Project AutoMate
Accord: Autonomic Composition

Manish Agarwal, The AutoMate Group
The Applied Software Systems Laboratory
Rutgers, The State University of New Jersey
http://automate.rutgers.edu

CAIP Autonomic Computing Tutorial/Workshop
June, 2003

Research Statement

Investigate issues and challenges in enabling dynamic service composition

Dynamic Service Composition

Ability to create a service plan at runtime based on available resources, user defined constraints, service objective and semantic description

Motivation

1. Enable on-demand construction of services
2. Increased degree of adaptability
3. Low cost solution
4. Short development time
5. Support autonomic behavior
   - minimum manual support
6. Extensible (support mutable interaction patterns)
   - new interactions and services can be introduced
   - existing interactions can be disabled
Simple scenario

- **Operations**
  - Merge or synchronize
  - Die
  - Divide
  - Add

- **Sample rules**
  - Move components and make them neighbors
  - Move them so that components are not neighbors
  - Add component
  - Remove component
  - Introduce new actors and interactions

Accord Composition Engine

Enables the construction, deployment and evaluation of autonomic service plans

Innovation?

1. Associate semantic information (keywords) with services
2. Using relational operations to support autonomic choreography (ad-hoc interactions)
3. Supporting and associating constraints with services and composition.

End-to-End Operation
Algorithm

[Initialization]
1. Each service description document is parsed for metadata
2. Semantic information is used to enhance service description
3. Composition request is made by the composer, consists of
   1. Composition Objective
   2. Semantic metadata
   3. Semantic threshold value i.e. the degree of correlation expected
   4. Constraints
   5. Start and target operations/services

[Selection]
1. Select appropriate services based on semantic matching
2. Executing constraints to refine services selection and composition

Algorithm

[Plan Generation]
1. All possible ad-hoc interactions are formulated
2. Service Graph is constructed, where
   1. Each operation acts as vertices of abstract graph
   2. The output argument types of operation are matched with the input parameters.
   3. If matching is correct, interaction link is created between operations
3. Constraints are executed to enable or disable inconsistent interactions
4. Initial and final operations are selected/specified based on semantic information of the composition and composition graph is created
5. Composition plan(s) is/are generated or status is returned
   1. Path in the interaction graph from source to destination operation corresponds to sequence of required message invocations.
   2. Operations lying on the path correspond to participating services.
   3. Scenarios where multiple composition plans can exist, the cost factor is evaluated for each path and least cost path is selected.

Architecture

ACCORD Composition Engine

- ACE Translator
- Graph Generator
- Constraint Analyzer
- Plan Generator

- Service Task
- Message Table
- Argument/Enviroment
Advantages

- Easy to create and configure:
  - Automated/Semi-Automated composition requires less effort than creating service manually
- Scalability
  - As the services become more ubiquitous, it’s not possible to consider all the permutations manually
- Adaptability
  - As new services become available and old services are removed from service pool, autonomic composition plans can be easily recomputed using ACE
- Evaluation and customized selection
  - Scenarios where multiple plans can exist, the cost of each plan can be pre-computed and most desirable plan selected

Challenges

- Guaranteed correctness
- Handle unreliable services
- Scalability
- Performance analysis of plans
- Constraints
  - Confluence (same final state irrespective of constraints exec order)
  - Observable determinism (actions are same)
  - Termination (cascaded constraints execution not allowed)
- Services or operations have (Solution: Constraints)
  - Multiple responses (sporadic data by sensor)
  - Multiple type of responses (array of virtual class)
- Services or operations have (Solution: "Void" type defined)
  - No response (?)
  - No input or output (?)

Conclusion and Future Work

Dynamic Service composition using ACE

- We talked about pre-invocation planning challenges - i.e. setting interactions autonomically, associating context, finding alternative plans, evaluating plans, etc.
- We found that by simple transformation of WSDL format of services to tables, power of relational algebra can be used along with semantics and constraints to create and evaluate compositions.
- In future ACE will provide composition policies to AutoMate invocation framework to deploy and do runtime management of composition instances
<table>
<thead>
<tr>
<th>Supporting Process Model</th>
<th>Context aware middleware and rules based control</th>
<th>Support for autonomic applications</th>
<th>Rules based tunability and adaptability</th>
<th>Dynamic Monitoring and Controlling Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSFL</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SING</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DROLL</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ACCORD</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WSDL compatibility</th>
<th>Rules based WSDL compatibility provides more features than WSDL</th>
<th>Supports QoS through Service Profile Class</th>
<th>No</th>
<th>No Uses WSEL extensibility elements to specify QoS</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSFL</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Uses WSEL extensibility elements to specify QoS</td>
</tr>
<tr>
<td>SING</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No Uses WSEL extensibility elements to specify QoS</td>
</tr>
<tr>
<td>DROLL</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No Uses WSEL extensibility elements to specify QoS</td>
</tr>
<tr>
<td>ACCORD</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No Uses WSEL extensibility elements to specify QoS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WSDL specification support</th>
<th>Full support for semantic description and discovery</th>
<th>Supports Process modeling using graph based process model and construct based process model</th>
<th>No</th>
<th>No No No No</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSFL</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Supports Process modeling using graph based process model and construct based process model</td>
</tr>
<tr>
<td>SING</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Supports Process modeling using graph based process model and construct based process model</td>
</tr>
<tr>
<td>DROLL</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Supports Process modeling using graph based process model and construct based process model</td>
</tr>
<tr>
<td>ACCORD</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Supports Process modeling using graph based process model and construct based process model</td>
</tr>
</tbody>
</table>

**Flow between services**

- **User defined Constraints**
  - runtime binding to a port based on runtime metadata and context information
  - **WSDL described Services**
  - **User defined and specified constraints** ...

**Constraint based, context aware runtime binding between different ports (implementing PortType) and managing flow of information**

- **Local binding**
  - Port binding
  - Abstract definition of service
  - ConstraintType(s)
  - Provider(s)
  - InMessage
  - OutMessage
  - Parts
  - Operation(s)
  - FaultMessage
  - Parts