programming

M. Aldinucci - Univ. of Torino
M. Danelutto - Univ. of Pisa
P. Kilpatrick - Queen’s Univ. of Belfast

IPDPS’09 - Rome - May 2009
Contents

- Rationale
- Functional & non-functional concerns
- Behavioural skeletons
- Hierarchical management of non-functional concerns in behavioural skeletons
- Implementation
- Experimental results
- Conclusions & Future work
Contents

- Rationale
- Functional & non-functional concerns
- Behavioural skeletons
- Hierarchical management of non-functional concerns in behavioural skeletons
- Implementation
- Experimental results
- Conclusions & Future work
Rationale

- Dynamic properties affecting parallel/distributed computations
  - non exclusive resource usage
  - resource heterogeneity in
dynamic resource recruitment
  - faults
  - application hot spots
- Well known policies & strategies
  - e.g. migration of computations,
    adjustment of parallelism degree, ... 
- Effectiveness requires
  - dynamic coupling of policies to triggering events
  - automatic policy implementation
Rationale

- Dynamic properties affecting parallel/distributed computations
  - non exclusive resource usage
  - resource heterogeneity in dynamic resource recruitment
  - faults
  - application hot spots

- Well known policies & strategies
  - e.g. migration of computations, adjustment of parallelism degree, ...

- Effectiveness requires
  - dynamic coupling of policies to triggering events
  - automatic policy implementation
Contents

- Rationale
- Functional & non-functional concerns
- Behavioural skeletons
- Hierarchical management of non-functional concerns in behavioural skeletons
- Implementation
- Experimental results
- Conclusions & Future work
Functional and non-functional concerns

- **Functional concerns**
  - Concur to determine *what* is actually computed
  - Business logic, domain specific

- **Non-Functional (extra functional) concerns**
  - Concur to determine *how* the results are actually computed
  - Business logic (& domain) independent

- **Separation of concerns enhance efficiency**
  - Business logic concerns → application programmers
  - Non business logic related concerns → system programmers
Functional and non-functional concerns

- **Performance**
  - parallelism degree setup & tuning, load balancing, resource targeting, ...

- **Security**
  - data & code securing, authentication, communication integrity

- **Fault tolerance**
  - fault recognition, fault repair, substitute resource recruiting, ...

- **Green computing**
  - power saving option exploitation (in clusters, blades, multicores)
  - adaptation of power consumption to expected performance w.r.t. application hot spots
Functional and non-functional concerns

- **Functional concerns**
  - concur to determine *what* is actually computed
  - business logic, domain specific

- **Non-Functional (extra functional) concerns**
  - concur to determine *how* the results are actually computed
  - business logic (& domain) independent

- **Separation of concerns enhance efficiency**
  - business logic concerns → application programmers
  - non business logic related concerns → system programmers
Contents

- Rationale
- Functional & non-functional concerns
- Behavioural skeletons
- Hierarchical management of non-functional concerns in behavioural skeletons
- Implementation
- Experimental results
- Conclusions & Future work
Behavioural skeleton (BS) concept

- Co-design of
  - parallelism exploitation pattern (skeleton)
    - known way of exploiting parallel machines
  - autonomic management of non-functional features
    - control loop based management, ensuring user defined QoS contract

- BS shields application programmer from the environment
Algorithmic skeleton & (parallel) design pattern theory

- around since late ‘80s
- most efficient parallel applications have a parallel structure (skeleton or pattern)
  - hierarchical composition of simple parallel patterns
  - pipeline, task farm, map, reduce, several flavors of data parallel, D&C, several flavors of loop, ...
- restricted parallelism models (eSkel, Muesli, P3L, ASSIST, Muskel, SkeTo, Skipper, CO$_2$P$_3$S, Google MapReduce, Hadoop, ...)

parallelism exploitation pattern restriction

- efficient implementation (reusable, by system programmers)
- optimizations (rewriting, JIT optimizations, normal forms, ...)
- performance models
Behavioural skeleton (BS) concept

- Co-design of
  - parallelism exploitation pattern (skeleton)
    - known way of exploiting parallel machines
  - autonomic management of non-functional features
    - control loop based management, ensuring user defined QoS contract

- BS shields application programmer from the environment
BS: autonomic management

