Enabling Autonomic Compositions in Grid Environments[*]

Manish Agarwal and Manish Parashar
The Applied Software Systems Laboratory
Department of Electrical and Computer Engineering
Rutgers University
{manishag,parashar}@caip.rutgers.edu

Abstract

In this paper we present the design, prototype implementation and operation of the Accord Composition Engine (ACE) that enables the dynamic and autonomic composition of Grid services. ACE builds on the Open Grid Services Architecture (OGSA) and autonomically synthesizes composition plans, when possible, from an available pool of services based on dynamically defined objectives and constraints. The key contribution is a dynamic composition model based on relational algebra and graph theory.

Keywords: Dynamic composition, Grid applications, Grid/Web services, Autonomic computing, OGSA.

1 Introduction

The Grid [9] is rapidly emerging as the dominant paradigm for wide area distributed computing. Its goal is to provide a service-oriented infrastructure [5] that leverages standardized protocols and services to enable pervasive access to, and coordinated sharing of geographically distributed hardware, software, and information resources. The fundamental concept underlying the emerging service oriented Grid architecture is the virtualization of entities as services and the seamless interactions and integration of these services. The Open Grid Service Architecture (OGSA) specification [8] defines standard interfaces and mechanisms for describing, invoking and managing Grid services. A service is defined as a network enabled entity that provides some capability and communicates through the exchange of messages. In OGSA, entities on the Grid are represented as services and new higher-level services and applications can be constructed from the available services. This motivates the need for a flexible and scalable dynamic service composition model.

Service composition can be simply defined as the process of taking existing services and combining them (based on user defined objective and constraints) to form new services. The composition model used by most existing Grid application development frameworks assumes that the composer has a priori knowledge of the composition objective, available services and their interaction patterns. In this model, the composer identifies relevant services, explicitly states their interactions and creates a composition "script". A flow engine then invokes this composition possibly using dynamic bindings to service instances. Unfortunately this relatively static composition approach is not very scalable. As the number of available services (resources, devices, applications) increase, a manual choreography of compositions and interactions is not very realistic. Furthermore the assumption that the composer has a priori knowledge about the composition objectives/requirements, the participating services and their interaction patterns is not valid for dynamic Grid environments and Grid applications. Finally, these models do not support autonomic on-demand compositions and require considerable human involvement.

A more dynamic composition model can address these issues. In such a model, composition plans are created at runtime based on dynamically defined composition objectives, their semantic descriptions, constraints, and available services and resources. Dynamic composition models are naturally suited for Grid environments where new services are constantly added and existing services are extended, modified or retired, and Grid applications where composition and interaction requirements are only known at runtime. These models can support autonomic (i.e. with minimum human support) behaviors and mutable interaction patterns where all services need not be known at design time and can be synthesized on-demand. However dynamic service composition presents significant challenges and requires addressing a number of critical issues such as discovering and identifying relevant services, formulating and ranking (and selecting) composition plans using current

[*]The research presented in this paper is supported in part by NSF via grants numbers ACI 9984357 (CAREERS), EIA 0103674 (NGS) and EIA-0120934 (ITR), and by DOE ASCI/ASAP (Caltech) via grant numbers PC295251 and 1052856.

context, goals, constraints and costs, binding to and invoking composition instances and checking their validity.

In this paper we present the Accord Composition Engine (ACE) that addresses dynamic service composition. The overall goal of ACE is to autonomically synthesize composition plans, when possible, from an available pool of services based on dynamically defined objectives and constraints. A key contribution is a dynamic composition model based on relational algebra and graph theory. Services are described using standard Web Service Description Language (WSDL) [6] and extended with semantic metadata (keywords). Relational joins are then used to generate composition plans and choreograph ad-hoc interactions at runtime, to satisfy the composer's objectives and constraints. Alternate plans may be evaluated and ranked using different cost criteria.

ACE is a key component of the Accord[1] composition framework that is part of Project AutoMate[2] [1]. The overall objective of AutoMate is to investigate key technologies to enable the development of autonomic Grid applications that are context aware and are capable of self-configuring, self-composing, self-optimizing and self-adapting. Specifically, it investigates the definition of autonomic components, the development of autonomic applications as dynamic composition of autonomic components, and the design of key enhancements to existing Grid middleware and runtime services to support the execution of these applications.

