Understanding Consistency Maintenance in Service Discovery Architectures during Communication Failure

Chris Dabrowski, Kevin Mills, Jesse Elder

WOSP 2002
Rome, Italy
July 25, 2002
Dynamic discovery protocols in essence...

enable **distributed software components**

1. to **discover** each other without prior arrangement,
2. to **express** opportunities for collaboration,
3. to **compose** themselves into larger collections that cooperate to meet an application need, and
4. to **detect and adapt** to failures.

Some examples:

<table>
<thead>
<tr>
<th>3-Party Design</th>
<th>2-Party Design</th>
<th>Adaptive 2/3-Party Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>JINI</td>
<td>UPnP Forum</td>
<td>SLP for Enterprise Networks</td>
</tr>
<tr>
<td>Vertically Integrated 3-Party Design</td>
<td>HAVi</td>
<td>Bluetooth™</td>
</tr>
<tr>
<td>Network-Dependent 3-Party Design</td>
<td>Network-Dependent 2-Party Design</td>
<td></td>
</tr>
</tbody>
</table>
General Architecture for Service Discovery Systems

**SM (Service Manager)**
- Discover Network Context
- Announce Service Processing
- Renewal
- Start Service Parameter Matching Task
- Announce Service Processing
- Cache Manager Discovery
- Discover Network Context (from Data View)
- Service Parameter Change Notification

**SU (Service User)**
- Discover Network Context
- Service Discovery
- Start Renewal Task
- Start Service Parameter Matching Task
- Service Cache Manager
- Start Aging Task

**Local Cache Manager**
- Manage Service User
- Notification Request
- Service Cache

**SCM (Service Cache Manager)**
- Service Repository
- Notification Request
- Notification Cache
- Service Cache

**Service Repository**
- Service Description
- Service Provider
- Parameter Change Notification Cache
- Service User

**Color key**
- Common Structures
- Third-Party Structures
- Optional Structures

7/25/02
Modeling and Analysis Approach

**Scenario**

- **ACTION** DB_Update : dual DB_Update;
  SCM_Update : SCM_Update;
  DD_SCM_Update : DD_SCM_Update;
  DISC_MODES : dual SCM_DISCOVERY_MODES;
  DDC_SEND_DIR : DIRECTED_2_STEP_PROTOCOL;

**IS INTERFACE**

- **TYPE** Directed_Discovery_Client
  
  -- removes SCMs accordingly
  -- Receives updates from SCM DB of discovered SCMs and
da list of discovery mode. Sends Unicast messages to SCMs on list of
  -- NOTE: Failure and recovery behavior are not
  -- yet defined and need review

**MODEL**

- **For All** (SM, SD, SCM): (SM, SD) isElementOf SCM registered services
  - (SM, SD) IsElementOf SCM registered services
  - (SM, SD) IsElementOf SCM registered services
  - (SM, SD) IsElementOf SCM registered services
  - SCM IsElementOf SM discovered SCMs &
  - SCM IsElementOf SM discovered SCMs &
  - SCM IsElementOf SM discovered SCMs &
  - SCM IsElementOf SM discovered SCMs &

- **For All** (SM, SD, SCM): (SM, SD) IsElementOf SCM registered services
  - SCM IsElementOf SM registered -SCMs &
  - SCM IsElementOf SM registered -SCMs &
  - SCM IsElementOf SM registered -SCMs &
  - SCM IsElementOf SM registered -SCMs &

- **For All** (SM, SD, SCM): (SM, SD) IsElementOf SCM registered services
  - SCM IsElementOf SM registered -SCMs &
  - SCM IsElementOf SM registered -SCMs &
  - SCM IsElementOf SM registered -SCMs &
  - SCM IsElementOf SM registered -SCMs &

**Consistency Conditions**

- **For All** (SM, SD, SCM): (SM, SD) isElementOf SCM registered -services
  - implies SCM isElementOf SM discovered -SCMs
  - implies SCM isElementOf SM discovered -SCMs
  - implies SCM isElementOf SM discovered -SCMs
  - implies SCM isElementOf SM discovered -SCMs

**Execute with Rapide**

- **TOPOLOGY**

**Behavior Model**

- **Analyzer** POSETs

**Assess Correctness & Performance**

**Use metrics to Assess Correctness & Performance**

**Execute**

<table>
<thead>
<tr>
<th>Time</th>
<th>Command</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>NodeFail</td>
<td>SM4</td>
</tr>
<tr>
<td>5</td>
<td>LinkFail</td>
<td>SCM1 SM4</td>
</tr>
<tr>
<td>10</td>
<td>GroupJoin</td>
<td>SM4 GROUP1</td>
</tr>
<tr>
<td>10</td>
<td>FindService</td>
<td>SU8 5 1 2 5 XYZ ALL</td>
</tr>
<tr>
<td>50</td>
<td>AddService</td>
<td>SM4 SCM3 T ATT API GUI 20 30</td>
</tr>
</tbody>
</table>
How do various service discovery architectures, topologies, and consistency-maintenance mechanisms perform under deadline during communication failure?