- **Control loop based**
  - **monitor**: perceive system current behaviour
  - **analyse**: compare current behaviour with the expected one
    - needs abstract model of well behaving system
  - **plan**: devise a strategy to handle malfunctioning (if any)
  - **execute**: actuate the chosen plan → sequence of actions

- **Rule based**
  - **pre-condition** → action rules
  - **analyse** ⇒ lookup of fireable rules in the DB (those with pre-cond true)
  - **plan, execute** ⇒ apply action rules
Behavioural skeleton (BS) concept

- Co-design of
  - parallelism exploitation pattern (skeleton)
    - known way of exploiting parallel machines
  - autonomic management of non-functional features
    - control loop based management, ensuring user defined QoS contract

- BS shields application programmer from the environment
Behavioural skeleton (BS) concept

- **Co-design of**
  - *parallelism exploitation pattern (skeleton)*
    - known way of exploiting parallel machines
  - *autonomic management of non-functional features*
    - control loop based management, ensuring user defined QoS contract

- **BS shields application programmer from the environment**
Behavioural skeleton (BS) concept

- **Co-design of**
  - **parallelism exploitation pattern (skeleton)**
    - known way of exploiting parallel machines
  - **autonomic management of non-functional features**
    - control loop based management, ensuring user defined QoS contract

- **BS shields application programmer from the environment**
Behavoural skeleton (BS) concept

- **Co-design of**
  - parallelism exploitation pattern (skeleton)
    - known way of exploiting parallel machines
  - autonomic management of non-functional features
    - control loop based management, ensuring user defined QoS contract

- **BS shields application programmer from the environment**
Behavioural skeleton concept

- Complete separation of concerns
  - System programmers
    - parallelism exploitation pattern
    - autonomic management
  - Application programmers
    - “best” BS
    - business application logic
    - QoS contract (rules) → BS AM
Behavourial skeleton concept

- Complete separation of concerns
  - System programmers
    - parallelism exploitation pattern
    - autonomic management
  - Application programmers
    - “best” BS
    - business application logic
    - QoS contract (rules) \(\rightarrow\) BS AM
Behavioural skeleton (history)

- Introduced in GCM
  - The Grid Component Model by CoreGRID:
    ① Hierarchical components,
    ② With collective, stream and event ports
    ③ With autonomic managers taking care of non-functional features
  - First implementation within GridCOMP
    ProActive/GCM reference implementation (http://gridcomp.ercim.org)
    → Task farm & data parallel BS
    → With autonomic management of performance
  - Validate through GridCOMP industrial use cases
    • Biometric identification systems
    • Electronic record processing
    • Wing design code
Contents

- Rationale
- Functional & non-functional concerns
- Behavioural skeletons
- Hierarchical management of non-functional concerns in behavioural skeletons
- Implementation
- Experimental results
- Conclusions & Future work
Hierarchical management

- **Computations as BS compositions**
  - e.g. BS task farm worker in a BS pipeline

- **Autonomic managers**
  - react to local triggers
  - using local rules
  - lack of “global view” (and decisions!)

- **Need for hierarchical management**
  - provide “global framework” for non-local optimized decisions
  - at different levels → each root node provides a global decision framework for its leaves nodes
  - alternatives:
    - decisions taken at the root node or
    - “driven” by root node at leaf nodes
Hierarchical management

- Computations as BS compositions
  - e.g. BS task farm worker in a BS pipeline

- Autonomic managers
  - react to local triggers
  - using local rules
  - lack of “global view” (and decisions!)

- Need for hierarchical management
  - provide “global framework” for non-local optimized decisions
  - at different levels → each root node provides a global decision framework for its leaves nodes
  - alternatives:
    → decisions taken at the root node or
    → “driven” by root node at leaf nodes
Hierarchical management: new issues

1. Contract propagation
2. Local violation of contract with no local alternatives
3. Phases (different rôles w.r.t. management)
Hierarchical management: new issues

1. **Contract propagation**

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)

- **User contract**
  - directed to root manager

- **Actual need**
  - one contract per manager
  - contract “propagation” through the manager tree
    - contract may or may not change while propagating
    - kind of propagation depends on the type of the node propagating the contract

\[ \text{throughput} = k \text{ tasks / sec} \]
Hierarchical management: new issues

1. **Contract propagation**

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)

