Synthesizing Autonomic Compositions in Grid Environment

Manish Agarwal
Dr. Manish Parashar
Outline

- Introduction: Building Grid Applications on the fly
  - Motivation and challenges
- Composition problem
  - Issues and challenges of dynamic composition
- Objective and contribution of paper
- Existing approaches
- AutoMate
- Accord Composition Model
- Accord Composition Engine
  - Operation and Architecture
- Application Scenario: Travel Guide Service
- Discussion – Advantages and Limitations
- Conclusions and Future Work
Building Grid Applications On-the-fly

- **Drivers**
  - Economics
    - Low cost solution
    - Short development time
  - Autonomic behavior
    - Minimum manual support
    - Extensible and flexible
  - ...

- **Challenges**
  - Creating application using dynamic composition
  - Specifying and evaluating constraints and QoS
  - Fault tolerance, ...

Composition Problem

- **Composition:**
  - ability to take existing services (or resources) and combining and recombining them to solve new problems by creating new services.

- **Type**
  - **Static**
    - participating services and interactions are chosen a priori
  - **Dynamic**
    - Services / operations are dynamically selected and ad hoc interactions are enabled
Dynamic Composition: Issues & Challenges

- Guaranteed correctness
- Unreliable available underline services
- Scalability
- Performance Analysis/Ranking of Plans
- Constraints
  - Confluence
    - same final state irrespective of exec order
  - Observable determinism
    - actions are same
  - Termination
    - cascaded constraints execution not allowed
Existing Approaches

- **Rule based expert systems**
  - SWORD Project: Deal only with informational web services

- **Framework supporting Static Compositions**
  - GriPhyN Project: Chimera Virtual Data Systems
  - Symphony: based on Sun’s JavaBeans architecture
  - Meteor Project: focus on runtime management of static compositions
  - COSMOS and Aurora: architecture for e-services management

- **Dynamic compositions of stateless services**
  - DySCo (Piccinelli and Mokrushin): couples planning with invocation
Objectives and Contributions

- **Objectives**
  - finding and evaluating multiple alternate composition plans using different cost models and evaluation techniques
  - enabling ad-hoc interactions
  - using user defined **constraints** to model composition behavior
  - enhancing service description with **semantic information (keywords)**

- **Autonomic Composition Model**
  - Based on relational algebra and graph theory.

- **Accord Composition engine (ACE)**
  - Design and prototype implementation
  - Enables construction & evaluation of alternate composition plans
**AutoMate: Autonomic Computing Framework**

- **Autonomous Application Component**
- **Composition engine instance**
- **Components Aspect Metadata**
  - Access Control Layer
  - Rule and Policy Layer
  - Context Information

**Profile**
- **Functional Aspects**
- **Behavioral Aspects**
- **Control Aspects**
- **Context and Metadata**
- **Policy Database (Rule Base)**

**Composition Service**
- App-Specific Service
- Context Awareness

**Policy Engine Service**
- Rules Scheduling
- Rule Verification

**Other Middleware Services**
- Peer-to-Peer, Context Aware Middleware

**Autonomic Components and Dynamic Compositions**
Accord Composition Model

- **Main Ideas**
  - Relational Joins are used to choreograph ad hoc interactions at run time
  - Constraints are used to select valid interactions
  - Provide mechanism to evaluate and rank plans based on user defined cost models
  - Standard services description (WSDL) is enhanced with semantic metadata (keywords)
Accord Composition Model

A.) Composition is based on a service graph $G(S, L)$ where, $S$ is a set of available services and $L$ a set of possible interactions.

- Service set $S = \{s_i\}$ and each $s_i$ is associated with an ordered set of keywords, $\{K(s_i)\}$.
- Interaction set $L = \{l_{i,j}\}$ such that $s_i, s_j \in S$. Each interaction $l_{i,j}$ has a cost value $Cost(l_{i,j})$ associated with it.

B.) In the service graph, $G(S, L)$, the available services are vertices and interaction are edges. The edges are created at runtime using a relational join operation,

$$l_{i,j} \in s_i \bowtie s_j(s_i(\text{OutputMsg.ArgTypes}) = s_j(\text{InputMsg.ArgTypes})).$$
C.) The composer specifies composition description as initial service, $s_{\text{initial}}$, final service, $s_{\text{final}}$, an ordered set of keywords, $\{K_{\text{composition}}\}$ and a set of constraints, $C = \{c_k\}$.

D.) A subgraph of the service graph called composition graph $G'(S', L')$ is generated using inputs.

- $\forall i, s_i \in S' \iff K(s_i) \subseteq \{K_{\text{composition}}\}$.

- $\forall i, j, l_{i,j} \in L' \iff s_i \in S', s_j \in S'$ and $Valid(l_j) = True$.