The rest of the paper is organized as follows. Section 2 presents related word. Section 3 presents an overview of dynamic composition and describes the Accord composition model. Section 4 presents the design, prototype implementation and operation of ACE. Section 5 presents a summary and conclusions.

2 Related Work

Composition models have received considerable attention in both academia and industry in recent years. Efforts within the Grid community that address composition aspects of workflow include Webflow [4], DAGMan [10], UNICORE [20] and XCAT [11]. Webflow is one of the earlier workflow systems and supports application composition in Grid environments. DAGMan is the meta-scheduler in Condor-G [10] and manages the dependencies between jobs. XCAT Application Factories [11] address workflow related issues for Grid-based components within the Common Component Architecture (CCA) [11] framework. Additionally, a number of composition and flow specification languages have been defined such as Grid Services Flow Language (GSFL) [24], Web Services Flow Language (WSFL) [14], XLANG [22], ebXML [7], and WSCI [2].

Composition and workflow has also been addressed by systems such as the Chimera Virtual Data System (GriPhyN) [13], Symphony [15], METEOR [17], COSMOS [12], Aurora [16], SWORD project [19] and DySCo [18]. The Chimera Virtual Data System (GriPhyN) [13] considers compositions as graphs of services. Unfortunately the overall service graph is static and assumes a priori knowledge of the participating services and their interaction patterns. Symphony [15] is a Java based composition and manipulation framework based on the Sun JavaBeans component architecture [21]. Its principle elements are a meta-program constructor and a back-end execution environment. Symphony supports only static compositions. METEOR [17] addresses runtime adaptability of a composed workflow. Its focus is primarily on runtime management rather than composition planning. COSMOS [12] and Aurora [16] are two examples of advanced architecture for e-service management. Once again, the main limitation of these systems is the rigidity in the interconnection and integration between services. SWORD [19] uses a rule-based expert system to find composition plans. Unlike our model, it only addresses informational services. DySCo [18] enables dynamic service composition and is based on the idea of functional incompleteness and multi-party orchestration. DySCo primarily address stateless e-services (unlike Grid services) and does not support constraint based management and control.

3 The Accord Composition Model

A key goal of the Grid is to provide ubiquitous resource and service availability. Furthermore, the Grid, by definition, is a dynamic and open environment where the availability and state of these services and resources are constantly changing. Emerging Grid applications are similarly complex, dynamic and heterogeneous. As a result, the ability to compose services (and applications) on the fly, based on currently available services, current context, and dynamically defined objectives and goals is critical. While the existing systems listed above do address many aspects of composition, they do not completely address the challenges of dynamic service composition. For example, the underlying composition approaches in these systems do not support dynamic definition of composition objectives and constraints, or ad hoc definition of service interactions and behaviors.

The primary focus of the Accord dynamic composition model presented in this paper is to autonomically synthesize composition plans (when possible) from the pool of available services to satisfy dynamically defined composition objectives, policies and constraints.

In Accord, composition objectives, and composition

[1]Autonomic Component, Compositions and Coordination
[2]http://automate.rutgers.edu

Steps	Actions
1	For every available service in service pool
	Standard description document(wsdl) is parsed for metadata
	User provide semantic description and context information
	Semantic information is supplemented by scanning the objective for keywords
2	Composer Initializes composition instance
	Associate constraints with composition
	Set composition context and Objective(K)
	Set semantic threshold value (i.e. degree of correlation between composition description and available services)
3	Ad-hoc and autonomic interactions (L) are constructed between available services (S)
	Corresponding to each selected service
	Metadata (input, output argument types, keywords, operations) is extracted
	Service Graph G(S, L) is created
	Each available operation act as vertices of the service graph
	All possible ad-hoc interactions are formulated
	If output arguments types of operation matches with the input parameter types
	Interaction link is created between the operations
4	Appropriate services are selected based on semantic matching
	Constraints are executed to further refine service selection process (S')
	Source operation, Target operation of composed service is selected or specified
5	Constraints are executed to select consistent interactions (L')
	Cost function Cost(L'), is used to associate desirability of each interaction in (L')
6	Composition plan(s) is generated by finding path(s) in Composition Graph G'(S',L')
	Cost or desirability of each path is calculated (based on user specified criteria)
	Ranking of multiple paths is done based on cost function for each path
	Failure in finding a path indicates either insufficient information or infeasible composition objective

Table 1. Accord Composition Algorithm.