   - **User contract**
     - directed to root manager
   - **Actual need**
     - one contract per manager
     - contract “propagation” through the manager tree
       - contract may or may not change while propagating
       - kind of propagation depends on the type of the node propagating the contract
Hierarchical management: new issues

1. **Contract propagation**

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)

   - **User contract**
     - directed to root manager

   - **Actual need**
     - one contract per manager
     - contract “propagation” through the manager tree
       - contract may or may not change while propagating
       - kind of propagation depends on the type of the node propagating the contract

\[
\begin{align*}
\text{AM (Seq)} & \quad \text{throughput = } k \text{ tasks / sec} \\
\text{AM (Farm)} & \quad \text{throughput = } k \text{ tasks / sec} \\
\text{AM (Pipe)} & \quad \text{throughput = } k \text{ tasks / sec}
\end{align*}
\]
Hierarchical management: new issues

1. **Contract propagation**

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)

   - User contract
     - directed to root manager

   - Actual need
     - one contract per manager
     - contract “propagation” through the manager tree
       - contract may or may not change while propagating
       - kind of propagation depends on the type of the node propagating the contract

\[ \text{throughput} = k \text{ tasks/sec} \]
Hierarchical management: new issues

1. Contract propagation

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)
   - Local violation of contract
     - with no local alternatives
     - concept of violation raised to upper level manager
   - Moving responsibilities to the higher level
     - with global view
   - While raising violation
     - keep maintaining the current state
     - and keep raising violation to upper level
Hierarchical management: new issues

1. Contract propagation

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)
   - Local violation of contract
     - with no local alternatives
     - concept of violation raised to upper level manager
   - Moving responsibilities to the higher level
     - with global view
   - While raising violation
     - keep maintaining the current state
     - and keep raising violation to upper level

Diagram:

- AM (Seq) → AM (Pipe) → AM (Seq)
- AM (Farm) → AM (Seq)

throughput < contract
Hierarchical management: new issues

1. Contract propagation

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)
   - Local violation of contract
     - with no local alternatives
     - concept of violation raised to upper level manager
   - Moving responsibilities to the higher level
     - with global view
   - While raising violation
     - keep maintaining the current state
     - and keep raising violation to upper level
Hierarchical management: new issues

1. Contract propagation

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)
   - Local violation of contract
     - with no local alternatives
     - concept of violation raised to upper level manager
   - Moving responsibilities to the higher level
     - with global view
   - While raising violation
     - keep maintaining the current state
     - and keep raising violation to upper level
Hierarchical management: new issues

1. Contract propagation

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)
   - Local violation of contract
     - with no local alternatives
     - concept of violation raised to upper level manager
   - Moving responsibilities to the higher level
     - with global view
   - While raising violation
     - keep maintaining the current state
     - and keep raising violation to upper level

Diagram:
- AM (Seq)
- AM (Farm)
- AM (Pipe)

Throughput < contract
Input task rate < contract
Violation: not enough input
Hierarchical management: new issues

1. Contract propagation

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)

   - Local violation of contract
     - with no local alternatives
     - concept of violation raised to upper level manager

   - Moving responsibilities to the higher level
     - with global view

   - While raising violation
     - keep maintaining the current state
     - and keep raising violation to upper level
Hierarchical management: new issues

1. Contract propagation

2. **Local violation of contract with no local alternatives**

3. Phases (different rôles w.r.t. management)
   - Local violation of contract
     - *with no local alternatives*
     - Concept of violation raised to upper level manager
   - Moving responsibilities to the higher level
     - *with global view*
   - While raising violation
     - *keep maintaining the current state*
     - **and** keep raising violation to upper level

![Diagram of hierarchical management with AM (Seq), AM (Farm), and AM (Pipe) nodes indicating contract and output rate increase]

**INCREASE output rate**

**VIOLATION**

- not enough input

**AM (Pipe)**

- input task rate < contract
- contract

**AM (Seq)**

- AM (Seq)
- AM (Farm)
- AM (Pipe)
Hierarchical management: new issues

1. Contract propagation

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)
   - AM phases
     - different rôles w.r.t. management
     - active
       - taking local decisions
     - passive
       - local “standby”
       - waiting for new strategies
         (new rules to be used in the AM)
Hierarchical management: new issues

