Rudder: An Agent-based Infrastructure for Autonomic Composition of Grid Applications

Zhen Li *and Manish Parashar
The Applied Software Systems Laboratory
Dept. of Electrical and Computer Engineering, Rutgers University,
Piscataway NJ 08854, USA
Email: {zhljenny, parashar}@caip.rutgers.edu

September 26, 2005

Abstract

This paper describes Rudder, a decentralized agent-based infrastructure for supporting the autonomic composition of Grid applications. Rudder provides agents and protocols for discovering, selecting, and composing elements. It also implements agent interaction and negotiation protocols to enable appropriate application behaviors to be dynamically negotiated and enacted. The defined protocols and agent activities are supported by Comet, a scalable decentralized shared-space based coordination substrate. The implementation, operation and experimental evaluation of the system are presented.

Keywords: Autonomic composition, self-managing Grid applications, agent-based system, decentralized coordination substrate.

*Corresponding author, tel: (732) 445-3807, fax: (732) 445-0593, email: zhljenny@caip.rutgers.edu

1 Introduction

The goal of the Grid infrastructure is to enable a new generation of applications that combine intellectual and physical resources spanning multiple organizations and disciplines, and provide vastly more effective solutions to scientific, engineering, business and government problems [20]. As Grid computing has evolved, the collaborative problem solving enabled by the Grid has also evolved from primarily file exchange to direct access to hardware, software and information components. The resulting Grid applications, which are based on seamless discovery, access to, and interactions among resources and services, have complex and highly dynamic computational and interaction behaviors, and when combined with the uncertainty of the underlying infrastructure, result in significant development and management challenges. The defining characteristics of these emerging systems and applications include: (1) *heterogeneity*, Grid environments and applications aggregate large numbers of computational and information resources; (2) *dynamism*, the computation, communication and information environment is continuously changing during the lifetime of an application, including the availability and state of resources and services; and (3) *uncertainty* caused by multiple factors, including dynamism, which introduces unpredictable and changing behaviors that can only be detected and resolved at runtime, failures, which have an increasing probability of occurrence as the system scales increase, and incomplete knowledge of global system state [20]. Autonomic development and management strategies, which are inspired by biological systems and the human autonomic nervous system, have recently been proposed [10, 11, 21] to address these challenges.

A key issue in the development and management of these autonomic applications is the autonomic composition of the distributed autonomous elements. A single available element might not address specific application requirements, and composing several elements to form a unit with integrated functionalities is necessary. Autonomic composition enables applications or parts of an application to be dynamically composed from discrete elements to meet the changing requirements and system behaviors, deal with element failures, optimize performance, address QoS constraints, etc. However, enabling autonomic composition is difficult in Grid environments because the available elements are typically numerous, heterogeneous, and available in a dynamic on-demand manner. Challenges include element descriptions, their discovery, and their dynamic and adaptive composition, interaction and coordination. Recent initiatives, such as the "Semantic Grid" [26], complement the Grid service-oriented architecture [9] to enhance the scientific process with seamless interaction on a global scale. Effectively, the composition of loosely coupled Grid services is emerging as the desired paradigm for constructing Grid applications. In this context, solutions being developed as part of the "Semantic Web" [1] can be leveraged to support accurate Grid service description, discovery, and composition. However, an effective autonomic composition infrastructure is still absent in Grid environments.

This paper presents Rudder, a decentralized agent-based infrastructure ad-

dressing autonomic composition for self-managing Grid applications. Rudder consists of a distributed agent framework [13] and the Comet decentralized coordination substrate [14]. The agent framework provides agents that enact a workflow-based composition model in a dynamic and negotiated manner. Each element in Rudder is modeled as an atomic service with a semantic description using the OWL-S [19] language. Discovery and interaction protocols are provided for registering and unregistering elements, dynamically searching and selecting elements, and negotiating the properties of composed elements. Comet, a fully decentralized shared-space coordination substrate, provides the core services for connecting agent networks and scalably supporting various agent interactions, such as mutual exclusion, consensus, and negotiation.

The rest of this paper is organized as follows. Section 2 outlines the requirements, challenges, and current approaches to the autonomic composition of Grid applications. Section 3 introduces the Rudder agent-based composition infrastructure. Section 4 describes the workflow-based autonomic composition mechanism supported by Rudder. Section 5 describes the implementation and operation of Rudder and presents an experimental evaluation. In section 6, some related work are introduced. Section 7 presents a conclusion.

2 Autonomic Composition of Grid Applications: Requirements and Current Approaches

As outlined above, the inherent scale, complexity, heterogeneity, and dynamism of emerging Grid environments and applications result in significant programming and runtime management challenges. As a result, developing Grid applications requires redefining Grid programming frameworks and middleware services. Specifically, it requires that static (defined at the time of instantiation) application requirements, their structures and system and application behaviors be relaxed, and that the behaviors and structures of elements and applications be sensitive to the dynamic state of the system and the changing requirements of the application and be able to adapt to these changes at runtime. Clearly, enabling autonomic composition of elements and applications is a key issue in effectively addressing these requirements.

Enabling autonomic composition requires conceptual frameworks and an implementation infrastructure. Conceptual frameworks consist of models, languages, standards, methods and constraints that govern the composition of elements. Implementation infrastructures provide the mechanisms, including programming and run time systems, to enforce the compositions specified using the conceptual framework.

