Performance aware open-world software in a 3-layer architecture

Diego Perez-Palacin, Universidad de Zaragoza, Spain
José Merseguer, Universidad de Zaragoza, Spain
Simona Bernardi, Università di Torino, Italy
Context

- Open world software
  - Publish-subscribe; SOA; grid computing; etc.
  - Key idea: software made of services.
    - Third parties providers; interplay without authorities.
  - Performance problems
    - Are valid the current assumptions in SPE?
    - Can we trust in these third-parties?
- Challenge
  - Self- adaptation or self-management

Context (2)

- Kramer & Magee proposal
  - Architecture for self-managed systems
    - Reference architecture.
    - Three layers → KM-3L
    - Benefits:
      - Scalability, abstraction, etc.
    - Inspired in autonomic systems (robotics), since they are self-managed systems.

KM-3L

Goal management

---

New Strategy

---

Strategy Request

Change management

---

Change Configuration

---

Status

Component control
KM-3L

- **Idea**
  - Identify what a self-managed system needs to carry out its mission, without human intervention.

- **Component control**
  - Carries out the system *mission*.
  - Sense environment; report *status*.

- **Change management**
  - Has the *strategy* to carry out the mission.
  - With a new *status*, executes the *strategy* to produce a new system *configuration*.
  - If the new configuration does not fulfill the mission then asks for a new *strategy*.

- **Goal management**
  - Produces strategies that satisfy the mission and consider the current configuration.

Challenge: exploit KM-3L for the open-world to incorporate performance
KM-3L-4-OpenWorld: Component Control

Goal management

Change management

Component control

- Change Configuration
- System Workflow
- Monitor

Status
KM-3L-4-OpenWorld: Component Control

Responsibilities:

1. Tracking performance of components.
2. Discover new components.
3. Discover which components are no longer available.
4. Bind & unbind components.

Key: monitor module

- (1) Measure time elapsed in the service calls.
- (2, 3 and 4.) As usual in open-world.

Other needs:

- Workflow (e.g., UML activity diagram)
- Syste m configuration (e.g., UML component diagram)

Output:

- Current status (monitored time, unreachable service)

Input:

- New configuration
KM-3L-4-OpenWorld: Change Management

Goal management

Change management

Component control

Reconfiguration Strategy

Reconfiguration Controller

Component Diagram

New Strategy

Strategy Request

Change Configuration

Status
KM-3L-4-OpenWorld: Change Management

- **Key:**
  - Reconfiguration controller module.

- **Output:**
  - New system configuration.

- **Input:**
  - System status.
  - A new strategy.

- **Actions:**
  - A component is no longer available or degraded.
    - Executes the strategy to find a proper substitute.
    - Reports new configuration.
  - A new component is available for a given service.
    - Updates the current system configuration.
KM-3L-4-OpenWorld: Goal Management

**Goal management**

1. Performance Goals
2. Reconfiguration Strategy Generator
3. System Workflow with SPE Specification

- New Strategy
- Strategy Request

**Change management**

**Component control**
KM-3L-4-OpenWorld: Goal Management

- **Responsibility:**
  - Produce performance aware reconfiguration strategies.

- **Key:**
  - Strategy generator module.

- **Approaches:**
  - Library of strategies.
  - Produce the strategy *on demand*.

- **Output:**
  - Strategy that meets the target performance goal (e.g., response time).

- **Input:**
  - The performance goal.
  - The workflow specification.
  - The current configuration.

- **Discussion point:**
  - To meet other goals (e.g., availability, price).
Example (inputs)

<<GaWorkloadEvent>>
{pattern = (open = (interArrivalTime = (exp(500, tu))))}

Call S1

<<GaAcqStep>>
{acqRes = C0, resUnits = 1}

<<PaStep>>
{extOpDemands = $S1 provider; extOpCount = 1}

Call S2

<<PaStep>>
{extOpDemands = $S2 provider; extOpCount = 3}

Call S3

<<GaRelStep>>
{relRes = C0, resUnits = 1}

<<PaStep>>
{extOpDemands = $S3 provider; extOpCount = 1}
Example (inputs)

