The Knowledge Engineering Review, Vol. 00:0, 1–15. © 2006, Cambridge University Press
DOI: 10.1017/S000000000000000 Printed in the United Kingdom

Enabling Dynamic Composition and Coordination for Autonomic Grid Applications using the Rudder Agent Framework [1]

Zhen Li and Manish Parashar
The Applied Software Systems Laboratory
Dept. of Electrical and Computer Engineering
Rutgers University

Piscataway NJ, 08854, USA. E-mail: {zhljenny, parashar}@caip.rutgers.edu

Abstract

This paper introduces Rudder, a peer-to-peer agent framework for supporting autonomic applications in decentralized distributed environments. The framework provides agents to discover, select, and compose elements, and defines agent interaction and negotiation protocols to enable appropriate application behaviors to be dynamically negotiated and enacted. The implementations of these protocols as well as agent coordination and negotiation activities are supported by Comet, a scalable decentralized coordination substrate. The operation and experimental evaluation of Rudder is presented.

1 Introduction

Emerging decentralized distributed Grid environments, such as pervasive information systems and global computational infrastructures are aimed at providing seamless access to hardware, software and information resources and services, and are enabling a new generation of applications. However, designers and programmers of these applications have to deal with significant development and management challenges stemming from the highly dynamic computational and interaction behaviors of the applications as well as the complexity and uncertainty inherent in the underlying distributed computing infrastructure. These challenges range from conceptual models to implementation architectures, and span all levels including programming system, runtime, middleware, and operating systems. *Autonomic* applications and systems that are capable of managing (configuring, optimizing, healing, protecting) themselves based on high-level guidance from users, have been proposed to address these challenges. Autonomic systems and applications are typically formulated as dynamic compositions of discrete autonomic elements that interact and coordinate their individual actions to achieve overall self-managing behaviors. Consequently, supporting dynamic compositions of the elements and coordinations of their actions in Grid environments is a critical requirement.

Dynamic composition enables applications or parts of an application to be composed on-the-fly from discrete elements to meet the changing application requirements, system state and/or availability of elements, deal with element failures, optimize performance, etc. In distritbuted environments, enabling dynamic composition is challenging due to the large numbers of available elements, and their heterogeneity and dynamism. Furthermore, maintaining common knowledge

[1]The research presented in this paper is supported in part by the National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826.

about the names/identifiers and addresses of these elements as well as the syntax and semantics of their interfaces in such environment is infeasible. Coordination is the management of runtime dependencies and interactions among the elements in the application. In the case of autonomic applications, these dependencies and interactions can be complex and various, such as producer-consumer, peer-to-peer, collaborative, etc. Further, these relationships and interactions can be ad hoc, ephemeral and opportunistic, may be defined by policies and context, and may be negotiated. Clearly, realizing these behaviors using low-level protocols to cope with the issues of data communication and synchronization as well as process cooperation and competition is non-trivial.

Software agents provide effective high-level mechanisms for information discovery and system/application management and adaptation in distributed environments. This paper presents Rudder, a peer-to-peer agent framework that addresses the issues discussed above. The objective of Rudder is to enable the runtime composition and coordination of autonomic elements in Grid environments. This framework consists of software agents and agent interaction and negotiation protocols. Peer agents in Rudder discover and select elements, and locally control element behaviors and interactions. New elements can be dynamically registered and discovered at runtime by the agents, and appropriate adaptation plans can be negotiated, selected and enacted by cooperating agents. The framework is supported by a fully decentralized shared-space substrate Comet (Li, Z. & Parashar, M. 2005), which provides the core messaging 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 presents the Rudder agent framework and describes the agent classification and coordination protocols. It then introduces the Comet substrate and describes the implementation of Rudder using Comet. Section 3 describes application self-managing behaviors enabled by Rudder. Section 4 describes the operation of Rudder and presents an experimental evaluation. In section 5, related research work and projects are described. Section 6 presents a conclusion.

2 Rudder: An Agent Framework for Autonomic Applications

Rudder addresses element compositions and coordinations using agent cooperations, interactions, and negotiations. It provides protocols to enable peer agents to individually and collectively achieve application adaptiveness. This framework is supported by a shared-space substrate Comet. A conceptual overview of this system is illustrated in Figure 1.

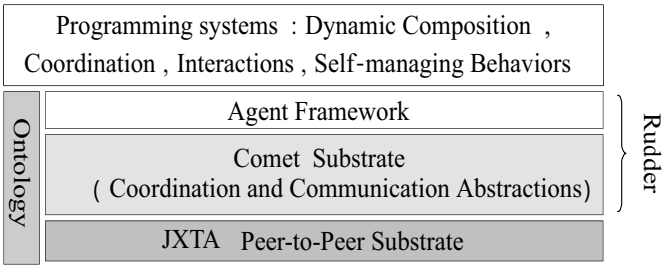


Figure 1 A conceptual overview of Rudder.