2.1 Conceptual Frameworks

Conceptual frameworks address the following issues: (1) *Element specification:* The conceptual framework should unambiguously identify an element,

and should be sufficiently rich to capture the capabilities of an element, including its functional attributes (e.g., input, output, precondition, effects, etc.) and non-functional attributes (e.g., cost, service quality, security, etc.). Further, the specification should be formally defined and capable of being processed, interpreted and reasoned using agents/machines, e.g., to check if two descriptions are equivalent, partially match, or are inconsistent. (2) *Application process specification:* The conceptual framework should provide information about the elements involved in the application process, their roles, and their interactions. This specification is similar to a workflow, and includes a set of activities and their execution dependencies. (3) *Composition policy specification:* The conceptual framework should define composition methods and constraints and enable users to specify requirements such as cost, performance, QoS, etc.

Related work in conceptual frameworks for autonomic composition includes efforts within the Semantic Web community addressing the description, discovery and composition of services. Projects such as myGRID [36] represent recent efforts aimed at uniting the Semantic Web and Grid computing communities. In particular, the Web Ontology Language (OWL) [19] is emerging as a standard in industry as well as in the scientific and engineering research communities, for Web service discovery, composition, and invocation. The OWL-S *Profile* specifies a service using three information components: service capability specified in terms of its inputs, outputs, preconditions, effects, and component sub-processes; service attributes such as QoS, cost, and classification in the taxonomy; and description of service providers. Note that in addition to describing advertised services, profiles can also be used to describe requested services. The OWL-S *Process Model* allows the requesters to decide whether and how to interact with a service. It defines the basic functions performed by service providers as atomic processes, which can be composed into more complex processes using control structures such as sequence, if-then-else, or split. Finally, OWL-S *Grounding* specifies the implementation details of a service such as messaging protocols and message formats.

In the Grid computing community, workflow models are popular approaches for describing and composing complex scientific applications. A Grid workflow is a set of tasks that are processed on distributed resources in a defined order to accomplish a specific goal. Workflow management techniques can be applied to generate Grid workflow and dynamically assemble applications using services and resources distributed across the Grid. In general, workflow-based systems enact abstract workflow descriptions as composition plans to discover and compose elements. The workflow description can be user-defined or autonomically generated. Workflow descriptions may use markup languages such as XML, WSFL [34], XLANG [35], BPEL4WS [4],and GSFL [12], or use a graphic representation such as Petri Nets [25] and UML (Unified Modeling Language). However, these languages do not provide well-defined semantics, which in turn limits their ability to support seamless service interoperability. On the other hand, while the OWL-S profiles allow descriptions to be more precise, its process model lacks flexibility. For example, OWL-S does not describe the relationships between the elements, their synchronization, or the termination of a process. A

possible solution is to extend workflow descriptions to use the OWL-S profile ontology for element and composition specification.

Finally, composition polices and constraints allow users to express specific requirements and expectations such as performance and availability. Grid environments provide a large number of similar or equivalent services and resources. These services may provide the same functionality but may optimize different non-functional aspects such as performance, cost, reliability, security, etc. Further, different users or applications may have different expectations and requirements. Therefore, only considering functional characteristics during the composition may be insufficient.

2.2 Implementation Infrastructures

Critical components of an implementation infrastructure for autonomic composition include an efficient, scalable and flexible discovery mechanism, and a high-level integration mechanism. The discovery mechanism enables the selection of appropriate elements while the integration mechanism enables selected elements to be composed coherently, without conflicts in element dependencies and interactions. Most of existing approaches compare the syntactic and semantic components [15] of element descriptions during the generation of the composition plan. However, this approach may not ensure runtime compatibility during application execution due to the dynamic availability and state of elements and resources on the Grid. As a result, runtime composability and compatibility checking is important for autonomic composition, specially since interactions can ad hoc, ephemeral and opportunistic.

Realizing an autonomic composition infrastructure for Grid applications presents several challenges. Such an infrastructure has to implement services and protocols to address element representation, discovery, and cooperation, while addressing the scale, heterogeneity and dynamism of Grid environments and applications. Key design issues include the overall system architecture as well as discovery, communication, and coordination subsystems. In a centralized architecture, every element publishes its existence, capabilities and functionalities in a globally known and possibly centralized registry, and every agent queries this central registry to discover elements and compose applications. However, such architecture suffers from performance and scalability bottlenecks, single point failures, and may be more vulnerable to denial of service attacks. On the other hand, decentralized architectures are more scalable, resilient and have higher availability, but require mechanisms for maintaining information consistency and tend to be more complex. High-level communication and coordination subsystems that are based on semantics rather than names/identifiers and addresses and provide abstractions for process cooperations, communication and synchronization, can help reduce this complexity.

The software agent paradigm provides decentralization, dynamic and coordinated decision-making and autonomous behaviors, and supports representation translation, dynamic discovery and negotiated coordination, making it an effective approach for realizing autonomic composition infrastructures.

3 An Agent-based Infrastructure for Autonomic Composition of Grid Applications

The Rudder autonomic composition infrastructure presented in this paper is composed of (1) an agent framework [13] that provides agent abstraction and coordination protocols for supporting dynamic composition, coordination, interactions and application self-managing behaviors, and (2) Comet [14], a decentralized coordination substrate that provides a shared-space abstraction and supports the implementation of the coordination protocols. A conceptual overview of the infrastructure is presented in Figure 1.

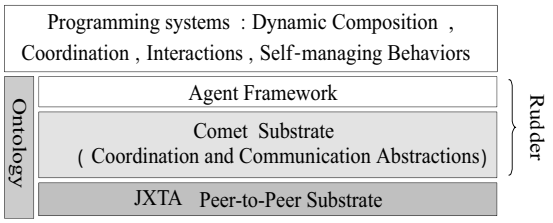


Figure 1: A conceptual overview of the Rudder infrastructure for autonomic composition of Grid applications.

