Behavioural skeletons: a programming abstraction relieving programmers of non functional concerns

Marco Danelutto

Dept. Computer Science, Univ. of Pisa, Italy &
Programming model Institute, CoreGRID
Outline

★ Characterization of distributed architectures
★ Behavioural skeletons
★ Applications
★ Experimental results
★ Ongoing work
★ Conclusions
Outline

★ Characterization of distributed architectures
★ Behavioural skeletons
★ Applications
★ Experimental results
★ Ongoing work
★ Conclusions
Distributed architectures: features of interest

★ heterogeneity
  • computing resources, interconnection network

★ dinamicity
  • node/network faults, non exclusive resource access

★ non negligible communication cost
  • communications, access to logically shared data
Cloud perspective ...

★ cluster of clusters
  • heterogeneous interconnection structure
★ successive upgrades/maintenance
  • heterogeneous resources
★ non exclusive usage
  • dynamic resource availability
Outline

★ Characterization of distributed architectures

★ Behavioural skeletons

★ Applications

★ Experimental results

★ Ongoing work

★ Conclusions
Behavioural skeleton

★ Programming abstraction (programming model)

★ Combine “existing” knowledge/technology
  
  • algorithmic skeletons & autonomic computing
  
  • parallelism exploitation & non functional features management

★ Raise the level of abstraction provide to the final user/programmer

★ Hide more and more details/programming efforts in RTS
The concept
The concept

- Algorithmic skeleton
- Autonomic manager
The concept

Behavioural skeleton
The concept

Functional concerns

Algorithmic skeleton

Behavioural skeleton
The concept

Functional concerns  Non functional concerns

Algorithmic skeleton

Autonomic manager

Behavioural skeleton
The concept

Behavioural skeleton
The concept

Parameters: application specific user responsibility
The concept

*Parameters*: application specific user responsibility

**Behavioural skeleton**

**working application**
Behavourial skeleton sample: task farm

★ Functional part: compute embarrassingly parallel application
  • parameters: function to be computed on each input item (task)

★ Non functional part: guarantee (best effort) a user supplied performance contract
  • in presence of faults, with dynamic and heterogeneous resources
AM: autonomic manager

- Implements classical autonomic cycle
  - (a) monitor current status
  - (b) look up for actions
  - (c) plan execution
  - (d) execute actions
  - (e) repeat
Sample autonomic management

★ In case of fault of Wi:

• lookup for new resource, instantiate new worker

★ Performance non compliant to user contract

• if not bound by inter-arrival time and/or communication latencies
  - add new worker, or
  
  - move slower worker to faster resources, if available
Sample autonomic management

★ Sensible programming effort required without BS:

• monitoring tools, analyse and planning capabilities, execution mechanisms

• code intermingled with functional code

• deep dependency on the parallelism exploitation pattern

  - if recognized could be reused (but for intermingled functional code!)
Kind of autonomic management

★ Reactive:
  • react to events (with the risk to be late)
    - rule triggering on monitoring events

★ Proactive
  • anticipate decisions in such a way the future behaviour of system may be influenced by proper actions (with the risk to be early)
    - rule triggering on monitoring events AND predicates on historical data

★ Actually relies on system programmer ability to define proper rules ...
Different rôles in autonomic management cycle

★ Passive rôle

• provide mechanisms: monitoring, actuation

★ Active rôle

• provide decision tools: analysis, action planning

★ Both are of interests

• with different features

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
Specialization of autonomic management

★ Active part: AM the Autonomic Manager

- analyses monitored data (computation status)
- takes decisions by applying known (or learned) policies

★ Passive part: AC the Autonomic Controller

- provides mechanisms to monitor computation and actuate actions
Specialization of autonomic management

queries measures
orders actions

answers monitored measures
reports about actions performed

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
Specialization of autonomic management

queries measures
orders actions

AC

AM

provided by BS programmer(s)

provided by BS programmer(s)

answers monitored measures reports about actions performed

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
Specialization of autonomic management

queries measures
orders actions

answers monitored measures
reports about actions performed

AC

programmed through contracts by user
provided by BS programmer(s)

AM

provided by BS programmer(s)
Going further ...