E.) Dynamic Service Composition can be defined as finding a path from $s_{\text{initial}}$ to $s_{\text{final}}$ in $G'(S', L')$. 
Operation of Accord Composition Engine

Diagram showing the operation of the Accord Composition Engine with nodes labeled ACE Agent, Composer, Service Pool, and Standard Information Provider Service, connected by arrows indicating Discovery (GRIP) and Lookup (GRIP) processes.
- **Translator**
  - Parse WSDL description
  - Update relevant tables
  - Add semantic metadata

- **Graph Generator**
  - Define interactions links
  - Desirability factor is associated with link

- **Constraint Analyzer**
  - Valid links are selected
  - Represented as simple SQL queries

- **Plan Generator**
  - Appropriate plans are generated
  - Plans are ranked based on cost factor
Accord Algorithm in Action

1. Composition Request
   - Objective
   - Constraints
   - Semantic metadata

5. Best composition plan is returned

4. Composition plan is generated
   - using DFS or BFS

2. Connect and Select Services
   - based on constraints
   - based on keywords
   - based on input arguments

3. Create interaction links
   - using relational join
   - based on arguments types
Sample Scenario: Travel Guide Service

- **Objective**
  - User is interested in a service that looks up travel direction between two addresses

- **Available Services**
  - Driving Direction Service (DDS)
  - Vehicle-dependent Driving Direction Service (VDDS)
  - Location Service (LS)
## Service Pool for Travel Guide Service

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Input Argument</th>
<th>Arguments Type</th>
<th>Output Arguments</th>
<th>Output</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driving Direction (DDS)</td>
<td>SrcAddr, TgAddr</td>
<td>String, String</td>
<td>Driving Direction</td>
<td>String</td>
<td>Driving Direction, MapQuest</td>
</tr>
<tr>
<td>Location Service (LS)</td>
<td>Location</td>
<td>String</td>
<td>Address</td>
<td>String</td>
<td>Address, Landscape</td>
</tr>
<tr>
<td>Location Service (LS)</td>
<td>firstname, lastname, city</td>
<td>String, String, String</td>
<td>Address</td>
<td>String</td>
<td>Address, Name, City</td>
</tr>
<tr>
<td>Vehicle Dependent Driving Service</td>
<td>SrcAddr, TgAddr, Vehicle</td>
<td>String, String, String</td>
<td>Driving Direction</td>
<td>String</td>
<td>Vehicle, Driving Direction, Yahoo</td>
</tr>
</tbody>
</table>

- Service Interfaces converted into Standard WSDL format
- ACE Translator parses WSDL formats and maintain service table
- User associate semantic information with available operations
- User inputs the constraints.
Formation of Ad-hoc Interaction Links

- Relational Join on Service Table results in 4X4 interactions
- Self Loops are not allowed (4 interactions are invalid, i.e. DDS-DDS, etc)
- Inconsistent parameter matching and semantic information results in the elimination of 8 more inconsistent interactions.

\[ \{ \text{i.e. DDS-VDDS, VDDS-DDS, LS(landscape) –LS(Name, City), } \]
\[ \text{LS(Name, City)-LS(landscape), DDS – LS(landscape), DDS- LS(Name, City), VDDS- LS(landscape), VDDS-LS(Name, City)} \] \]
## Participating Services for Scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Service Request</th>
<th>Services Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Name-to-Driving-Direction-Service</td>
<td>Location Service, Location Service, Driving Direction Service</td>
</tr>
<tr>
<td>B</td>
<td>Vehicle-Dependent-Direction-Service</td>
<td>Location Service, Location Service, Vehicle Dependent Driving Service</td>
</tr>
<tr>
<td>C</td>
<td>Driving-Direction-Service</td>
<td>Location Service, Location Service, Driving Direction Service</td>
</tr>
</tbody>
</table>

### Source and Sink Services

<table>
<thead>
<tr>
<th>Possible Sink Services or Operations</th>
<th>Driving Direction Service, Vehicle Dependent Driving Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Source Services or Operations</td>
<td>Location Service(Landscape), Location Service (First name, Last name, City)</td>
</tr>
</tbody>
</table>
### Composition Graph Instances / Scenarios

**Scenario A**
- Service Request: Name-to-Driving-Direction-Service
- Invocation parameters: [First name, Last name, City], [First name, Last name, City]
- Description: Looks up driving directions between two persons homes given their name and cities

**Scenario B**
- Service Request: Vehicle-Dependent-Direction-Service
- Invocation parameters: Landscape, Landscape, Vehicle
- Description: Gives directions between two addresses as a function of available vehicle

**Scenario C**
- Service Request: Driving-Direction-Service
- Invocation parameters: Landscape, Landscape, Keywords
- Description: Returns driving directions between locations given constraints such as shortest path, avoiding highways, etc
Discussion – Advantages

- **Easy to create and configure**
  - requires less effort than creating service manually

- **Scalability**
  - not possible to consider all the permutations manually

- **Adaptability**
  - updates are easy

- **Efficiency and QoS assurance**
  - Allows evaluation and customized selection
    - cost of alternate plans can be pre computed and most desirable plan can be selected
  - No planning overhead during invocation
Implementation Challenges

- Services or operations have
  - Multiple responses (sporadic data dust by sensor)
  - Multiple type of responses (array of virtual class)
    - Solution: Constraints
- Services or operations have
  - No response (?)
  - No input or output (?)
    - Solution: “Void” type defined
- Exception Conditions
  - Composition plan does not exist
  - Composition request is incomplete (insufficient)
  - Constraints are invalid
  - Available service are not sufficient
  - Appropriate semantic information is not provided
  - Ambiguity in selecting start and sink operations
Conclusions

- We talk about pre invocation planning challenges i.e. setting interactions autonomic ally, 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.

- We present design and architecture of ACE that enables formulation and evaluation of different composition plans.

- In future ACE will provide composition policies to AutoMate invocation framework to deploy and do runtime management of composition instances.
Future Work

- **Security**
  - Trusted association between participating services

- **Ontology based description**
  - Currently cost, availability, response time, etc are described by simple keywords
  - Ontology based approach is required for better semantic matching and classification

- **Autonomic Middleware Services**
  - To support invocation of composed service

- **Performance Evaluation**
  - Difficult to evaluate QoS metrics