The motivations for employing agents to address autonomic composition for self-managing Grid applications include two aspects. First, agents with knowledge capabilities provide a natural abstraction for bridging external and internal data structures in the system. Typically, discovery systems provide external representations that enhance element accessibility and allow users to relatively easily express what they can offer or what they want, e.g., using the OWL-S profile ontology. Internally, discovery substrates use specific representation, such as keywords, for data indexing and query resolution to achieve efficient and scalable data lookup. Agents provide an effective mechanism for translating between and gluing these representations. Second, the adaptive behaviors of agents enable the composition plans and policies to be enacted through a dynamic negotiation process. Agent negotiation mechanisms can be used to the selection of the most appropriate elements from those that are currently available. This includes evaluating non-functional attributes of the elements, which can be difficult to estimate or predict in dynamic Grid environments and may result in sub-optimal selections.

3.1 Classification of Rudder Agents

The Rudder agent framework [13] defines two types of agents: Component Agent (**CA**) and Composition Agent (**CSA**). CAs represent discrete elements and use OWL-S profile to identify and control the elements. An element may be an application, service or resource unit (e.g., computer, instrument, and data store).

Such an element along with its CA represents a managed element in Rudder. The responsibilities of a CA include advertising the capabilities of the element, providing uniform access to the element, configuring the element based on its execution context, and managing its execution. Transiently generated CSAs dynamically discover and compose managed elements to realize applications. CSAs employ predefined composition plans to discover relevant elements and to negotiate with the CAs to select, configure, and compose the elements.

In Rudder, composition plans are generated from the application process and are available to the CSA [1]. Further, the semantics of terms and concepts used in the composition plans as well as the application specific ontology are common knowledge among agents. A composition plan has three components: (1) a set of atomic tasks, each of which has a semantic description, using the application ontology, that can be used to discover and select elements to fulfill the task; (2) a process description describing the dependencies and interactions between tasks; (3) constraints, which reflect user requirements and may be defined at the task level (e.g., minimize the execution time of a task) as well as the plan level (e.g., minimize total cost). A composition plan is enacted by a CSA by using the task descriptions to semantically discover elements, selecting and configuring appropriate elements, composing these elements using the process description and coordinating with other agents to satisfy constraints and application requirements.

3.2 Coordination Protocols in Rudder

Coordination protocols provided by Rudder include discovery protocols and interaction/negotiation protocols.

Discovery Protocol: enables agents to register, unregister, and discover elements. In Rudder, element profiles are categorized based on an application defined taxonomy, and mapped onto a corresponding semantic space. The discovery process consists of navigating this semantic space to narrow an element query to a small set of potential matching profiles and then performing semantic matching on these profiles.

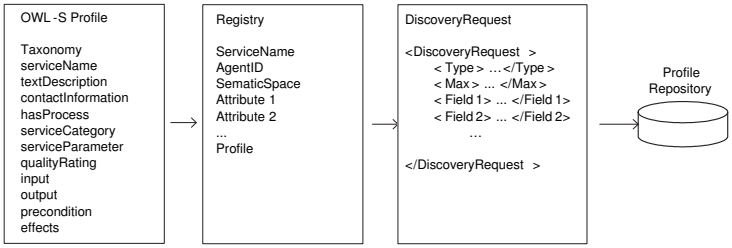


Figure 2: Discovery message for registering/unregistering an element.

[1] Plans may be automatically generated through AI planning and deductive theorem proving. However, this is not currently addressed in Rudder.

When an element is added to the system, its associated CA parses the element's OWL-S profile description, and creates a registry entry that uniquely identifies the element in Rudder. The attributes form the coordinates of a semantic space. For example, a computational storage resource may belong to the 3D storage space with coordinates "space", "band width", and "cost". The process is shown in Figure 2. The registry itself is decentralized and is implemented using the Comet substrate described below. The CA is also responsible for maintaining the consistency of this information in the registry and updates the registry when one or more of the element's attributes change. A periodic heart-beat message is used to ensure the liveliness of elements. When the element permanently leaves the system, the agent unregisters the service and deletes the corresponding registry entry.

The discovery protocols allow agents to search for elements. Searching consists of two steps. First, the agent generates the request description identifying the semantic space and consisting of relevant keywords, partial keywords, and/or wildcards. It then searches for candidate elements within the decentralized repository in a distributed manner. The matching process consists of an initial lexical matching of the keywords in the query followed by a semantic matching [33] to evaluate the similarity between the request and matched element profiles. The matching elements are returned to the requesting agent.

Interaction Protocols: allow distributed agents to interact, coordinate and negotiate during composition in order to reach a mutually acceptable agreement. Implemented protocols are based on existing agent interaction protocols [5], such as those for consensus, mutual exclusion, bargaining, auctions, distributed constraint satisfaction, coalition formation, distributed planning, etc.

The appropriate protocols are selected based on the composition context. During element selection, the CSA can make a decision based on a fixed criteria (e.g., minimum execution time), and consequently the simple and efficient Contract-Net Protocol (CNP) [29] (see Section 4.1) is employed. Similarly, a Marketplace like service-oriented negotiation protocol [28] (see Section 4.2) is employed when the agents need to achieve a mutually acceptable agreement within a dynamic context, for example, the negotiation of non-functional element properties.