policies and constraints ($\{c_k\}$) can be dynamically defined as simple SQL statements. The pool of currently available services (service pool) is represented as a graph where the nodes represents services, s_i, in the pool and the links, $\{l_{i,j}\}$, can be modeled as possible interactions. Service descriptions are augmented with semantic information in the form of keywords and context information ($\{K(s_i)\}$). This semantic information along with polices and constraints are used to select applicable services ($s^{'}$) and interactions ($l_{i,j}$ where $Valid(l_{i,j}, \{c_k\}) = True$). Candidate composition plans can be represented as paths in this graph $G^{'}(S^{'}, L')$. Alternate plans may be evaluated and ranked based on different cost factors. The Accord dynamic composition model can be formally defined as follows:

1. 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.
2. 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, $l_{i,j} \in s_i \bowtie s_j (s_{i(OutputMsg.ArgTypes)=s_{j(InputMsg.ArgType)}})$.
3. The composer specifies composition description as initial service, $s_{initial}$, final service, s_{final}, an ordered set of keywords, $\{K_{composition}\}$ and a set of constraints, $\{c_k\}$.
4. A subgraph of the service graph called composition graph $G^{'}(S^{'}, L^{'})$ is generated using these inputs as follows:
   - $\forall\, i, s_i \in S^{'} \iff K(s_i) \subseteq \{K_{composition}\}$.
   - $\forall\, i, j, l_{i,j} \in L^{'} \iff s_i \in S^{'}, s_j \in S^{'}$ and $Valid(l_{i,j}, \{c_k\}) = True$.
5. Dynamic Service Composition can be defined as finding a path from $s_{initial}$ to s_{final} in $G^{'}(S^{'}, L')$.

The complexity of the plan generation algorithm is $O(S^{'} + L^{'})$. Note that the model defined above assumes that the composer will provide a proper set of constraints, and the set of constraints will satisfy properties of confluence, termination and observable determinism.

The Accord composition algorithm is presented in Table 1. In the initialization and service selection step, the services in the current service pool are parsed to generate

Service Name	Input Argument	Arguments Type	Output Arguments	Output	Keywords
Driving Direction (DDS)	SrcAddr, TgtAddr	String, String	Driving Direction	String	Driving Direction, MapQuest
Location Service (LS)	Location	String	Address	String	Address, Landscape
Location Service (LS)	firstname, lastname, city	String, String, String	Address	String	Address, Name, City
Vehicle Dependent Driving Service	SrcAddr, TgtAddr, Vehicle	String, String, String	Driving Direction	String	Vehicle, Driving Direction, Yahoo

Table 2. Service Pool for the Example Travel Guide Service.

Source Operation	Target Operation	Comment
Location Service (Landscape)	Driving Direction Service	Location Service, provides address to Driving Direction Service
Location Service (Landscape)	Vehicle Dependent Direction Service	Location Service provides address to Vehicle Direction Service
Location Service (Name, Name, City)	Vehicle Dependent Direction Service	Name-Location Service provides address to Vehicle Direction Service
Location Service (Name, Name, City)	Driving Direction Service	Name-Location Service provides address to Driving Direction Service

Table 3. Interaction Table for the Example Travel Guide Service.

service set S. A relational join operation is then used to construct the set of ad-hoc interactions, L, by matching interfaces, and to create service graph G. The composer (the user or an agent) specifies a composition request as a set of constraints $\{c_k\}$, keywords ($\{K_{composition}\}$), input service ($s_{initial}$) and output services (s_{final}). The keyword set and constraint set are used to select the participating services, $S^{'}$, generate the set of associated interactions $L^{'}$, and the composition graph $G^{'}$. Cost associated with each $l^{'}_{i,j}$ is calculated. Candidate composition plans can now be generated as paths in $G^{'}$ between $s_{initial}$ and s_{final} using graph path algorithms (DFS, BFS). The composition plans can be ranked based on costs. These costs could reflect economic factors, operational environments and/or user defined factors. Constraints can belong to different categories and can control aspects of both services and compositions. Examples of constraint categories include security constraints, behavioral constraints and integrity constraints.

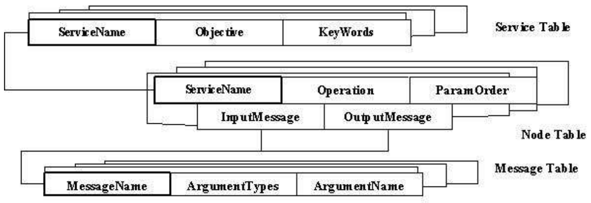


Figure 1. Schemas for ACE Service Pool Tables.