1. Contract propagation

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)
   - AM phases
     - different rôles w.r.t. management
     - active
       - taking local decisions
     - passive
       - local “standby”
       - waiting for new strategies
         (new rules to be used in the AM)
Hierarchical management: new issues

1. Contract propagation
2. Local violation of contract with no local alternatives
3. Phases (different rôles w.r.t. management)

- **AM phases**
  - *different rôles w.r.t. management*
  - *active*
    - taking local decisions
  - *passive*
    - local “standby”
    - waiting for new strategies (new rules to be used in the AM)
Hierarchical management: new issues

1. Contract propagation

2. Local violation of contract with no local alternatives

3. Phases (different rôles w.r.t. management)

   AM phases

   • different rôles w.r.t. management

   • active
     • taking local decisions

   • passive
     • local “standby”
     • waiting for new strategies (new rules to be used in the AM)
Hierarchical management: new issues

1. Contract propagation
2. Local violation of contract with no local alternatives
3. Phases (different rôles w.r.t. management)

- **AM phases**
  - *different rôles w.r.t. management*
  - *active*
    - taking local decisions
  - *passive*
    - local “standby”
    - waiting for new strategies
      (new rules to be used in the AM)
Hierarchical management: new issues

1. Contract propagation
2. Local violation of contract with no local alternatives
3. Phases (different rôles w.r.t. management)
   - AM phases
     - different rôles w.r.t. management
     - active
       - taking local decisions
     - passive
       - local “standby”
       - waiting for new strategies (new rules to be used in the AM)
Contents

- Rationale
- Functional & non-functional concerns
- Behavioural skeletons
- Hierarchical management of non-functional concerns in behavioural skeletons
- Implementation
- Experimental results
- Conclusions & Future work
**BS component implementation**

- **Composite component → component → black box**
- **Functional interface**
  - Set worker
    - component actually computing a single task
    - multiple instances organized within the BS
  - Compute tasks
    - stream or RPC port
- **Non-Functional interface**
  - Set contract
    - determine the intended behaviour w.r.t. a given non-functional concern
  - Report behaviour
    - query current and actual status of the computation
BS component implementation

- Composite component $\rightarrow$ component $\rightarrow$ black box
- Functional interface
  - Set worker
    - component actually computing a single task
    - multiple instances organized within the BS
  - Compute tasks
    - stream or RPC port
- Non-Functional interface
  - Set contract
    - determine the intended behaviour
    - w.r.t. a given non-functional concern
  - Report behaviour
    - query current and actual status of the computation
AM implementation

- AM implemented on top of JBoss drools
  - beans implementing behaviour
    - monitoring
    - actuator actions

- Sample rules
  - performance management

```java
rule "CheckRateLow"
when
  $departureBean : DepartureRateBean( value < ManagersConstants.FARM_LOW_PERF_LEVEL )
  $arrivalBean : ArrivalRateBean( value >= ManagersConstants.FARM_LOW_PERF_LEVEL )
  $parDegree: NumWorkerBean(value <= ManagersConstants.FARM_MAX_NUM_WORKERS)
then
  $departureBean.setData(ManagersConstants.FARM_ADD_WORKERS);
  $departureBean.fireOperation(ManagerOperation.ADD_EXECUTOR);
  $departureBean.fireOperation(ManagerOperation.BALANCE_LOAD);
end
```
AM implementation

- AM implemented on top of JBoss drools
  - beans implementing behaviour
    - monitoring
    - actuator actions
- Sample rules
  - performance management

```java
rule "CheckRateLow"
  when
  $departureBean : DepartureRateBean( value < ManagersConstants.FARM_LOW_PERF_LEVEL )
  $arrivalBean : ArrivalRateBean( value >= ManagersConstants.FARM_LOW_PERF_LEVEL )
  $parDegree: NumWorkerBean(value <= ManagersConstants.FARM_MAX_NUM_WORKERS)
  then
    $departureBean.setData(ManagersConstants.FARM_ADD_WORKERS);
    $departureBean.fireOperation(ManagerOperation.ADD_EXECUTOR);
    $departureBean.fireOperation(ManagerOperation.BALANCE_LOAD);
  end
```
Hierarchical management with rules

- Raising violations
  - $actuatorBean.fireOperation(ManagerOperation.RAISE_VIOLATION);