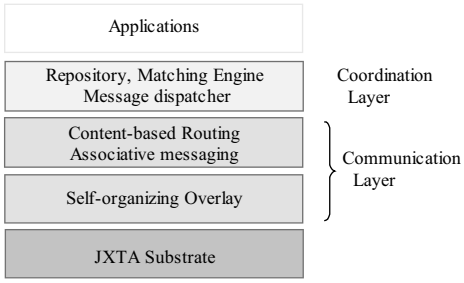


Figure 3: A schematic overview of the Comet system architecture.

3.3 The Comet Decentralized Coordination Substrate

Comet [14] is a scalable peer-to-peer content-based coordination middleware for wide-area distributed environments. It provides an abstraction of a Linda-like [7] shared semantic space that can be associatively accessed by all peer agents without knowledge of the physical location of the tuples in the space or the identifiers of hosts over which the space is distributed.

A schematic overview of the Comet architecture is shown in Figure 3. Comet is composed of layered abstractions prompted by a fundamental separation of communication and coordination concerns. The communication layer provides scalable content-based routing and data delivery operations, including a content-based routing engine and a structured self-organizing overlay. The routing engine provides a decentralized information discovery and associative messaging service, which guarantees queries specified using flexible content descriptors will be routed to all nodes with matching tuples with bounded costs. The current prototype employs Chord [31], which has a ring topology, primarily due to its guaranteed performance, efficient adaptation as nodes join and leave the system, and the simplicity of its implementation. The coordination abstraction supports the shared-space based coordination model and provides Linda-like [7] coordination primitives. The main components of this layer include a data repository for storing tuples and templates, a flexible matching engine, and a message dispatcher that interfaces with the communication layer to convert the coordination primitives to messaging operations and vice versa.

In Comet, tuples are defined using simple XML strings. This lightweight format is flexible enough to represent information required by a range of applications and leads to efficient implementations. A tuple can be retrieved if it exactly or approximately matches a template. Exact matching requires the field names of the template to be specified without any wildcards (i.e., "*"), as in Linda. However, this strict matching must be relaxed in highly dynamic environments, since applications may not know the exact structure of a tuple. Comet supports tuple retrievals with incompletely specified tuples using approximate matching, where only the tag of the template needs to be specified as a keyword or a partial keyword.

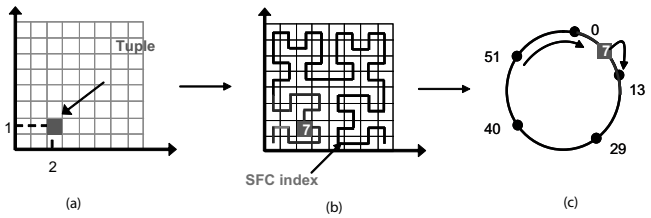


Figure 4: Example of tuple insertion in Comet: (a) a tuple is represented in a 2D keyword space, as the point (2, 1); (b) the point (2, 1) is mapped to index 7 using the Hilbert SFC; (c) the tuple is inserted at node 13 (the successor of SFC index 7).

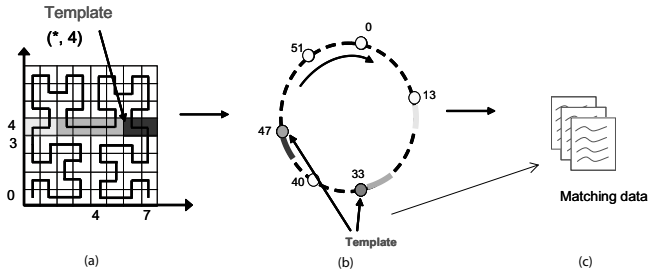


Figure 5: Example of tuple retrieval in Comet: (a) the template defines a rectangular region in the 2-dimensional space consisting of 3 clusters; (b) the nodes that store the clusters are queried; (c) results of the query are sent to the requesting node.

Comet is designed as a distributed hash table (DHT) constructed on a 1-dimensional structured overlay. The recursive and locality-preserving Hilbert Space-Filling Curve (SFC) [16] is used as a hashing function to deterministically map tuples from a semantic information space (defined by the application ontology) to indices in the DHT index space, and to a dynamic set of nodes in the peer overlay. The Comet coordination layer provides primitives to support shared-space based coordination model including *Out*, *In*, *Rd*, *InAll*, and *RdAll*. Each tuple is associated with k keywords selected from its tag and attribute names. This set of k keywords can be viewed as a point in a k-dimensional information space and can be mapped to an index in the DHT index space and a node in the overlay. Similarly, a template can be mapped to a point (if it is fully specified) or a region (if it is partially specified) in the information space and correspondingly segments of the DHT index space and nodes in the overlay. This is illustrated in Figures 4 and 5. Note that Comet additionally supports dynamic transient shared spaces that can be used to exploit context locality during coordination.

3.4 System Implementation

The Comet coordination substrate has been implemented on top of JXTA [24] and has been deployed on distributed clusters and the PlanetLab [23] wide area distributed test bed. JXTA is a platform independent peer-to-peer framework, where peers can self-organize into peergroups, discover peer resources, and communicate with each other. Comet is implemented as a JXTA peergroup service that can be concurrently exploited by multiple applications. The peergroup provides a secure environment where only member peers can access the service instances running on peers of the group. If any one peer fails, the collective peergroup service is not affected and the service is still available from other group members.

The current prototype of Rudder has been implemented using Comet. Each peer node in Comet provides an agent environment responsible for generating,

configuring, and destroying agents. Agents are implemented in JAVA as single thread processing units. The agents communicate with each other by associatively reading, writing, and extracting tuples, and interact using the Rudder interaction protocols. These protocols are implemented using the abstractions and services provided by Comet.