★ AM

- policies and strategies
  - fired upon conditions on the current computation status
  - actuated through sequences of actions

- most naturally expressed by
  - (business) rule system
Business rule system

★ Set of pre-condition(params) → action(params) rules

• if pre-condition(params)==true then apply action(params)

• possibly

  - more than a single rule pre-condition holds at a time

  - priorities + algorithms (e.g. Rete) to solve conflicts/order rules
AM with rules

Fetch parameters through AC

analyse

monitor

plan

execute

Lookup for fireable rules (solving conflicts)

Actuate rule right hand section through AC

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
Sample behaviour (AM)

★ Initially:

• rules used to customize the AM

★ Steady state

• AM → autonomic cycle

★ Rule firing

• AM performs actions through AC
Sample behaviour (AM)

★ Initially:

- rules used to customize the AM

★ Steady state

- AM → autonomic cycle

★ Rule firing

- AM performs actions through AC
Sample behaviour (AM)

★ Initially:

- rules used to customize the AM

★ Steady state

- AM → autonomic cycle

★ Rule firing

- AM performs actions through AC
Sample behaviour (AM)

★ Initially:
   - rules used to customize the AM

★ Steady state
   - AM \rightarrow \text{autonomic cycle}

★ Rule firing
   - AM performs actions through AC
Sample behaviour (AM)

★ Initially:

• rules used to customize the AM

★ Steady state

• AM → autonomic cycle

★ Rule firing

• AM performs actions through AC
Sample behaviour (AM)

★ Initially:
  - rules used to customize the AM

★ Steady state
  - AM → autonomic cycle

★ Rule firing
  - AM performs actions through AC
Sample behaviour (AM)

★ Initially:
  • rules used to customize the AM

★ Steady state
  • AM → autonomic cycle

★ Rule firing
  • AM performs actions through AC
Sample rules (Jboss/GridCOMP GCM BS)

rule "CheckInterArrivalRate"
salience 5
when
    $arrivalBean : ArrivalRateBean( value < ManagersConstants.LOW_PERF_LEVEL)
then
    $arrivalBean.setData(ManagersConstants.notEnoughTasks_VIOL);
    $arrivalBean.fireOperation(ManagerOperation.RAISE_VIOLATION);
    System.out.println( "InterArrivalTime not enough - Raising a violation");
end
rule "CheckRateLow"
when
    $departureBean : DepartureRateBean( value < ManagersConstants.LOW_PERF_LEVEL )
    $parDegree: NumWorkerBean(value <= ManagersConstants.MAX_NUM_WORKERS)
then
    $departureBean.fireOperation(ManagerOperation.REPLICATE_SHARE);
    $departureBean.fireOperation(ManagerOperation.BALANCE_LOAD);
    System.out.println( "Adding "+ManagersConstants.ADD_WORKERS+ "workers");
end
rule "CheckRateHigh"
when
    $departureBean : DepartureRateBean( value > ManagersConstants.HIGH_PERF_LEVEL )
    $parDegree: NumWorkerBean(value > ManagersConstants.MIN_NUM_WORKERS)
then
    $departureBean.fireOperation(ManagerOperation.KILL);
    $departureBean.fireOperation(ManagerOperation.BALANCE_LOAD);
    System.out.println( "Rate "+$departureBean.getValue()+" (Removing 1 workers)");
end
Sample rules (Jboss/GridCOMP GCM BS)

rule "CheckInterArrivalRate"
saliency 5
when
$arrivalBean : ArrivalRateBean( value < ManagersConstants.LOW_PERF_LEVEL)
then
$arrivalBean.setData(ManagersConstants.notEnoughTasks_VIOL);
$arrivalBean.fireOperation(ManagerOperation.RAISE_VIOLATION);
System.out.println( "InterArrivalTime not enough - Raising a violation");
end

rule "CheckRateLow"
when
$departureBean : DepartureRateBean( value < ManagersConstants.LOW_PERF_LEVEL)
$parDegree: NumWorkerBean(value <= ManagersConstants.MAX_NUM_WORKERS)
then
$departureBean.fireOperation(ManagerOperation.REPLICATE_SHARE);
$departureBean.fireOperation(ManagerOperation.BALANCE_LOAD);
System.out.println( "Adding "+ManagersConstants.ADD_WORKERS+ "workers");
end

rule "CheckRateHigh"
when
$departureBean : DepartureRateBean( value > ManagersConstants.HIGH_PERF_LEVEL)
$parDegree: NumWorkerBean(value > ManagersConstants.MIN_NUM_WORKERS)
then
$departureBean.fireOperation(ManagerOperation.KILL);
$departureBean.fireOperation(ManagerOperation.BALANCE_LOAD);

Outline

★ Characterization of distributed architectures

★ Behavioural skeletons

★ Applications

★ Experimental results

★ Ongoing work

★ Conclusions
Which kind of applications?