2.1 Classification of Rudder Agents

The agents in Rudder are categorized as Component Agent (**CA**) and Composition Agent (**CSA**). CAs represent discrete elements and use OWL-S (OWL-S 2004) profile to identify and control 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.

A CA is responsible for advertising the capabilities of the element, providing uniform access to the element, configuring the element based on its execution context, and managing its execution. CAs manage the computations performed locally within elements and their interactions and operations. The CAs individually controls the element autonomic behaviors by executing predefined rules. Rules incorporate high-level guidance and practical human knowledge in the form of an IF-THEN expression. A rule fires when its condition expression evaluates to be true and the corresponding actions are executed. For example, CAs use behavioral-rules (Liu, H. & Parashar, M. 2004) to control the runtime functional behaviors of an element (e.g., the dynamic selection of algorithms, data representation, input/output format used by the element).

A CSA is transiently generated. It employs predefined composition plans to dynamically discover relevant elements and negotiate with the CAs to select, configure, and compose the elements to realize applications. In Rudder, composition plans are generated from the application process and are available to the CSA [2]. 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.

2.2 Coordination Protocols in Rudder

A set of coordination protocols are defined in Rudder for agents to discover elements and coordinate with each other.

The **Discovery Protocol** allows the agents to register, unregister, and semantically discover elements using domain specific ontologies. When an element is added to the system, its associated CA uses this protocol to register the element's profile. The agent 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. 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. In Rudder, a discovery process consists of two steps: (1) the agent generates the request description identifying the semantic space and consisting of relevant keywords, partial keywords, and/or wildcards. It navigates this semantic space within the decentralized repository and narrows an element query to a small set of potential matching profiles; (2) the semantic matching (Tang, S. 2004) are performed on these profiles by evaluating the similarity between the request and matched element profiles. The matching elements are returned to the requesting agent.

The **Interaction Protocol** allows a group of distributed agents to interact, cooperate, and negotiate during the composition in order to reach a mutually acceptable agreement.

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

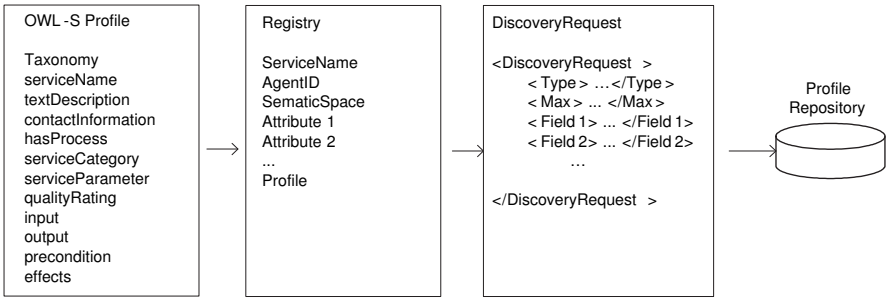


Figure 2 Discovery message for registering/unregistering an element.

Existing agent interaction protocols (Bussmann, *et al.* 2002), such as those for consensus, mutual exclusion, bargaining, auctions, distributed constraint satisfaction, coalition formation, distributed planning, etc., can be supported and customized for specific application or system context. For example, the simple and efficient Contract-Net Protocol(CNP) (Smith, R. G. 1988) is employed for element selection. The CSA can make a decision based on a fixed criteria (e.g., minimum execution time). In the non-functional element property configurations, a Marketplace like service-oriented negotiation protocol (Sierra, C. *et al.* 1997) is used when the agents need to achieve a mutually acceptable agreement within a dynamic context.

2.3 The Comet Coordination Substrate

Comet (Li, Z. & Parashar, M. 2005) is a scalable peer-to-peer content-based coordination space for wide-area distributed environments. This substrate provides an abstraction of a Linda-like (Carriero, N. & Gelernter, D. 1989) shared semantic space that can be associatively accessed by all peer agents without requiring the physical location of the tuples or identifiers of the host.

In Comet, tuples are defined as simple XML strings. This lightweight format is flexible enough to represent information required by all kinds of applications. Comet provides coordination and communication abstractions. The communication abstraction provides scalable content-based messaging and manages system heterogeneity and dynamism. It guarantees that information queries, specified using flexible content descriptors, are served with bounded cost. The coordination abstraction supports the shared-space based coordination model providing Linda-like coordination primitives.

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 (Moon, B. *et al.* 2001) 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.