- Contract propagation
  - $actuatorBean.fireOperation(ManagerOperation.PUSH_LIST);
    - reconsult(file) or change bean values (constants) at the target AM

- Active / passive phases
  - active:
    one or more rules with pre-condition holding true and non-violation actions
  - passive:
    rule(s) with pre-condition holding true and violation raising actions
Contract propagation

rule "detectViolation"
when

$exception:ViolationIdentityBean(value==ManagerInitialValues.OUT_OF_RANGE_VIOLATION)
then

$exception.setData((Object)
   new MessageList(
      new ViolationMessage[]{
         new ViolationMessage(ManagerInitialValues.OUT_OF_RANGE_VIOLATION, $exception.getIdentity(), "PartitionSizeBean", "lowerBound:0.0;higherBound:30.0"),
         new ViolationMessage(ManagerInitialValues.OUT_OF_RANGE_VIOLATION, $exception.getIdentity(), "PartitionSizeBean", "lowerBound:0.0;higherBound:30.0")})
   $exception.fireOperation(ManagerOperation.PUSH_LIST);
end
Hierarchical management with rules

- **Raising violations**
  - $\text{actuatorBean.fireOperation(ManagerOperation.RAISE_VIOLATION)}$

- **Contract propagation**
  - $\text{actuatorBean.fireOperation(ManagerOperation.PUSH_OBJECT)}$
  - reconsult(file) or change bean values (constants) at the target AM

- **Active / passive phases**
  - active:
    - one or more rules with pre-condition holding true and non-violation actions
  - passive:
    - rule(s) with pre-condition holding true and violation raising actions
Contents

- Rationale
- Functional & non-functional concerns
- Behavioural skeletons
- Hierarchical management of non-functional concerns in behavioural skeletons
- Implementation
- Experimental results
- Conclusions & Future work
Experimental results

- **Sample application in GCM**
  - three stage pipeline with parallel second stage
  - rendering of mammography series to evidence possible cancer sites

- **First stage: Sequential code**
  - produce image stream (may vary image rate, if requested)

- **Second stage: Task farm**
  - process images (may vary parallelism degree (self-decision) or raise violations (no locally effective rule fireable)

- **Pipeline**
  - deals with violations raise from task farm
  - monitors pipeline stage balancing
Experimental results

- **Sample application in GCM**
  - three stage pipeline with parallel second stage
  - rendering of mammography series to evidence possible cancer sites
- **First stage: Sequential code**
  - produce image stream (may vary image rate, if requested)
- **Second stage: Task farm**
  - process images (may vary parallelism degree (self-decision) or raise violations (no locally effective rule fireable))
- **Pipeline**
  - deals with violations raise from task farm
  - monitors pipeline stage balancing
Experimental results

- Sample application in GCM
  - three stage pipeline with parallel second stage
  - rendering of mammography series to evidence possible cancer sites

- First stage: Sequential code
  - produce image stream (may vary image rate, if requested)

- Second stage: Task farm
  - process images (may vary parallelism degree (self-decision) or raise violations (no locally effective rule fireable)

- Pipeline
  - deals with violations raise from task farm
  - monitors pipeline stage balancing
Experimental results (sample)
Activity of pipeline manager

- **Top Manager Logic**
  - `delWorker`
  - `rebalance`
  - `addWorker`
  - `raiseViol`

- **Farm Manager Logic**
  - `0.0`
  - `0.2`
  - `0.4`
  - `0.6`
  - `1.0`

- **Global Behaviour**
  - Input Task Rate
  - Contract Throughput
  - Sensors
  - Effectors

- **Resources**
  - # cores
  - CPU cores
  - # reconfigurations

- Wall clock Time (mm:ss)
Activity of farm manager

Top Manager Logic
- delWorker
- rebalance
- addWorker
- raiseViol
- contrLow
- contrHigh
- tooMuch
- notEnough
- unbalance

Farm Manager Logic

Global Behaviour
- Contract
- Throughput
- Input Task Rate

Resources
- # cores
- CPU cores
- reconfigurations

Wall clock Time (mm:ss)
Application performance contract

- **Top Manager Logic**
  - delWorker
  - rebalance
  - addWorker
  - raiseViol

- **Farm Manager Logic**
  - contrLow
  - contrHight
  - tooMuch
  - notEnough
  - unbalance