**UPnP**

- UPnP Multicast Group
- Service User
- Service Manager
- HTTP/TCP Unicast Links
- HTTP/UDP Unicast Links
- Five SUs
- One SM

**Jini**

- Aggressive Discovery
- Multicast Group
- Service User
- Service Cache Manager
- Remote Method Invocation
- Unicast Links
- Lazy Discovery
- Multicast Group
- One SM
- Five SUs
- Optional 2nd SCM

**Two-Party Polling**

- Change Service
- Poll
- Poll
- Poll
- SM
- SU

**Two-Party Notification**

- Change Service Request
- Notification
- Notification
- SM
- SU

**Three-Party Polling**

- Change Service
- Poll
- Poll
- Poll
- SM
- SCM
- SU

**Three-Party Notification**

- Change Service Request
- Notification
- Notification
- SM
- SCM
- SU
Modeling Communication Failures

1. Choose a time to introduce the change \([\text{uniform}(Q, D/2)]\)
2. For each node, choose a time to introduce a communication failure \([\text{uniform}(Q, D-(D*F))]\)
3. When each failure occurs, choose a scope for the failure, where each of \([Rx, Tx, Both]\) has an equal probability

\[Q = \text{end of quiescent period (100 s in our experiment)}\]
\[D = \text{propagation deadline (5400 s in our experiment)}\]
\[F = \text{failure duration (variable from 0\% - 75\% in 5\% increments in our experiment)}\]
Monitoring Consistency

For All (SM, SU, SD):
(SM, SD [Attributes1]) IsElementOf SU discovered-services
SD [Attributes2] IsElementOf SM managed-services
implies Attributes1 = Attributes2

How well does the system restore consistency after restoration of communication?
Division of Failure Recovery Responsibilities:

<table>
<thead>
<tr>
<th>Communication Protocol</th>
<th>Discovery Protocol</th>
<th>Application Software</th>
</tr>
</thead>
</table>

**Application Software**
Application-specific behaviors, including responses to remote exceptions: (1) ignore, (2) retry for awhile and then give up, and/or (3) discard local knowledge of remote components

**Application behaviors required by the discovery protocol**

**Remote Exceptions**

<table>
<thead>
<tr>
<th>Protocol Type</th>
<th>Delivery Guarantees</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>attempts reliable delivery</td>
</tr>
<tr>
<td>Unicast UDP</td>
<td>no delivery guarantees</td>
</tr>
<tr>
<td>Multicast UDP</td>
<td>no delivery guarantees</td>
</tr>
</tbody>
</table>

**The Network**

**Discovery Protocol**
periodic transmission of key messages
Measures the probability that information will propagate successfully to a SU before a deadline, $D$. 

Update Effectiveness UPnP (2-Party) vs. Jini (3-Party)
Update Responsiveness UPnP (2-Party) vs. Jini (3-Party)

Measures the proportion of the available time remaining after the information is propagated.

Median Update Responsiveness

- 2-Party Notification
- 2-Party Polling
- 3-Party Notification 1 SCM
- 3-Party Polling 1 SCM
- 3-Party Notification 2 SCMs
- 3-Party Polling 2 SCMs

Interface Failure Rate (%)
Update Efficiency UPnP (2-Party) vs. Jini (3-Party)

Measures ratio of theoretical minimum number of messages needed to propagate a change to number of messages observed.
Conclusions

• Executable architectural models represent essential complexity and reveal collective dynamics – leading to valuable insights
  – paper specifications do not represent dynamics very well
  – reference implementations exhibit substantial incidental complexity

• A single architectural model can be analyzed for behavioral, performance, and logical properties
  – limits errors and inconsistencies that can creep in when using multiple models to represent different facets of a design

• 2-party and 3-party discovery architectures share similar robustness properties during communication failure, but
  – sole reliance on TCP retransmissions to recover notifications leads to an unexpected saw-tooth in update effectiveness, which is most pronounced for UPnP (Jini includes some SM behaviors which compensate)
  – adding a redundant SCM in the 3-party architecture improves effectiveness and responsiveness nearly to the level of the 2-party architecture, but adding a redundant SCM also lowers efficiency