The Rudder discovery protocol is implemented as follows. When an element is added to the system, its CA registers the element by writing its profile into the global tuple space using the Comet *Out (tsname, t)* operation, where *tsname* identifies the semantic space and tuple t encapsulates the registration request. The registration request is routed by Comet to the appropriate peer node in the overlay and the profile is stored in a local repository at that node. Similarly, an element can be unregistered using the *In* operator.

Agents can query a single matching element using the *Rd* operator or all matching elements using the *RdAll* operator. As in the registration case, the query is routed by Comet to the appropriate peer node(s) in the overlay, where semantic matching is used to check the similarity between the request and available profiles. The semantic matching process compares the syntactic and semantic composability of the elements. It is similar to semantic web service matching [33] and ensures that the interacting elements are compatible in aspects of operation modes (request-response), messages, number of parameters, data types, binding protocols, etc. This matching can be implemented using OWL-S matching tools such as OWL-S Matcher [18].

Interaction protocols are implemented using the communication abstractions provided by Comet as follows. For each negotiation, the first step involves session setup where the initiating agent creates a session identifier. This agent then sends the setup message to the selected agents and waits for their acceptance. Once the negotiation has been setup, the initiator informs the participants of the interaction protocol and related information, such as negotiation item, bargaining strategies, roles, etc. After the setup is complete, the agents engaged in the negotiation can directly interact in a peer-to-peer manner.

4 Autonomic Composition of Grid Workflows using Rudder

Autonomic composition supported by Rudder uses the workflow model as its basis. A Grid workflow can be viewed as a set of tasks organized as a well-defined flow of executions, and can be thought of as a composition of Grid services with interaction dependencies. Workflows are effective integration strategies for developing dynamic Grid applications. Further, recent advances in workflow-based techniques allow a workflow to be decomposed as sub-flows and enacted in a decentralized manner, enhancing both performance and scalability [3].

Unlike traditional workflow management system, the infrastructure presented in this paper also addresses dynamic service selection and negotiation. In business workflow management, services are selected during the plan gener-

ation phase based on user defined constraints or parameters. This approach is effective for business workflows as these applications generally consist of shorter transaction processing tasks with small amounts of data. However, Grid applications are generally more dynamic and involve more long running tasks, larger data flows, and utilize heterogeneous and dynamic resources. This requires dynamic workflows involving dynamic selection, configurations and composition of services. Further, negotiations are required to resolve conflicts and competition between candidate services at runtime to meet user objectives.

In autonomic workflow composition, the first step is to generate composition plans. A composition plan includes a predefined workflow process and user specified constraints. Composition plans are generated as follows. First, the application process is represented using a standard workflow language. This representation is then structurally decomposed into a set of component workflows. A composition plan is created by syntactic processing each component workflow. Specifically, the process description is directly extracted from the workflow description. Each task description is defined as an OWL-S profile. Users may additionally specify non-functional properties for tasks, such as QoS, cost, etc.

Once the composition plans have been generated, the system instantiates CSAs to enact the plans in a distributed manner. The CSAs employ discovery protocols to search for candidate elements for each task in the plan, and ensure that the selected elements have compatible syntactic and semantic attributes. As several existing elements may provide "similar" functionalities, the agent may use the non-functional properties of the element (e.g., cost, security, privacy, time, availability, etc.) to select the most appropriate one. Further, dynamic runtime selection between these elements may require negotiation. Key steps in the autonomic composition process are illustrated below using sample negotiation protocols.

4.1 Dynamic Element Selection

Dynamic element selection is based on negotiations among CSAs and CAs and is illustrated using the Contract Net Protocol (CNP) [29], assuming that the CSA has a composition plan to enact and there are several elements and corresponding CSs capable of executing tasks in this plan. The CNP based negotiation process is illustrated in Figure 6. During negotiation, the CSA acts as the manager and the CAs act as contractors. The process consists of the following steps: (1) The CSA searches for candidate CAs using the discovery protocol and advertises the specified task to all candidate CAs. (2) CAs analyze the received task information and respond with a bid; (3) The CSA evaluates received bids, assigns the task to the CA with the best bid, and refuses the other CAs; (4) The CA delegates the task to its associated element and returns the result(s) from task execution to the CSA within a bounded time. If result(s) are not received by the CSA within this time, the CSA explicitly terminates the process. The protocol is implemented using the communication and coordination abstractions provided by Comet. For example, the discovery phase is implemented using the

RdAll operation, where "tsname" is the name of semantic space used in the task description and "template " consists of keywords from this space. Subsequent agent interactions use peer-to-peer communication abstractions provided by Comet .

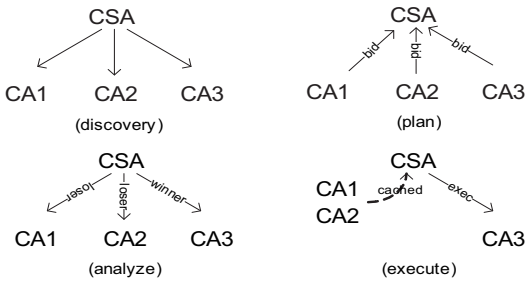


Figure 6: Dynamic element selection using CNP-based negotiation.

The primary reasons for using the CNP-based negotiation protocol are its efficiency and flexibility. The cost of the Contract-Net Protocol is O(N), where N is the number of participating agents. The CNP negotiation process can be customized to specific application requirements. For example, the criteria used by the CSA for choosing CAs can be dynamically specified. Further, the CSA may cache information about discovered elements, and can (re)-negotiate with these cached elements if the selected element can no longer perform the task for some unexpected reason. The overall process can be further optimized by having the CSA evaluate bids after receiving a percentage of the bids instead of waiting for all responses.