- **Global Behaviour**
  - Contract
  - Throughput
  - Input Task Rate

- **Resources**
  - # cores
  - CPU cores
  - # reconfigurations

- **Wall clock Time (mm:ss)**
  - 35:20 to 39:00
Resources recruited to the application
Start: poor image feeding to farm
Farm raises violation: “cannot achieve required performance with this input image rate”

**Global Behaviour**
- **Contract**
- **Throughput**
- **Input Task Rate**

**Resources**
- **# cores**

**Wall clock Time (mm:ss)**
- 35:20
- 35:40
- 36:00
- 36:20
- 36:40
- 37:00
- 37:20
- 37:40
- 38:00
- 38:20
- 38:40
- 39:00

**Farm Manager Logic**
- **Top Manager Logic**
- **Farms**
- **Sensors**
- **Effectors**

**Resources**
- **Wall clock Time (mm:ss)**
- 35:20
- 35:40
- 36:00
- 36:20
- 36:40
- 37:00
- 37:20
- 37:40
- 38:00
- 38:20
- 38:40
- 39:00

**Events**
- **endStream**
- **tooMuch**
- **notEnough**
- **inquire**
- **decrRate**
- **incrRate**
- **unbalance**
- **notEnough**
- **tooMuch**
- **contrLow**
- **contrHigh**
- **raiseViol**
- **addWorker**
- **rebalance**
- **delWorker**
Pipeline issues new contract: “increase image production rate”
Image rate increases

- Top Manager Logic
  - delWorker
  - rebalance
  - addWorker
  - raiseViol
  - contrLow
  - contrHigh
  - tooMuch
  - notEnough
  - unbalance

- Farm Manager Logic
  - 0.0
  - 0.2
  - 0.4
  - 0.6
  - 0.8
  - 1.0

- Global Behaviour
  - Contract
  - Throughput
  - Input Task Rate

- Resources
  - # cores
  - # CPU cores
  - reconfigurations

Wall clock Time (mm:ss)

# cores

Autonomic management of non-functional concerns in distributed & parallel application programming
Farm fires local rule:

“image rate good greater than contract, performance below contract → add worker”
Farm fires violation: “too high image rate”
Pipeline issues new contract: “lower image rate”
Image rate decreases
Farm fires local rule: “high image rate, contract low → add worker”
Steady state ensuring contract
Demo (video)

- **Mandelbrot (for the sake of simplicity, same structure as before)**
  - first pipeline stage generates column coordinates, second stage (a task farm) generates pixel columns, third stage displays columns on video
  - Video 1: violation raised
  - Video 2: worker adjustment to ensure contract

- **Mammography rendering**
  - Video 3:
    - overload resource used (artificially)
    - manager adapts worker number
Raising violations (not enough input)
Raising violations (not enough input)

Not enough input (columns)
Throughput out of contract

Contract: 2 to 3 columns plotted per second
Raising violations (not enough input)
Worker adjustment to ensure contract
Worker adjustment to ensure contract

Enough input (columns)

Throughput out of contract

Autonomic management of non-functional concerns in distributed & parallel application programming
Worker adjustment to ensure contract

Further workers added (two)
Reaction to overload on used resources
Reaction to overload on used resources

AM reacts increasing resources used
Reaction to overload on used resources
Reaction to overload on used resources

further AM activity ensures again contract
Contents

- Rationale
- Functional & non-functional concerns
- Behavioural skeletons
- Hierarchical management of non-functional concerns in behavioural skeletons
- Implementation
- Experimental results
- Conclusions & Future work
Future work

- **Multi-concern management**
  - evaluating different kind of strategies
    - super manager (aka application manager)
    - co-management at top level manager in different hierarchies
    - co-management at active local level
  - assessing feasibility of rule based system
  - compiling user friendly QoS contracts to actual rules

- **Hierarchical management**
  - considering different non-functional concerns (security, green, etc.)

- **Adaptive rules**
  - learning from past events
  - new rules generated and evaluated on the fly
Conclusions

- **Feasibility of behavioural skeletons**
  - extended to hierarchies of managers
  - verified “on the field” with real applications

- **Minimal programming effort**
  - further reduced if compiler available (user syntax → actual rules)

- **New concepts introduced**
  - contract (propagation), violations, active/passive mode, ...
  - can be reused in multi-concern framework
Thank you for your attention

Any questions?