To illustrate the operation of the Accord composition model consider a scenario where a user requests a travel guide service that provides a travel route between two locations. The set of available services includes a *Driving Directions Service (DDS)* that simply returns driving directions between two specified addresses, a *Vehicle-dependent Driving Direction Service (VDDS)* that returns directions as a function of the specified vehicle (e.g. car, train, boat, bicycle), and a *Location Service (LS)* that returns the exact address given an approximate location. The service pool and interaction table for this example are shown in Table 2 and Table 3 respectively. In this scenario, if a service request has exact endpoint addresses, the service DDS is directly invoked. If one or both of the endpoints in the request are not exact, the composition of LS and DDS is required. If vehicle information is included in the request, then VDDS is invoked instead of DDS. The decision to include or exclude any service is based on the service request and specified constraints. The interaction links between the selected services are also generated at runtime.

4 The Accord Composition Engine

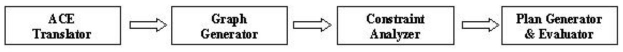


Figure 2. Architectural Overview of ACE.

4.1 Architecture Overview

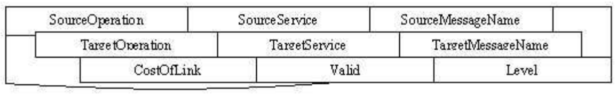
An architectural overview of the Accord Composition Engine (ACE) is presented in Figure 2. ACE can be a part of composition services available on the Grid or composition agents within the Grid middleware. It builds on OGSA [8] and the emerging Grid middleware. A service in ACE corresponds to a Grid service as specified in the Grid Service Specification [23] and is described using WSDL. The description field is used to add semantic information in the form of keywords describing the service. A *service pool* is the set of services that are available to a composer. The current service pool is defined by a *Node Table*, *Message Table* and *Service Table* which are constructed dynamically using existing OGSA discovery mechanisms such as SQUID [1], MDS [23] or UDDI [3]. The ACE architecture consists of four key modules: ACE translator, Graph Generator, Constraint Analyzer, Plan Generator and Evaluator. These modules are described below.

4.1.1 ACE Translator

The ACE translator modules parses the WSDL service description for each service in the current service pool and uses this information to update the relevant tables. It creates a row in the *Node Table* corresponding to each “operation” in this description, which contains the service name, operation name, ordered sequence of input parameters, input message name and output message name. For each message name, a separate entry is created in the *Message Table* with the message name as primary key. Each message entry also contains argument names and argument type attributes. The schemas for these tables are presented in Figure 1.

4.1.2 ACE Graph Generator

The ACE Graph Generator module is responsible for defining the interaction links between services in the service pool using relational joins. This is done based on the message description in the *Message Table*. If the arguments and attribute types associated with the output message of a source operation is a superset of the arguments associated with the input message of a target operation, then a directed edge exists from source operation to target operation. Corresponding to each such link, an entry is created in the *Link Table*. The attributes of *Link Table* are the source operation name, source service name, source message name, destination operation name, destination service name, destination message name, cost of the link (defined by the context), level of composition (in cases where composition span across multiple service pools), and a valid flag that is true if the current link is active. The schema for the *Link Table* is shown in Figure 3.



SourceOperation	SourceService	SourceMessageName
TargetOperation	TargetService	TargetMessageName
CostOfLink	Valid	Level

Figure 3. Schema for ACE Link Table.

4.1.3 ACE Constraint Satisfaction Module

The Constraint Satisfaction Module is responsible for evaluating and executing the constraints associated with individual services and service composition requests. In ACE, constraints are represented by simple SQL expressions that modify the validity of interaction links. Thus the ACE constraint satisfaction module operates on *Link Table* and enables or disables link entries in the table.

4.1.4 Service Plan Generator and Evaluator

The dynamic service plan generator and evaluator module is responsible for generating composition plans in response to a composition request. It works in conjunction with Constraint Satisfier Module and operates on the *Link Table*. A plan is an ordered set of services and their interactions that can satisfy the request. Service and link costs are used to rank plans when multiple plans exist.