2.4 Implementation of Rudder

The current prototype of Rudder has been implemented on Project JXTA (Project JXTA), a platform-independent peer-to-peer framework. Each peer node in Rudder provides an agent environment responsible for generating, configuring, and destroying agents. Agents are implemented in JAVA as single thread processing units with predefined rule sets. The agents communicate with each other by associatively reading, writing, and extracting tuples, and interact through

coordination protocols which are implemented using the abstractions and services provided by Comet substrate.

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 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 (OWLSM 2005).

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.

3 Enabling Autonomic Applications

In Rudder, individual agents select local plans based on local information, and then negotiate with peer agents to reach global agreement on the overall plan to enact. Rudder provides mechanisms for supporting existing negotiation models (Shen, W. *et al.* 2002), such as Game Theory Based Model, Contract-Net Protocol(CNP), Auction Model, etc. For example, in the illustrations presented below, the CNP is used to enable dynamic composition and Marketplace based negotiation is adopted for supporting multi-stage self-adaptation.

3.1 Enabling Dynamic Composition

Rudder enables dynamic composition using CNP based negotiations among CSAs and CAs, assuming that the CSA has an application workflow-plan to enact and that there are several CAs capable of executing it. The CSA dynamically discover and select the most suitable CA to execute the workflow-plan through negotiation.

In the negotiation process illustrated in Figure 3 , the CSA acts as the manager and the CAs act as contractors. It consists of the following steps: (1) The CSA searches for potential CAs using the discovery protocol and advertises the specified task information 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 others; (4)The CA delegates the task to its associated element and informs the CSA of the execution result(s) within a bounded time, otherwise the CSA terminates the process explicitly. This protocol is implemented using Comet. The associative communication abstraction and the semantic discovery protocol are used by the CSA to obtain the identifiers of all the potential CAs, and subsequent interactions between the CSA and these CAs use direct peer-to-peer communication.

The reasons of using this protocol are its efficiency and flexibility. The cost of the Contract-Net Protocol is O(N), where N denotes the number of participating agents. Further, the process can be customized to meet specific application requirements. For example, the criteria used by the

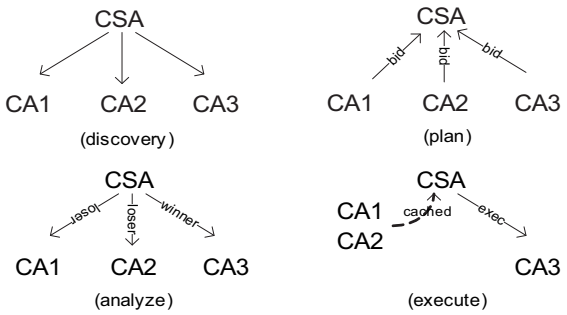


Figure 3 Dynamic composition using CNP-based negotiation.

CSA for choosing CAs can be dynamically specified. Further, the CSA may cache information about discovered elements, and can negotiate with these cached elements if a selected element can not perform the task due to unexpected reasons. The overall process can be further tailored by allowing the CSA to evaluate bids after receiving a percentage of the bids instead of waiting for all responses to reduce the composition overhead.

3.2 Enabling Multi-stage Self-Adaptation

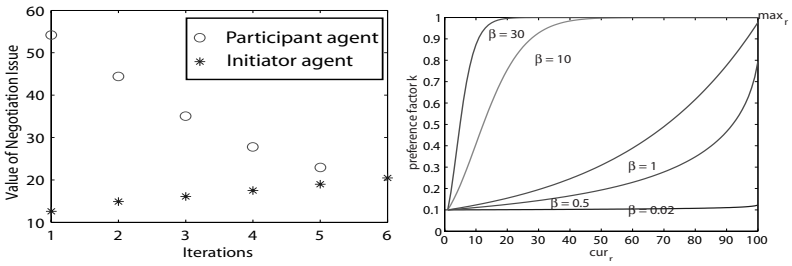
The runtime self-adaptation behaviors of applications must be decided based on current states and context, and may have a global scope. Agents, in this case, will have to reach consensus on the plan to enact and the new system/application configurations to enforce. In Rudder, Marketplace negotiation model is used, where a finite set of issues are exchanged iteratively between buyer and seller agents. When an agent receives a "plan" from another agent, the agent evaluates the received plan and decides either to accept it and stop the negotiation with an agreement, or reject it and propose a counter plan. This mechanism enables the agents to resolve locally decided strategies and achieve a mutually acceptable strategy considering all factors.

In Rudder, each negotiation is allocated a limited resource (e.g., iteration times). The negotiating agents use the remaining amount of this resource to determine its plan in successive negotiation iteration (Sierra, C. *et al.* 1997) based on the following function:

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

in which the value 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. In case of unsuccessful negotiations in which no agreement is reached when the resource is exhausted, the participating agents can choose to be further coordinated by a mediator, which can be an agent or a system administrator.