4.2 Multi-Stage Property Negotiation

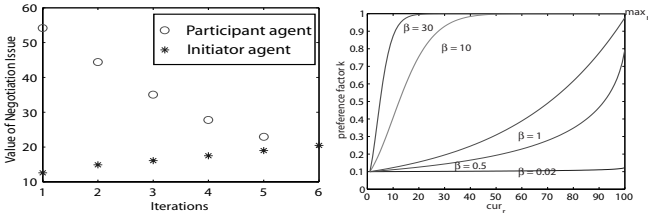
The marketplace model can be used to negotiate non-functional properties of a composition plan. In this negotiation model, instead of a one time determination, values are decided through multiple stage adjustment. This is achieved by iteratively exchanging a finite set of issues between the buyer and seller agents. A buyer agent receives a "plan" from a seller agent, evaluates it and decides either to accept it and stop the negotiation with an agreement, or reject it and propose a counter plan. In case of unsuccessful negotiations, the participating agents can choose to be further coordinated by a mediator, which can be an agent or a system administrator. This mechanism enables agents to resolve locally decided strategies and select a mutually acceptable strategy.

In the implementation of the marketplace model, each negotiation session is setup by an initiator agent. The initiator agent may use the discovery protocol to discovery other participants, which is similar to that used in the CNP protocol described above. Once setup is complete, the agents engaged in the negotiation directly communicate using Comet peer-to-peer communication abstractions. Each negotiating agent uses the remaining amount of a local resource [28] (e.g.,

remaining number of iterations) to determine its plan in successive negotiation iterations as follows:

$$f(cur_r) = k^{(\frac{max_r - cur_r}{max_r - min_r})^{\beta}} \quad (1)$$

where the possible values of the allocated resource is between $[min_r, max_r]$, the current value is cur_r, the preference factor k $(0 < k < 1)$ determines the initial value of the issue under negotiation, and the conceding rate $\beta(\beta > 0)$ determines the agent behaviors. If $\beta > 1$ the function concedes faster and results in a greedy agent. If $\beta <= 1$, the agent is selfless.



(a) (b)

Figure 7: Example of two agent negotiation using the marketplace model.

The mechanism described above enables applications to dynamically tune their configurations to ensure that they continue to meet the composition constraints despite system dynamism and uncertainty. For example, a self-managed distributed Video-On-Demand application must select the appropriate level of network service that can best meet the user requirement while minimizing cost. The desired value of service level must thus be negotiated between the video file server element and the end-user client. For instance, the server has an acceptable range [10, 25] and the client can accept the value in the range of [15, 60]. The appropriate value can decided in Rudder using negotiation between the two component agents. Let the initiator agent have $\beta = 10$ and the other agent have $\beta = 5$. The resource-driven function with $k = 0.1$ used by the both agents are plotted in Figure 7(b). As shown in Figure 7(a), an agreement is reached after 6 iterations and the negotiated value of the issue is 21. The effectiveness and efficiency of this negotiation process can be tailored using different agent configurations.

5 System Operation and Evaluation

This section describes overall system operation and presents an experimental evaluation of Rudder. The experiments were conducted using a deployment of Rudder over a distributed network of computers at Rutgers University. Each computer served as a peer node in the Comet overlay. The overall operation of the system consists of two phases: *bootstrap* and *running*. During the *bootstrap*

phase, peer node join the system and exchange messages with the rest of the group. The *running* phase consists of stabilization and user modes. In the stabilization mode, peer nodes respond to queries issued by other peers in the system to ensure that the routing tables at each peer node are up to date, and to verify that the other peer nodes in the system have not failed or left the system. In the user mode, peer nodes interact as part of the system to provide the coordination services and support autonomic composition.

The experimental evaluation focuses on element discovery and element selection for dynamic composition. The execution time is measured for systems with different number of peer nodes, and for different numbers of elements and agents. In case of two agent negotiation, the overall communication cost is based on the number of negotiation iterations and the latency of peer-to-peer messaging, which is independent of the system size.

Element discovery: This experiment measures the time required to semantically discover registered elements, which is the interval between when a CSA issues a discovery request and when results containing all element profiles matching the query are returned. This time includes the time for routing the templates at the peers, locally matching the template profile with registries in the local repository at the node, and returning matching results. Note that this measurement does not include the cost of semantic matching. The template used in this experiment is specified using element attributes, including service type, location, and performance/QoS guarantees, etc., and has a size of at least 440 bytes. The average execution time shown in Figure 8 illustrates that the discovery time increase (from 0.1s to 65s) is much slower than the increase in the number of elements (from 3 to 2700), and is independent of the system size. This demonstrates the scalability of the system and its suitability to distributed decentralized systems.

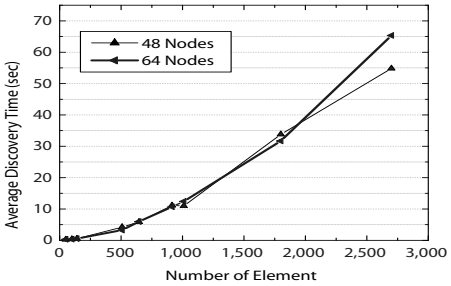
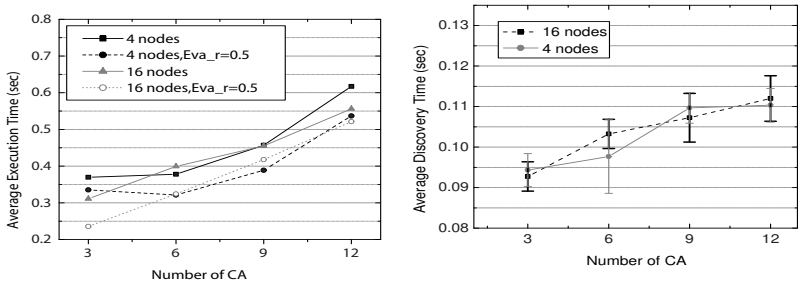


Figure 8: Scalability of element discovery.

