An Infrastructure for Dynamic Composition of Grid Services

Zhen Li and Manish Parashar
The Applied Software Systems Laboratory
Dept. of Electrical and Computer Engineering, Rutgers University, USA
{zhljenny, parashar}@caip.rutgers.edu

I. Introduction

As Grid computing has evolved, multi-organization collaboration has shifted from primarily file exchange to decentralized, seamless and scalable service-oriented approaches [3]. Dynamic service composition that enables an application or parts of an application to be dynamically composed from existing Grid elments/services is emerging as a desired paradigm. However, enabling dynamic service composition requires infrastructure support for element discovery and workflow composition and enactment. This poster presents Rudder, an agent-based composition infrastructure for Grid services using associative composition space abstractions. Rudder supports dynamic workflow enactment using software agents to semantically discover and select services required by the workflow, and to compose the selected services. Further, the agents manage the workflow at runtime and adapt the workflow reactively to address the dynamism of the Grid execution environment and application requirements. The composition space abstractions in Rudder are realized by the Comet substrate [1], which provides a robust and scalable, decentralized shared-space for wide area environments. Comet supports a two level shared space abstraction for workflow composition and execution: a global persistent space supports scalable registration and semantic discovery of services; and dynamically constructed contextually local spaces provide the communication medium to support workflow enactment.

II. Architecture

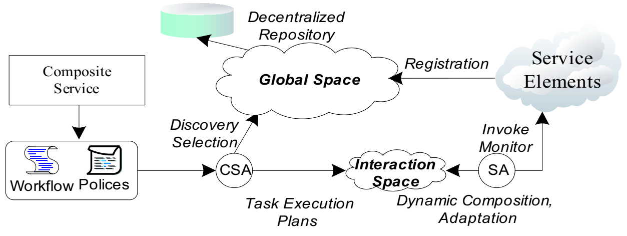

Fig. 1. A conceptual overview of the Rudder infrastructure.

A conceptual overview of the Rudder infrastructure is shown in Figure 1, and consists of 4 key components:

- *GlobalSpace*, which is a persistent space that supports the registration, publishing and semantic discovery of service elements. A service provider can access the *GlobalSpace* to register and publish its service in Rudder. Details of the semantic discovery and dynamic selection have been presented in [2].
- *InteractionSpace*, which is a dynamically constructed contextually localized space that is dedicated to a particular workflow. This space provides the interaction/coordination medium for configuring and enacting a workflow, and only includes the providers of services that are a part of the workflow.
- *Service agent (SA)*, which manages a service element and the execution of the workflow task assigned to that service. This agent is also responsible for monitoring the service and enforcing the workflow adaptation to respond to application/system dynamics based on specified adaptation policies.
- *Composition agent (CSA)*, which manages one or more workflows. This agent is responsible for discovering service elements to perform workflow tasks, instantiating a SA to manage each discovered service element, dynamically negotiating with the SAs to select elements based on system context and user preferences, and generate task execution plans to enact the workflows.

III. Dynamic Workflow Composition and Adaptation in Rudder

Rudder views dynamic composition, as the runtime configuration, enactment, and adaptation of workflows. In Rudder, a service workflow is specified using XML, along with adaptation policies that define how a task or the workflow may be adapted during execution to satisfy non-functional requirements, and enacted using two types of composition tuples. A *plan tuple* describes a workflow task execution plan and is generated by the CSA. It consists of 5 fields: name, which identifies the task in workflow; status, which can be “Activated” or “Stopped”; dependencies, which specify the task’s relationships to its immediate predecessor(s) and successor(s); SAlist, which includes the selected SA identifier and all candidate SA identifiers; and adaptation policy, which specifies how to adapt the workflow in response to specific events. A *task tuple* has 3 fields: name, predecessor, and input data (host and port). A task tuple can be retrieved using a matching task template. The task tuples and templates are generated by the SAs based on task execution plans. Workflow composition and adaptation in Rudder is enabled by CSA and SA agents using the composition tuples as described below.

A. Dynamic Workflow Composition

Dynamic workflow composition in Rudder consists of 3 phases. Figure 2 illustrates the *discovery and selection* phase, which has 4 steps. The user submits the workflow and adaptation polices of the service. A CSA is instantiated to process the user inputs. It parses the workflow specification and generates a discovery request for each workflow task. It then uses the *GlobalSpace* to discover a group of candidate services for each task, and instantiates a SA for each discovered service. The CSA negotiates with each SA group to dynamically select an appropriate service and marks others as backups. Finally, the CSA generates a task execution plan for each workflow task.

