Grid programming with components: an advanced COMPonent platform for an effective invisible grid

WP3 - Perspective roadmap for autonomic management

Marco Aldinucci
Dept. Computer Science – University of Pisa
Programming Model Institute - CoreGRID

aldinuc@di.unipi.it

Unipi & ISTI-CNR working group
M. Danelutto, D. Laforenza, M. Aldinucci, N. Tonellotto, S. Campa, P. Dazzi, G. Zoppi
assembly of autonomic components

active autonomic components (in insulation)

passive autonomic components (in insulation)

component model enabling features

• distributed policies
• orchestration of policies
• formal tools, abstraction tools

• behavioural skeletons
• management policies

• behavioural skeletons
• reconfiguration operations
• component specific and infrastructural monitoring sensors

• lightweight primitives
• dynamic binding
• suitable life-cycle control
Passive Autonomic Components

- BeSke are parametric composite components
  - extend controllers with ABC & MC
    - non-functional ports for reconfiguration & monitoring
  - implements a modified LC
    - because the manager is a component that cannot be stopped
  - currently provide several flavours of inner components replication
    - stateless farm, stateless farm with initialization, stateless dataparallel, stateful dataparallel
- enough to cover use cases
- accomplishment state: 95%
  - remaining 5% is mostly related to code reengineering
Farm (functional replication)

- farm BeSke
  - stateless
  - composite with a pair of **streaming** unicast/from_any ports
  - support composite workers
  - provide parallelism degree change, load-balancing, etc ...
  - autonomic policy for performance assessed
    - for performance at least (in insulation)
  - support several use case
    - e.g. Atos
    - just wrap the **sequential** code in a primitive component, and farm it out
  - see Nicola’s talk & demo
farm (in theory ...)
farm (in reality ...)

Notes
- rebalancing operation is logically not needed and it has been introduced because tasks cannot be transiently stopped at the composite membrane
- not sure it works with non-streaming ports (not void return type)
Data parallel

- data parallel BeSke family
  - stateless and stateful
  - composite exhibiting any number of multicast-gather ports
  - they are bound to inner components according to a fixed predefined parametric pattern
    - e.g. server: scatter  client: gather
    - e.g. server: scatter and broadcast  client: none
  - autonomic policy for performance assessed
    - for performance at least (in insulation, not yet implemented)
  - supports use case (e.g. IBM)
  - stateful DP provides data redistribution according to user defined&implemented set/get redistribution ports

- see Sonia’s talk & demo
Data Parallel (stateless, pure map)
Data Parallel (stateful, distrib state, map)

Grid programming with components: an advanced COMPonent platform for an effective invisible grid

Notes
- any number of server and client ports (either RPC or stream, in theory)
- the model cannot (structurally) enforce init happens before requests on other ports
- port reconfiguration and data redistribution should be atomic (no tasks should be distributed in the middle. We are not sure we can enforce with proactive
- data redistributions are functional requests (they should be because they are related to business logic) but the inner components execute them by way of a NF port in such a way they can be executed in stopped state (workaround)
- enqueued task cannot be cancelled (IBM use case)
A more defined picture

Non-Functional client & server ports

receive new QoS contract from outer components
raise exceptions toward outer components

issue action execution

receive monitor

read inner passive component's sensors
reconfigure inner passive component
enforce new QoS contracts to inner active components
catch exceptions from inner active components

membrane

ABC

AM make decisions

Grid programming with components: an advanced COMPonent platform for an effective invisible grid
Active Autonomic Components

Active = passive + manager

- non-functional internal and external client and server ports
  - supporting event based orchestration

- the manager defines a local control policy + co-ordination with other managers
  - **when-event-if-cond-then-actions**
    - where actions can be invocations to the ABC or messages toward other managers

- reaction engine can be implemented via JBoss rules
  - already experimented with behavioural skeletons in SCA
      *Advances in autonomic components & services. Currently submitted*
## Active management policies - examples

**Component** | **Manager Contract** | $m_i$
---|---|---
$C_1$ active (pipe) | $c_P = K_{low} \leq T_{self} \leq K_{high}$ | $K_{low}, K_{high}$ constants; $T_{C_1}, T_{C_3}, T_{C_4}$ monitored
$C_2$ active (farm) | $c_P = (c_P_{super}) \land (T_{self} \leq T_{self})$ | $TT_{self} =$ request inter-arrival time; $n_{self} =$ #workers
$C_3$ active (pipe) | $c_P = c_P_{super}$ | $T_{C_3}, T_{C_4}, T_{C_5}$ monitored

### Component model enabling features

- **assembly of autonomic components**
- **active autonomic components** (in insulation)
- **passive autonomic components** (in insulation)