Element selection: This experiment evaluates the Contract-Net Protocol based element selection, in which CAs represent elements randomly distributed at the peer nodes and a CSA attempts to find the best CA to execute a task. The task length is fixed and independent of the element selection time. The tasks are generated using a Poisson process with inter-arrival mean time of 1s

and 5s to simulate different application behaviors. The CSA begins the bid evaluation process when (1) it receives all the bids or (2) it receives a certain percentage of bids. The measured execution time is from the time when the CSA announces a task to the time when it gets the results from the selected CA to which the task is assigned, and does not include the task execution time. The time thus includes task announcement, element selection, and result return communication time.



(a) (b)

Figure 9: Average CNP-based element selection and execution time.

Figure 9(a) plots the average execution time for the two cases, i.e., when the CSA begins to evaluate the bids after receiving all the bids and only 50% of the bids (i.e., Eva_r=0.5). In Figure 9(a) , the execution time increases linearly with the number of CAs, and the performance of this process is improved in case (2). Element discovery time is also measured in this experiment, and is separately plotted in Figure 9(b) . Once again, the scalability of discovery is demonstrated, and the discovery time is fairly independent of the system size - the discovery time increases only about 20% when the number of matched profiles increases 400%.

6 Related Work

Reseach efforts related to this paper can be divided into related works in software agent and coordination substrate based system management architectures, and agent-based service discovery and negotiation. These efforts are briefly discussed below.

The TuCSoN [17] and MARS [6] systems adopt programmable tuple spaces to support interaction among co-located mobile agents.These systems have been exploited to support the applications in the areas of Internet information retrieval, workflow management and E-Commence. The tuple space implemented by these systems is centralized has a client-server architecture, making scalability a seriously limitation of these systems. Project MARE [32] exploits mobile

agents and the L^{2}imbo tuple space [8] to address resource discovery and configuration in mobile ad hoc environments.

Software agents have been demonstrated to be an effective mechanism for service discovery and negotiation in various computing environments. An agent marketplace architecture [2] for wireless networks has been proposed and implemented on FIPA-OS [22] for automating service detection, selection, price and service feature negotiation. The negotiating based approach presented in [30] is used in sensor networks to allocate sensor and computational resources so as to optimize the accuracy of multi-sensor target tracking. Adaptive agent negotiations based on multiple models and strategies has also been proposed in [27] to support system adaptations to changing computing needs and resources in Grid environments. However, the proposed approach is not implemented or evaluated.

7 Conclusion

This research addressed the development of autonomic Grid applications, and specifically the dynamic selection and composition of discrete autonomic elements and their negotiation based adaption. The paper presented a decentralized scalable agent-based composition infrastructure for self-managing Grid applications. Agents dynamically discover, select, and compose autonomic elements, and negotiate to execute appropriate self-managing behaviors. A prototype implementation, its operation, and its experimental evaluation were presented.

Acknowledgements

We acknowledge financial support of this research by the National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826.

References

[1] Berners-Lee, T.; Hendler, J.; Lassila, O.: *The Semantic Web*, In: Scientific American, May 2001.

[2] Bircher, E.; Braun, T.: *An Agent-Based Architecture for Service Discovery and Negotiations in Wireless Networks*, In: Proceedings of 2nd International Conference on Wired/Wireless Internet Communications, Germany, February 2004.

[3] Buhler, P.; Vidal, J.: *Towards Adaptive Workflow Enactment using Multiagent Systems*, In: Information Technology and Management Journal: Special Issue on Universal Enterprise Integration, 2004.

[4] Business process execution language for web services (version 1.1),2003, ftp://www6.software.ibm.com/ software/developer/library/ws-bpel.pdf.

[5] Bussmann, S.; Jennings, N. R.; Wooldridge, M.: *Reuse of Interaction Protocols for Decision-oriented Applications*, In: Proceedings of 3rd Int Workshop on Agent-Oriented Software Engineering, 2002.

[6] Cabri, G.; Leonardi, L.; Zambonelli, F.: *Reactive Tuple Spaces for Mobile Agent Coordination*, In: Proceedings of the Second International Workshop on Mobile Agents, 1998.

[7] Carriero, N.; Gelernter, D.: *Linda in Context*, In: Communications of the ACM, Apr. 1989, 32:4, pp. 444-459.

[8] Davies, N.; Friday, A.; Wade, S.; Blair, G.: *L^2imbo: A Distributed Systems Platform for Mobile Computing*, In: MONET, 1998, 3:2, pp.143-156.

[9] Foster, I.; Kesselman, C.; Nick, J.; Tuecke, S.: *Grid Services for Distributed System Integration*, In: Computer, June 2002.

[10] Ganek, A.G.; Corbi, T.A.: *The dawning of the autonomic computing era*, In: IBM Systems Journal, Special Issue on Autonomic Computing,2003,42:1, pp.5-18.

[11] Kephart, J.; Parashar, M.; Sunderam, V.; Das, R. eds: Proceedings of the First International Conference on Autonomic Computing, IEEE Computer Society Press, 2004.