4.2 ACE Operation

In this section, we use travel guide service example to illustrate the working of the ACE algorithm and our current prototype. The overall end-to-end operation of ACE is shown in Figure 4. Service composition is initiated when a service request (objectives, constraints) is presented to the ACE agent. The ACE agent uses the composition model and algorithm presented in this paper to synthesizes one or more composition plan(s) consisting of a set of participating services and the interactions between them. For the discussion below consider a request for a service that provides driving directions between two addresses. Possible composition scenarios for the service pool in Table 2 are presented in Table 4. The autonomic service composition process is presented below.

- ***Step 1***: The composer (user, agent, service) makes a composition request to the ACE agent. For example, the request may be for a “Name to Driving Direction Service”, where the user provides the name and city for the two endpoints and service is expected to return driving directions between them. In a variation of this request, the user may also specify the vehicle as a part of service request. In yet another scenario, the user can provide additional constraints, for example, that the *Yahoo Maps* service must be used, or the shortest route that avoids all highways must be found. Sample composition scenarios for this example are listed in Table 4.

- ***Step 2***: Once the ACE agent receives the composition request, it contacts the service pool (see Figure 4) to get the list of services that are currently available. The ACE Translator then parses the standard service descriptions (WSDL) of the available services, extracts relevant metadata, and stores it in a tabular format using the schemas presented in Figure 1. Note that services in the service pool may belong to different directories and may be provided by different providers.

  The user may provide additional semantic information for each service entry in the table. This information is used to support advance querying and search operations. A sample snapshot of service table for our example is presented in Table 2.

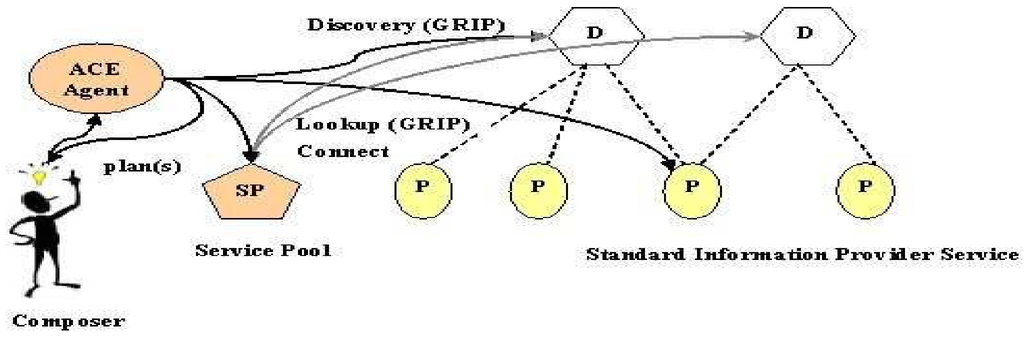


Figure 4. Operation of the Accord Composition Engine.

Scenario	Service Request	Invocation parameters	Description
A	Name-to-Driving-Direction-Service	[First name, Last name, City], [First name, Last name, City]	Looks up driving directions between two persons homes given their name and cities
B	Vehicle-Dependent-Direction-Service	Landscape, Landscape, Vehicle	Gives directions between two addresses as a function of available vehicle
C	Driving-Direction-Service	Landscape, Landscape, Keywords	Returns driving directions between locations given constraints such as shortest path, avoiding highways, etc

Table 4. Sample Composition Requests.

- ***Step 3***: The ACE Graph Generator processes the composition request and selects the appropriate services using semantic matching based on the keywords. Selected services for different service composition scenarios are listed in Table 5. A relational join operation is then used to construct the set of interaction links. Cost of each the service and interaction is specified or evaluated. Finally a service graph is created.

- ***Step 4***: The ACE Constraint Module creates a composition graph from the service graph using the set of constraints (C) defined by the user. In our example, the valid interaction links for the composition graph are presented in Table 3. The candidate composition plans are generated as paths in the composition graph. Some simple scenarios for our composition request (see Table 4) are illustrated in Figure 5.

Finally, the composition plan(s) is(are) generated by the agent and returned to the composer. In cases where multiple plans are generated, the plan costs are used to rank the plans. The composition request fails if (1) a plan does not exist, (2) the composition request is insufficient, or (3) the constraints are invalid. The first case occurs when no sequence of services exists for the current pool of services that can satisfy the request. This situation may be handled by increasing the number of available services in service pool and lowering the degree of semantic keyword matching. In the second case, the composer can be asked for additional specifications for the composition. For the third case, ACE currently assumes that the constraints specified by the composer are valid, i.e. they exhibit the property of confluence (have the same effect irrespective of their execution order), observable determinism (actions are same) and termination (cascaded constraints execution not allowed). If the specified constraint set does not satisfy these properties, ACE will fail to generate a valid plan.