★ any one matching BS semantics

- e.g. task farm BS
  - image processing (e.g. noise reduction, medical image rendering)
  - parameter sweeping (e.g. financial data processing)
  - number crunching (e.g. FFT or LU co-processor)

- semantics: independent computations, same function, stream/bunch of tasks
Behavioural skeletons

★ Functional replication BS

- meta BS with parameters
  - S: the task distribution policy
  - C: the result gathering policy
  - AM: the management policies

- **Farm**: S=unicast, C=gather
- **Data Parallel**: S=scatter, C=gather/reduce
- **Fault tolerant**: S=broadcast, C=voting,
  Wi=different algorithm for the same f
Outline

★ Characterization of distributed architectures
★ Behavioural skeletons
★ Applications
★ Experimental results
★ Ongoing work
★ Conclusions
Pisa research activity in the field

**1990**

**P3L**
skeleton only
no autonomic management
parallelism degree computed dynamically

**muskel**
(task farm) skeleton only
first autonomic manager
main goal: fault tolerance

**2000**

**ASSIST**
coordination language + skeleton(s)
primitive autonomic management
for task farm and pipeline computations

**GCM**
behavioural skeleton
fully fledged autonomic management
main goal: performance management/tuning

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
P3L

★ Pisa Parallel Programming Language

- joint project Dept. Computer Science & HP Pisa Science
- first working skeleton based framework

P3L (Prolog compiler targetting Meiko CS/2, ‘91) ➔ Anacleto (open source, C + MPI host code, compiler, ‘95) ➔ SkIE (C, C++, F77, Java + MPI host code compiler ‘96) ➔ ...

- scalability demonstrated, range of applications within PQE2000 (Italian national project, with QSW and other University, CNR bodies)
Muskel

★ micro SKEleton Library

- full Java/RMI skeleton library
- task farm + pipeline skeletons
- translated to macro data flow code
- executed by a distributed interpreter
- with manager ensuring parallelism degree contract

Synthetic application
ASSIST: task farm

Synthetic application

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
ASSIST: data parallel

Synthetic application
ASSIST + muskel

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies

**Figure 4.** Dynamic (MPI parmod) vs. static (ASSIST) implementation of parmod: unbalanced computation case: completion times (upper) and efficiency (lower)

**Figure 5.** Dynamic (MPI parmod) vs. static (ASSIST) implementation of parmod: balanced computation case: completion time (upper) and efficiency (lower)
★ Grid Component Model

• developed within CoreGRID + reference implementation within GridCOMP

• behavioural skeleton concept introduced

  - reuses most of the ASSIST experience

• AM based on JBoss rule engine
GCM: data parallel adaptation

Biometric identification
GCM: task farm performance management

Medical image processing

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
GCM: task farm

Medical image processing

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
Outline

★ Characterization of distributed architectures

★ Behavioural skeletons

★ Applications

★ Experimental results

★ Ongoing work

★ Conclusions
Ongoing work

★ Fully fledged, hierarchical, rule based autonomic managers for behavioural skeletons
  • contract propagation in the BS tree
    - user defined top level contracts (SLAs)
    - derived inner contracts
  • experimenting composition strategies & policies
★ Proactive rules
  • e.g. re-considering temporarily unavailable resources
    - computation completely unrelated events trigger rules
    - rules set up new execution frameworks
Ongoing work (2)

★ exploitation of historical data
  • applies to proactive and reactive adaptation

★ (semi-)formal tools supporting manager design and development

★ merging with software engineering and agent pre-existing and complementary results
Conclusions

★ Large experience

• currently finalized to GCM, Muskel and ASSIST frameworks

★ So many things to do!

★ But definitely:

• important and effective tools to support efficient massively parallel/distributed programming
Acknowledges to: M. Aldinucci, S. Campa, G. Zoppi (Univ. Pisa),
P. Kilpatrick (Queen’s Univ, Belfast), P. Dazzi (ISTI/CNR Pisa)

Thank you for your attention
¿ Any questions ?