Design of Self-Managing Dependable Systems with UML and Fault Tolerance Patterns

Matthias Tichy, Daniela Schilling, Holger Giese
Application Example - RailCab

- Vision: Combine
  - individual transport
  - cost-effectiveness of public transport
- Autonomously operating shuttles using linear drive (maglev train)
- Build **contact-free** convoys for energy savings
- Information about the exact position necessary
  - Computation based on the stator waves
Motivation

- Example service structure:
  
  ![Diagram]

- Problem: position calculation service must be **highly reliable**
Contents

- Problem: position calculation service must be highly reliable

Solution: self-healing, fault-tolerant software

1. Application of software fault tolerance patterns, which capture:
   a) Abstract service structure of fault tolerance techniques
   b) Deployment restrictions (explicitly taking fault tolerance into account)

2. Automatic deployment
   a) Initial deployment
   b) Self-healing by deployment reconfiguration during runtime
- Capture the abstract structure of a well known fault tolerance technique
- Example: Triple Modular Redundancy (TMR)
Naïve, unrestricted deployment may yield unwanted results

- Deployment must respect restrictions imposed by the fault tolerance technique (redundancy, heterogeneity)
- Instead of manual deployment, enrich fault tolerance patterns by deployment restrictions
Deployment restrictions for the TMR pattern:

- Avoid crash failures:
  - Deploy redundant services to distinct nodes.

- Avoid single-point-of-failure of voter / multiplier:
  - Deploy voter and user to the same node.
  - If the user fails, the failure of the voter is no problem.

- Heterogeneous hardware platform:
  - Require different CPUs.

- Note:
Fault Tolerance Patterns

Application of TMR pattern

 SEN: StatorWaveSensor

 MULT: Multiplier

 PC1: Position Calculation

 PC2: Position Calculation

 PC3: Position Calculation

 VOT: Voter

 CC: Convoy Controller

 SERVICE1

 SERVICE2

 SERVICE3

 USER

 PROVIDER

 Multiplier

 Position Calculation

 Voter

 Convoy Controller

 StatorWaveSensor

 Position Calculation

 ConvoyController
Compute a correct / reliable deployment

- Use a standard constraint solver
Compute a correct / reliable deployment

- Mapping to a standard constraint solver

<table>
<thead>
<tr>
<th>Nodes (j)</th>
<th>Services (i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gorlois</td>
<td>1</td>
</tr>
<tr>
<td>Taliesin</td>
<td></td>
</tr>
<tr>
<td>Gareth</td>
<td></td>
</tr>
<tr>
<td>Avalon</td>
<td></td>
</tr>
<tr>
<td>Arthur</td>
<td></td>
</tr>
</tbody>
</table>

• Variables $x_{i,j} \in \{0,1\}$
• Constraint: (each service is deployed to exactly one node)

$\forall i \exists j \text{ Services: } \sum x_{i,j} = 1$
Compute a correct / reliable deployment

- Restriction: Services must be executed on same node
- Graphically:

  - Constraint:

  j Nodes: 
  \( (x_{vot,j} + x_{cc,j}) = 2 \) or \( (x_{vot,j} + x_{cc,j}) = 0 \)

<table>
<thead>
<tr>
<th>( x_{i,j} )</th>
<th>sen</th>
<th>mul</th>
<th>pc1</th>
<th>pc2</th>
<th>pc3</th>
<th>vot</th>
<th>cc</th>
<th>( \Sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Gareth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Taliesin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Gorlois</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Uther</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Arthur</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Compute a correct / reliable deployment

- Initial deployment:

<table>
<thead>
<tr>
<th>$X_{i,j}$</th>
<th>sen</th>
<th>mul</th>
<th>pc1</th>
<th>pc2</th>
<th>pc3</th>
<th>vot</th>
<th>cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Gareth</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Taliesin</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Gorlois</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Uther</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Arthur</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Graphically:
Repair

- Due to TMR, one crashed redundant service is tolerable

- Second crash is **not** tolerable
- Therefore: Self-heal by restarting the crashed service on a working node.
- But on which node? Compute it timely!

```
Arthur <<deploys>>

pc3:PositionCalculation

Uther <<deploys>>

pc2:PositionCalculation

Gorlois <<deploys>>

pc1:PositionCalculation
```

<<Service1>>

<<Service2>>

<<Service3>>
Repair

- Solve restricted deployment problem

<table>
<thead>
<tr>
<th>Service</th>
<th>sen</th>
<th>mul</th>
<th>pc1</th>
<th>pc2</th>
<th>vot</th>
<th>cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Gareth</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Taliesin</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Gorlois</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Uther</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Arthur</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- Fix variables $x_{i,j}$ of unaffected service/node combinations
- Free variables $x_{i,j}$ for affected service/node combinations

Service migration should be avoided!
Refrain

- No solution found -> relax the problem

- First round ✗
  - Service migration necessary

- Second round ✗

- Third round ✓

<table>
<thead>
<tr>
<th>$x_{i,j}$</th>
<th>sen</th>
<th>mul</th>
<th>pc1</th>
<th>pc2</th>
<th>pc3</th>
<th>vot</th>
<th>cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Gareth</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Taliesin</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Gorlois</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Uther</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Arthur</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Fixed variables
- Free variables
Conclusions / Future Work

- Fault tolerance patterns capture fault tolerance techniques
  - Contain: Structure + Deployment restrictions
  - Graphical specification
  - Easy application

- Automatic deployment
  - Initial deployment
  - Self-healing by deployment reconfiguration during runtime

- Future Work
  - Heterogeneous service restrictions in pattern application
  - Synthesize behavior of voter and multiplier services
  - Use fault tolerance application knowledge for automatic fault tree analysis
Compute a correct / reliable deployment

- Restriction: Services must be deployed on distinct nodes
- Graphically:

<table>
<thead>
<tr>
<th>Constraint:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_{j} (x_{pc1,j} + x_{pc2,j}) \leq 1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nodes</th>
<th>$x_{i,j}$</th>
<th>sen</th>
<th>mul</th>
<th>pc1</th>
<th>pc2</th>
<th>pc3</th>
<th>vot</th>
<th>cc</th>
<th>$\sum$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Gareth</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Taliesin</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Gorlois</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Uther</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Arthur</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Arthur Avalon Gareth Taliesin Gorlois Uther
Compute a correct / reliable deployment

- Restriction: CPU of nodes for redundant service must be distinct

- Constraint:

\[ X_{s1,j2} \uparrow X_{s2,j2} \uparrow X_{s3,j3} \]
\[ j1.cpu \ominus j2.cpu \ominus j3.cpu \]
Multistage

- Multiple applications of TMR
- Transform to a multiple stage arrangement
  - with tripled voter and multiplier
- Deployment restrictions are transformed too
Tool Support

- Already working:
  - Graphical specification of fault tolerance patterns and deployment restrictions
  - Automatic mapping to ILOG solver software

- Next:
  - Runtime support
Motivation

- System without fault-tolerance, node crash
- (1) fault tolerance pattern
- Introduce redundancy (TMR) but deploy two redundant services on one node, crash of that node
- Better deploy redundant services to distinct nodes
- (2) deployment restrictions for fault tolerance pattern
- Example with distinct nodes, crash failure, everything ok
- (3) which nodes
- Now self-heal the system by restarting a crashed failure on a new node
Motivation

Goals:

- Easy application of fault tolerance techniques
- Deployment issues taken into account
  - Easy deployment specification
  - Reusable deployment specification