4.3 Advantages and Limitations

In Grid environment, composite service creation is not necessarily a one-time effort. Composition may need to adapt to the changes in the environment and underlying resources. Moreover as the services become more ubiquitous, it is not possible to consider all the permutations manually. Thus involving end users in service composition is unacceptable, creating a need for systems such as ACE that enable the construction of autonomic service composition plans. ACE also provides the mechanism to rank different plans and select the most appropriate one. An additional advantage of generating multiple plans is redundancy and fault tolerance. If one plan fails, an alternate plan can be invoked, or multiple plans can be used simultaneously for reliability or QoS.

Dynamic service composition is extremely challenging and requires addressing a number of critical issues such as guaranteed correctness, scalability, performance analysis, and constraints analysis. In traditional service environments, response time depends primarily on resource latencies and network loads. With dynamic service composition, planning time can become an additional overhead. As a

Scenario	Service Request	Services Selected
A	Name-to-Driving-Direction-Service	Location Service, Location Service, Driving Direction Service
B	Vehicle-Dependent-Direction-Service	Location Service, Location Service, Vehicle Dependent Driving Service
C	Driving-Direction-Service	Location Service, Location Service, Driving Direction Service

Table 5. Participating Services For Composition Scenarios.

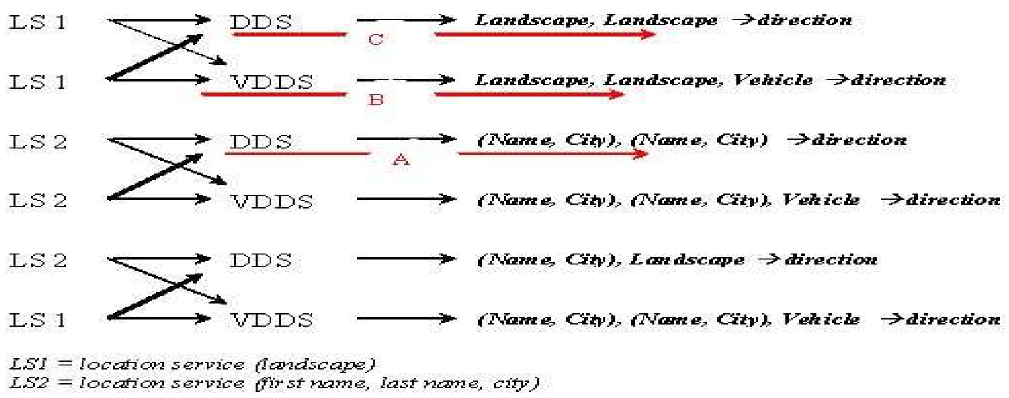

LS 1
DDS
C
Landscape, Landscape →direction
LS 1
VDDS
B
Landscape, Landscape, Vehicle →direction
LS 2
DDS
A
(Name, City), (Name, City) →direction
LS 2
VDDS
(Name, City), (Name, City), Vehicle →direction
LS 2
DDS
(Name, City), Landscape →direction
LS 1
VDDS
(Name, City), (Name, City), Vehicle →direction
LS1 = location service (landscape)
LS2 = location service (first name, last name, city)

Figure 5. Composition Graph Instances.

result composition planning mechanisms must be very efficient. Another important challenge is in ensuring guaranteed correctness. In many cases, it may not be possible to find any guaranteed correct plan for a composition request. ACE specifically provides no such guarantee and is based on the notion that "uncertain plan" is better than no plan. In static composition, the process is bound with the service at design time and designer can evaluate the performance metrics associated with it. However, in dynamic composition the binding is not possible until the plans are found and invoked. In ACE, the ranking of different plans is done based on costs rather than performance data. Other challenges that need to be addressed include missing or no inputs and outputs, multiple service responses or multiple responses types.

5 Summary and Conclusion

This paper addressed issues and challenges in enabling dynamic service composition on the Grid. We present the design and prototype implementation of the Accord Composition Engine (ACE). The ACE composition model enables autonomic generation of composition plans, when possible, from available pool of services based on dynamically defined objectives and constraints. It enhances the standard (OGSA) service descriptions with semantic metadata, and uses this metadata along with the current context, dynamically defined composition objectives and constraints and relational algebra to choreograph ad-hoc interactions and composition plans at runtime. Alternate plans may be evaluated and ranked based on different cost factors. The main motivation is to create composition on-demand at runtime.

