Performance of Service-Discovery Architectures in Response to Node Failures

Chris Dabrowski, Kevin Mills, Andrew Rukhin

2003 International Conference on Software Engineering Research and Practice (SERP'03)
Las Vegas, Nevada

June 23-26, 2003
Dynamic discovery protocols... 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</td>
<td>SLP</td>
</tr>
<tr>
<td>Vertically Integrated 3-Party Design</td>
<td>Network-Dependent 3-Party Design</td>
<td>Network-Dependent 2-Party Design</td>
</tr>
</tbody>
</table>
Service Discovery Protocols in Distributed Environments

Enable dynamic location and combination of remote services to perform critical, real-time tasks.

“Actuators” receive ad-hoc requests.

“Fast” sensors send readings every 2 seconds.

“Slow” sensors send readings every 30 seconds.

SU is Service User
SM is Service Manager

Mobile command post Service User dynamically locates and combines sensors and actuators.
How Well Do Service Discovery Protocols Replace Services Lost to Node Failure?

- "Actuators" receive ad-hoc requests
- "Fast" sensors send readings every 2 seconds
- "Slow" sensors send readings every 30 seconds

- SU is Service User
- SM is Service Manager

Mobile command post Service User dynamically locates and combines sensors and actuators
Two generic architectures underlie most service discovery protocols

- **In two-party architectures**, Service Users discover Service Managers directly and invoke services.
- **In three-party architectures**, both Service Managers and Service Users discover Service Cache Managers (SCMs); SU obtains services through SCM intermediary and then invokes.
Two generic architectures (continued)

- **Discovery (2-party)**: SU discovers SMs through multicast search strategies
  - **Registration on SM**: SU registers for notification of change in service (renews every 300s)
- **Discovery (3-party)**: both SMs and SUs discover SCMs through multicast search
  - **Registration on SCM**: SMs register services (renews every 300s for fast sensors; 60s for slow sensors and actuators); SU registers notification requests (renews every 300s)
- **Failure Detection** by SU: through (1) SM non-response or (2) failure of registration renewal (heartbeat mechanism) and notification in 3-party case
  - **Recovery**: 2-party SU multicasts queries to SMs every 120s
  - **Recovery**: 3-party SU queries SCMs for service; if SCMs lost, SU listens for SCM announcements (every 120s) & SMs do the same
**Experiment Design**

<table>
<thead>
<tr>
<th>Initial Discovery occurs</th>
<th>Node failure randomly occurs for SMs and SCMs using MTF = (1-R)*D and “stepped” normal distribution. Three failure durations: short, medium, and long. SU is not failed.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TIME</strong></td>
<td></td>
</tr>
<tr>
<td>Q = end of quiescent period (60 s)</td>
<td>D = propagation deadline (1800 s)</td>
</tr>
</tbody>
</table>

- **Goal of SU is to be** *functional*; e.g., to continually possess one instance of each type of service (“fast” sensor, “slow” sensor, & actuator).
  - When >= 1 type of sensor is missing, SU is *non-functional*
  - To focus on alternative architectures & associated processes, mechanisms such as service caching factored out
- **Formal conditions for measuring latency in detecting service failure and replacing lost service provide basis for metrics**

**Detecting Failure of Services in Use**

Service User should hold services that are being actively managed (e.g., available)

For All (SM, SU, SD)

- (SM, SD) isElementOf SU discovered-services implies
- SD isElementOf SM managed-services

**Recovering and replacing failed services**

SDP should provide Service User with needed services if they are available

For All (SM, SU, SD)

- SD [capabilities] isElementOf SM managed-services
- SD [capabilities] isElementOf SU required-services
- ResourceNeeded (SU, SD) implies
- (SM, SD) isElementOf SU discovered-services
Modeling and Analysis Approach: Use Rapide ADL to Model and Understand Dynamics of Service Discovery Protocols

--- 3.3 DIRECTED DISCOVERY CLIENT INTERFACE ---
This is used by all JINI entities in directed discovery mode. It is part of the SCM_Discovery Module. Sends Unicast messages to SCMs on list of SCMS to be discovered until all SCMS are found. Receives updates from SCM DB of discovered SCMs if removes SCMs accordingly.

NOTE: Failure and recovery behavior are not yet defined and need review.

**Type** Directed_Discovery_Client
(SourceID : IP_Address; InSCMsToDiscover : SCMList; StartOption : DD_Code; InRequestInterval : TimeUnit; InMaxNumTries : integer; InPV : ProtocolVersion)

**Is Interface**

For All (SM, SD, SCM):

- (SM, SD) IsElementOf SCM registered -services (CC1)
- SCM IsElementOf SM discovered -SCMs

For All (SM, SD, SCM):

- (SM, SD) IsElementOf SCM managed -services
- (SM, SD) IsElementOf SCM registered -services & Performance

Analyze POSETs
Use metrics to Assess Correctness & Performance
Functional Effectiveness of Two-Party vs. Three-Party When One SM of Each Type is Always Available

Measures the proportion of time the Service User possesses the operational set of remote services needed to accomplish its task during $D$.

Replacement case with 60 repetitions per data point

Average Functional Effectiveness

Failure Rate (%)
Efficiency of Two-Party vs. Three-Party When One SM of Each Type is Always Available

Replacement case with 30 repetitions per data point

- **Two Party**: Slope = -10.55
- **Three Party, 1 SCM**: Slope = -6.97
- **Three Party, 2 SCMs**: Slope = -1.64
- **Three Party, 3 SCMs**: Slope = +2.37

Measures the *average number of messages required during D.*
Decomposing non-functional time:

- **Detection Latency** - delay in detection failure
- **Recovery Latency** – delay in restoring required services

-> Detection latency was dominant in 2-party case; in 3-party case, proportion of recovery latency increased as failure rate increased due to unavailability of SCMs
**Functional Effectiveness of Two-Party vs. Three-Party When All SMs of Each Type Can Fail**

Measures the *proportion of time the Service User* possesses the operational set of remote services needed to accomplish its task during $D$.

- **Two Party**
- **Three Party (3 SCMs)**
- **Three Party (2 SCMs)**
- **Three Party (1 SCM)**

Replacement case with 30 repetitions per data point.
**Conclusions and Future Work**

- Service discovery protocols possess basic capabilities to enable failure detection and recovery under conditions of node failure.
  - Results of experiments in node failure:
    - Three-party SCM is potential point of vulnerability at very high failure rates; reduced functional effectiveness
    - Effectiveness of three-party architecture approached level of two-party architecture as number of SCMs were added
    - Two-party architecture showed better efficiency than three-party architecture; redundant SCMs increases overhead (though subject to protocol variations in messaging)
    - Performance of both architectures can be improved by optimizing heartbeat mechanisms (registration refresh rate)

**Ongoing and Future Work**

- Repeat experiments with adaptable 3-party architecture that switches to 2-party mode when no SCMs can be found (SLP)
- Investigate robustness of service discovery strategies in larger scale environments.