### Plan

<table>
<thead>
<tr>
<th>Plan</th>
<th>Expected Cost</th>
<th>Expected Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLF1 move the slower worker $C_w$ on a faster platform, if any</td>
<td>$h = \text{cost}{\text{stop}(C_w); \text{deploy}(C_w); \text{start}(C_w)}$</td>
<td>decrease service time. $T_P(\zeta_{t+\Delta}) = T_{C_{g_1}}(\zeta_1)\Delta$, $0 \leq \Delta \leq 1$ speed difference between the platforms</td>
</tr>
<tr>
<td>PLF2 increase parallelism degree (allocate $k$ new workers)</td>
<td>$h = \text{cost}{\text{deploy}(C_s); \text{start}(C_w)}$ for $k$ instances</td>
<td>decrease service time. $T(\zeta_{t+\Delta}) = T(\zeta_t)n/(n + k)$</td>
</tr>
<tr>
<td>PLF3 decrease parallelism degree (de-allocate $k$ workers)</td>
<td>$h = \text{cost}{\text{stop}(C_w)}$ for $k$ instances</td>
<td>increase service time. $T(\zeta_{t+\Delta}) = T(\zeta_t)(n + k)/n$</td>
</tr>
<tr>
<td>PLF4 raise violation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Active management - resume

- accomplishment state: 75%

- for the next deliverable
  - equip current demo with a simple manager
    - working in insulation
    - some of those already have in reality
    - code reengineering work mostly
    - farm and data parallel supported

- for the end of the project
  - example of manager coordination
    - probably miming IBM use case
  - dynamic pluggable QoS contracts
    - defined as mobile beans
Management orchestration - concepts

Grid programming with components: an advanced COMPonent platform for an effective invisible grid
accomplishment state: 50%

- so high just because we don’t plan to fully investigate the problem within project lifetime
- we plan just a single experiment
  - maybe related to IBM use case
- in general, we still have many open problems in this regard
GCM Component model thoughts

- possible technical flaws
  - non-uniformity of involved languages
    - ADL, xml, Java, Proactive (or other middleware)
  - lack of static tools
    - correctness of ADL files - really error prone
    - factories & automatic generation of non-creative code
      - how many copy-paste do you need to develop an application?
      - do components actually ease the development?
  - dynamic reservation & deployment support
    - cluster reservation mechanism should be rethought in the light of reconfiguration needs?
    - G5K

- conceptual flaws
  - what is the pragmatic of the composite?
  - is reuse really exploited?
GCM/Proactive thoughts & feedbacks

- scarce separation between component model and its run-time support (proactive)
  - the component run-time support is difficult to extend
  - a deep run-time knowledge is strictly required to develop any non-toyish application
- asynchronous communications + queues impair run-time reactivity
  - events (e.g. monitor) refers to current state, but any structural reconfiguration will be applied to non yet enqueued requests
    - e.g. task cancel IBM fingerprint
  - 90% of development effort consists in avoiding unwanted behaviour related to active object model
- life-cycle: stopped/running states probably not enough
  - proper destruction mechanism, states related to reconfiguration (reconf-safe),
- monitor still missing
  - no real possibility to experiment autonomic policies without it
- GCM has a pretty fat middleware
  - as said in Palma 2007
  - hard to know all the features (then hard to exploit them)
- Proactive versions
  - 3.9_beta
  - any porting to a new version will be complex at this point, apart for expected changes (e.g. monitor)

GCM implementation status

- GCM features under refinement
- My fat-free (underhanded) wishes
  - Avoid fat specification
    - Any implementation will hardly be compliant
    - Maybe already too fat
  - Avoid fat implementation
    - Nobody will use it, especially in the HPC community
- Trying to add a “dietetic” QoS control
  - less possible impact on the middleware, thus if the users don’t want it, they should not spend time avoiding it
Resume and perspectives

Grid programming with components: an advanced COMPonent platform for an effective invisible grid

gridCOMP

GridComp use case

management policies
QoS contracts & their management
contract engine (manager)
GridComp use case v2
G5K experiments

management co-ordination mechanisms and policies
use case examples
many open problems

autonomic applications

active autonomic components

passive autonomic components

component model features

monitoring API
reconfiguration API
Behavioural Ske
GridComp use case

methodology
programming tools
performance
NF & F features

performance
RISC & stable API

component model features

area of interest of WP3 (Unipi/ISTI at least)

now
end of the project
Grid programming with components: an advanced COMPonent platform for an effective invisible grid
Resume and perspectives

- **passive autonomic components**
  - almost all planned features have been implemented
    - LC, ABC, MC, and patches to several controllers
    - behavioural skeletons (functional replication stateless and stateful)

- **active autonomic components**
  - implementation of management policies for performance (farm and DP)
  - QoS contracts as JBoss rules (static and dynamic)
    - not originally planned, already experimented in SCA

- **orchestration of management**
  - working on the design
  - formalisation of reconfiguration and QoS contracts
  - formalisation of orchestration strategies
    - uniformly supporting important concepts for HPC and pervasive
      - locality, fault-tolerance, geographic position, ...

- all use cases are supported by BeSke