<table>
<thead>
<tr>
<th></th>
<th>phase1</th>
<th>phase2</th>
<th>phase3</th>
</tr>
</thead>
<tbody>
<tr>
<td>C11</td>
<td>(5,3000)</td>
<td>(20,6000)</td>
<td></td>
</tr>
<tr>
<td>C21</td>
<td>(10,6000)</td>
<td>(70,2000)</td>
<td>(250,2000)</td>
</tr>
<tr>
<td>C22</td>
<td>(35,6000)</td>
<td>(140,4000)</td>
<td></td>
</tr>
<tr>
<td>C31</td>
<td>(20,2000)</td>
<td>(70,2000)</td>
<td></td>
</tr>
<tr>
<td>C32</td>
<td>(30,∞)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(MeanServiceTime, MeanSojournTime)
Example (strategy graph)

- Reconfiguration strategy → directed graph
- *Nodes* are system configurations
- *Edges* represent changes of configurations
  - *Forward* edges:
    - Replacement of a component.
    - Phase change of a component.
    - Labels → confidence levels.
  - *Backward* edges:
    - Timeouts to bring back the system to a previous configuration.
Example ($1^{st}$ Step: create initial node)

- Assume each provider works in best mode, i.e., minimum mean service time
- Four possible configurations in the example.

<table>
<thead>
<tr>
<th>Mean response time estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>C11:ph1</td>
</tr>
<tr>
<td>C11:ph1</td>
</tr>
<tr>
<td>C11:ph1</td>
</tr>
<tr>
<td>C11:ph1</td>
</tr>
</tbody>
</table>

- Each configuration parameterizes the Petri net.
- Solve the Petri nets and choose the best configuration.
Example (Petri net)

\[ GSPN = (N, \{\lambda_{S1provider}, \lambda_{S2provider}, \lambda_{S3provider}\}) \]

- **RequestArrival**
  - \( \lambda = 1/500tu \)

- **Call S1**
  - **AcqRes,Start_CallS1**
  - **S1OpDemand**
  - **End_CallS1**

- **Call S2**
  - **End_CallS2**
  - **Start_CallS2**
  - **S2OpDemand**

- **Call S3**
  - **Start_CallS3**
  - **S3OpDemand**
  - **RelRes,End_CallS3**
Example (2nd Step: create adjacent nodes)

- Consider that current providers can degrade their performance → 3 adjacent nodes

  - **Node1** (provider one degraded)
    - No choice → only one provider
    - Solve the Petri net using phase2 of C11.
    - Is the performance goal achieved?

  - **Node2** (provider two degraded)
    - Alternatives: use C22 or C21 in phase2.
    - Again four possible configurations.
      Solve the Petri net.
Example (3rd Step: Labels)

- Rational:
  - Our confidence in a configuration change.
  - Ad-hoc heuristic under the open workload assumption.

- Confidence = Improvement/((Improvement + Lost)
  - Improvement = RT_source_ch_phase - RT_target (OK reconfiguration)
  - Lost = RT_target - RT_source (wrong reconfiguration)
Rational

- We can perform erroneous reconfigurations.
- So, after a timeout *bring back the system* to a state that performs better:
  - Identify nodes where components perform in their worst phase.
- Ideal timeout? → Future work
We analyzed the system *without reconfigurations* and using the components with their best mean response times → Response time: 494 tu

We analyzed the system *using the strategy graph* → Response time: 436 tu

Improvement: 11%
Related works


- Performance evaluation in open-world. Assuming components evolving independently and unpredictably.
  - Queuing networks.
  - Does not address the problem of generate strategies.

[10,11,15] Menascé's works (ICWS'07, Performance Evaluation'07 and WOSP'05)

- Evaluate service-based software.
Conclusion

- **Original idea**
  - Introduce a reference architecture from self-managed systems in the open-world context.

- **Contributions**
  - Adapt KM-3L to open-world software → focus the Performance problem.
  - Proposal for reconfiguration strategies module.

- **Challenge**
  - From models to real implementations → software with the ability to reconfigure itself.
  - Problem → run-time Petri net evaluation with *exact analysis techniques*.
  - Solution → Use Petri net *bounds*.

- **Final Remark**
  - The algorithm has been implemented.
Thanks!