[12] Krishnan, S.; Wagstrom, P.; von Laszewski, G.: *GSFL: A Workflow Framework for Grid Services*,July 2002, http://www.globus.org/cog/papers/gsfl-paper.pdf.

[13] Li, Z.; Parashar, M.: *A Decentralized Agent Framework for Dynamic Com position and Coordination for Autonomic Applications*, In: Proceedings of the 3rd International Workshop on Self-Adaptive and Autonomic Computing Systems, August 2005.

[14] Li, Z.; Parashar, M.: *Comet: A scalable coordination space for decentralized distributed environments*, In: Proceedings of the 2nd International Workshop on Hot Topics in Peer-to-Peer Systems, July 2005.

[15] Medjahed, B.; Bouguettaya, A.; Elmagarmid, A. K.: *Composing Web Services On The Semantic Web*, In: The VLDB Journal, 12:4, 2003, pp.333-351.

[16] Moon,B.; Jagadish, H.v.; Faloutsos, C.; Saltz, J.H.: *Analysis Of The Clustering Properties Of The Hilbert Space Filling Curve*, In: IEEE Transactions on Knowledge and Data Engineering, 2001, 13:1, pp.124-141.

[17] Omicini, A.; Zambonelli F.: *Coordination for Internet Application Development*, In: Autonomous Agents and Multi-Agent Systems, 2:3, 1999, pp.251-269.

[18] OWL Matcher. http://ivs.tu-berlin.de/Projekte/owlsmatcher/.

[19] OWL-S: Semantic markup for web services 1.1. http://www.daml.org/services/owl-s/1.1, 2004.

[20] Parashar, M.; Browne, J.C.: *Conceptual and Implementation Models for the Grid*, In: Proceedings of the IEEE, Special Issue on Grid Computing,2005, 93:3, pp.653-668.

[21] Parashar, M.; Hariri, S.: *Autonomic Computing: An Overview*, In: UPP 2004, Lecture Notes in Computer Science, 2005,3566, pp.247-259.

[22] Poslad, S.; Buckle, P.; Hadingham,R.: *The FIPA-OS agent platform: Open Source for Open Standards*, In: Proceedings of 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, 2000, pp.355-368.

[23] Project PlanetLab. http://www.planet-lab.org.

[24] Project JXTA. http://www.jxta.org.

[25] Reisig, W.: *Petri nets: an Introduction*, Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[26] Roure, D.; Jennings, D.; Shadbolt, N.: *The Semantic Grid: Past, Present, and Future*, In: Proceedings of the IEEE, volume 93, March 2005.

[27] Shen, W.; Li, Y.; Ghenniwa, H.; Wang C.: *Adaptive Negotiation for Agent-Based Grid Computing*, In: Proceedings of WorkShop Challenges02 in 1st International Conference on Autonomous Agents Multiagent Systems, 2002.

[28] Sierra, C.; Faratin, P.; Jennings, N.: *A Service-oriented Negotiation Model Between Autonomous Agents*, In: Proceedings of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, Ronneby, Sweden, 1997.

[29] Smith, R. G.: *The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver*, In: Distributed Artificial Intelligence,1988, pp.357-366.

[30] Soh, L.; Tsatsoulis, C.: *Reflective Negotiating Agents for Real-Time Multisensor Target Tracking*, In: Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01), 2001, pp.1121-1127.

[31] Stoica, I.; Morris, R.; Karger, D.; Kaashoek, F.; Balakrishnan, H.: *Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications*, In: Proceedings of the 2001 ACM SIGCOMM Conference, 2001, pp. 149160.

[32] Storey, M.; Blair, G.; Friday, A.: *Mare: Resource Discovery and Configuration in Ad hoc Networks*, In: Mobible Network Appl.,2002, 7:5, pp.377387.

[33] Tang, S.: *Matching of Web Service Specifications Using daml-s Descriptions*, Thesis, March 2004, Technische Universität Berlin.

[34] Web services flow language (wsfl) 1.0. http://www306.ibm.com/software /solutions/webservices/pdf/WSFL.pdf, 2001.

[35] Web services for business process design. http://www.gotdotnet.com/team/xml wsspecs/xlang-c/default.htm, 2001.

[36] Wroe, C.; Stevens, R.; Goble, C.; Roberts, A.;Greenwood, M.: *A Suite Of daml+oil Ontologies To Describe Bioinformatics Web Services And Data*, In: International Journal of Cooperative Information Systems, 2003.

Authors Bios

Zhen Li is a Ph.D. student in the Department of Electrical and Computer Engineering at Rutgers, The State University of New Jersey. She received a B.S. degree in Computer Science from Zhengzhou University, and a M.E. degree in Computer Engineering from Beijing University of Posts and Telecommunications, China. Zhen is a researcher of The Applied Software Systems Laboratory at the Center for Advanced Information Processing at Rutgers. Her research interests include multi-agent systems, autonomic computing, parallel and distributed computing, and Grid computing.

Manish Parashar is Professor of Electrical and Computer Engineering at Rutgers University, where he also is director of the Applied Software Systems Laboratory. He received a BE degree in Electronics and Telecommunications from Bombay University, India and MS and Ph.D. degrees in Computer Engineering from Syracuse University. He has received the Rutgers Board of Trustees Award for Excellence in Research (2004-2005), NSF CAREER Award (1999) and the Enrico Fermi Scholarship from Argonne National Laboratory (1996). His research interests include autonomic computing, parallel & distributed computing (including peer-to-peer and Grid computing), scientific computing, software engineering. He is a senior member of IEEE, a member of the IEEE Computer Society Distinguished Visitor Program (2004-2007), and a member of ACM.