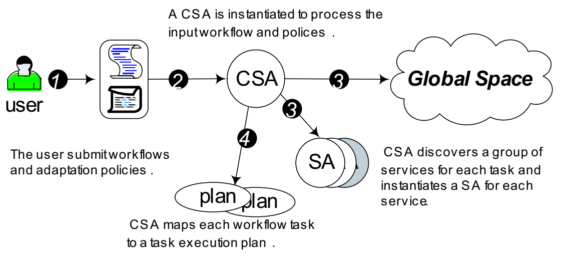
A CSA is instantiated to process the inputworkflow and polices .
user
CSA
Global Space
SA
plan
plan
The user submitworkflows and adaptation policies .
CSA discovers a group of services for each task and instantiates a SA for each service.
CSA maps each workflow task to a task execution plan .

Fig. 2. Phase 1: Discovery and selection.

Figure 3 illustrates the *composite service setup* phase. The CSA initiates a transient Comet space as the *InteractionSpace*, and invites all the selected SAs to join this space. Each SA accepts the invitation and executes the Comet join protocol [1]. After joining the space, the SA locally accesses it to read its task execution plan tuple. Once all SAs have joined, the CSA inserts the “Activated” plan tuples into the space, which will be extracted by the corresponding SAs. The SAs generate the task templates and wait for their task tuples.

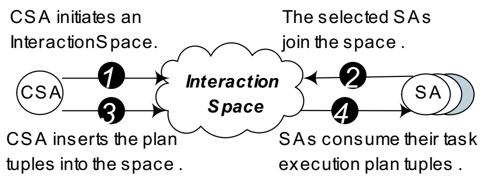
CSA initiates an InteractionSpace.
The selected SAs join the space .
CSA
Interaction Space
SA
CSA inserts the plan tuples into the space .
SAs consume their task execution plan tuples .

Fig. 3. Phase 2: Composite service setup.

Figure 4 illustrates the *workflow execution* phase. The CSA inserts the first task tuple to start the execution, and waits for an end task tuple or a “Stopped” plan tuple. The SAs then coordinate to execute the workflow by taking and writing task tuples. The SA corresponding to the last task inserts an end task tuple, which will be consumed by the CSA. Finally, the CSA collects the output data and returns to user. Once the composite service is permanently terminated, the CSA informs the infrastructure to destroy the *InteractionSpace* and the SAs.

B. Workflow Adaptation

The SA supports workflow adaptation using element switching, which allows a failed or an active element to be replaced with a single element at run time to obtain good performance.

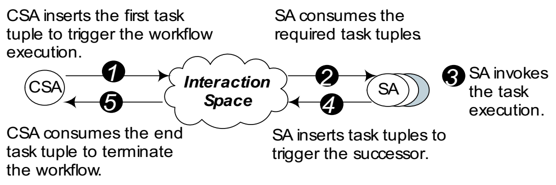
CSA inserts the first task tuple to trigger the workflow execution.
SA consumes the required task tuples.
CSA
Interaction Space
SA
SA invokes the task execution.
CSA consumes the end task tuple to terminate the workflow.
SA inserts task tuples to trigger the successor.

Fig. 4. Phase 3: Workflow execution.

A SA has 3 main components, shown in Figure 5. The *task manager* accesses the *InteractionSpace* and manages task execution. It extracts the plan tuple from the space, generates the task templates, dispatches the retrieved task, generates the resulting task tuples, and inserts them into the space. The *element controller* monitors the element and invokes the task execution. If execution is successful, it returns the results to the *task manager*. The *element proxy* forwards the task request from the *task manager* to a replacement candidate element. It dynamically selects a candidate SA as a replacement, sends it a task execution request, and forwards the results to the *task manager*.

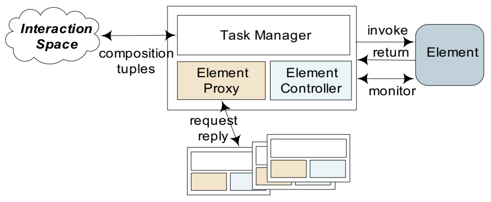

Fig. 5. The structure of a service agent (SA).

IV. Conclusion

This poster presented Rudder, an agent-based infrastructure for the dynamic composition of Grid services. Rudder is based on an associative composition space abstraction, and supports dynamic workflow enactment using software agents. The agents semantically discover and select services required by the workflow, and compose the selected services. The agents manage the execution of the workflow at runtime, and adapt the workflow reactively. Prototypes of Rudder and the Comet substrate have been developed and deployed as Java services within the JXTA framework.

Acknowledgment

The research presented in this paper is supported in part by National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826, and by Department of Energy via the grant number DE-FG02-06ER54857.

References

[1] Z. Li and M. Parashar. Comet: A scalable coordination space for decentralized distributed environments. In *Proceedings of the 2nd International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 104 – 112, 2005.
[2] Z. Li and M. Parashar. Rudder: An agent-based infrastructure for autonomic composition of grid applications. *Multiagent and Grid Systems - An International Journal*, 1(3):183 – 195, 2005.
[3] M. Parashar and J. Browne. Conceptual and Implementation Models for the Grid. In *Proceedings of the IEEE, Special Issue on Grid Computing*, volume 93, pages 653–668, 2005.