References

[1] M. Agarwal, V. Bhat, Z. Li, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, M. Parashar, B. Khargharia, and S. Hariri. AutoMate: Enabling Autonomic Applications on the Grid. In *Proc of Autonomic Computing Workshop, 5th Annual International Active Middleware Services Workshop(AMS 2003)*, pages 365–375, Seattle, WA, June 25 2003.

[2] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek, August 2002. Web Service Choreography Interface (WSCI) 1.0, http://www.w3.org/TR/wsci/.

[3] T. Bellwood. UDDI (Universal Description Discovery and Integration) Version 2.04 API Specification. http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm, July 19, 2002.

[4] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, and G. PremChandran. WebFlow : A Visual Programming Paradigm for Web/Java Based Coarse Grain Distributed Computing. *Concurrency: Practice and Experience*, 9(6):555–577, 1997.

[5] M. Champion, C. Ferris, and E. Newcomer, November 14, 2002. Web Services Architecture., http://www.w3.org/TR/ws-arch.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, 15 March, 2001. Web Services Description Language (WSDL) 1.1.http://www.w3.org/TR/wsdl.

[7] ebXML Requirements Team, May 8, 2001. ebXML Requirements Specification, Version 1.06 , http://www.ebxml.org/specs/ebREQ.pdf.

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid:An Open Grid Services Architecture for Distributed Systems Integration. In *Open Grid Service Infrastructure WG,Global Grid Forum*, June 22 2002.

[9] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 201.

[10] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC10)*, pages 7–9, San Francisco, CA, August 2001.

[11] M. Govindaraju, S. Krishnan, K. Chiu, A. Slominski, D. Gannon, and R. Bramley. XCAT 2.0: A Component-Based Programming Model for Grid Web Services. Technical report-tr562, Dept. of C.S., Indiana Univ, June 2002.

[12] F. Griffel, M. Boger, H. Weinreich, W. Lamersdorf, and M. Merz. Electronic Contracting with COSMOS - How to Establish, Negotiate and Execute Electronic Contracts on the Internet. In *2nd Int. Enterprise Distributed Object Computing Workshop (EDOC '98)*, 1998.

[13] M. Wilde, I. T. Foster, J. Vckler and Y. Zhao. Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In *SSDBM 2002*, pages 37–46.

[14] F. Leymann. Web Services Flow Language (WSFL) 1.0. http://www-3.ibm.com/software/solutions/webservices/pdf/ WSFL.pdf, IBM, May,2001.

[15] M. Lorch and D. Kafura. Symphony : A Java-based Composition and Manipulation Framework for Computational Grids. In *Proc. of 2nd IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, pages 136–143, Berlin, Germany, 2002.

[16] M. Marazakis, D. Papadakis, and C. Nikolaou. Aurora: An Architecture for Dynamic and Adaptive Work Sessions in Open Environments. In *Proc of the International Conference on Database and Expert System a Applications (DEXA'98)*, Springer-Verlag LNCS Series, 1998.

[17] J. Miller, D. Palaniswami, A. Sheth, K. Kochut, and H. Singh. WebWork: METEOR's Wen-based Workflow Management System. *Journal of Intellegent Information Systems*, 10(2):185–215, 1998.

[18] G. Piccinelli and L. Mokrushin. Dynamic e-service composition in DySCo. In *Proc of 21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01)*, Mesa, Arizona, April 16 - 19 2001.

[19] S. R. Ponnekanti and A. Fox. SWORD: A Developer Toolkit for Web Service Composition. In *11th World Wide Web Conference (Web Engineering Track)*, Honolulu, Hawaii, May 7-11 2002.

[20] M. Romberg. The UNICORE Grid Infrastructure. *Scientific Programming, Special Issue on Grid Computing*, 10(2):149–157, 2002.

[21] Sun Microsystems Inc., July, 2002. The JavaBeansTM Component Architecture.

[22] Satish Thatte, December, 2001. XLANG: Web Services for Business Process Design, http://www.gotdotnet.com/team/xlang-c.

[23] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman, February 2002. Grid service specification.

[24] Patrick Wagstrom, Sriram Krishnan, and Gregor von Laszewski. GSFL: A Workflow Framework for Grid Services. In *SC'2002*, pages 11–16, Baltimore, MD, November 2002.