Using the mechanisms described above, Rudder enables applications to dynamically tune their configurations and operations to ensure that they continue to meet the performance objective despite system dynamism and uncertainty. For example, a desired adaptive behavior of a distributed Video-On-Demand application is to choose an appropriate level of network service that can best meet the user requirement and pricing constrains. In such a scenario, the desired value must be negotiated between the video file server element and the end-user client element. For instance, the server has an acceptable range [10, 25] and the client can accept the value in the range of [15, 60]. In Rudder, the appropriate value is decided by the two component agent negotiation. The initiator agent has $\beta = 10$ and the other agent has $\beta = 5$. The resource-driven function with $k = 0.1$ used by both agents are plotted in Figure 4(b). As shown in Figure 4(a), an agreement is reached after 6 iterations and the negotiated value is 21. The effectiveness and efficiency of the negotiation process can be tailored using different agent configurations.



(a) (b)

Figure 4 Example of two agent negotiation using the marketplace model.

4 System Operation and Evaluation

This section presents the overall system operation and experimental evaluations. The experiments were conducted by deploying Rudder over a distributed network of Linux-based computers in Rutgers University as well as a wide-area environment using PlanetLab (Planetlab) test bed. PlanetLab is a large scale heterogeneous distributed environments composed of interconnected sites with various resources on a global scale.

In the experiments, each machine ran an instance of Rudder, serving 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, a peer node joins the system and exchanges messages with the rest of the group. The *running* phase consists of stabilization and user modes. In the stabilization mode, a peer node responds to queries issued by other peers in the system to ensure that routing tables of peer nodes are up to date, and to verify that other peer nodes in the system have not failed or left the system. In the user mode, each peer node interacts as part of the system to provide the agent generation and coordination services.

The experimental evaluation of the system performance focuses on the element discovery and 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.

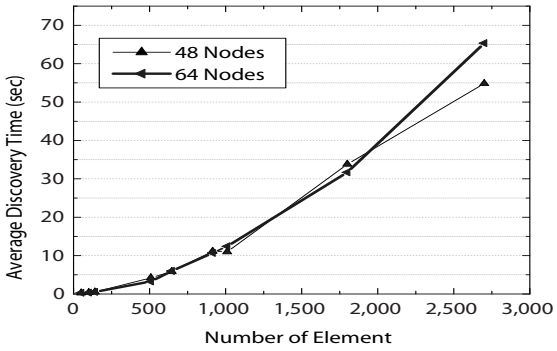


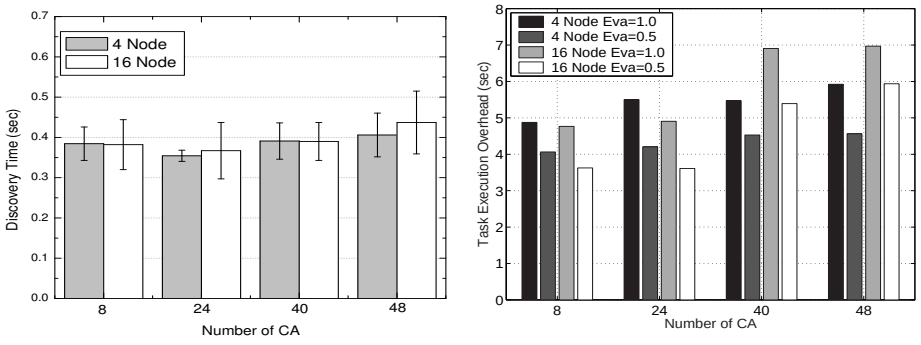
Figure 5 Scalability of element discovery.

Element discovery: This experiment was conducted on the Rutgers campus network. Each machine was a peer node in Rudder. The 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 to peers, matching the profile repository within the node memory, and returning matched 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. with size at least 440 bytes. The average execution time shown in Figure 5 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.

Element selection: These set of experiments were conducted on the PlanetLab test bed. The experiments evaluate element selection using the Contract-Net Protocol. In this experiment, CAs representing elements are randomly distributed at peer nodes, and a CSA attempts to find the best CA to execute arriving tasks. The length of a task is fixed and is independent of the element selection time. Tasks are generated through a Possion process with an inter-arrival mean time of 10 seconds. The CSA begins the bid evaluation process (1) when it receives all the bids or (2) when it receives a certain percentage of bids. The element selection overhead is measured 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, excluding the task execution time. This time includes the time for task announcement, element selection, and returning the results from task execution.

Figure 6 plots the average execution time for the two cases: (1) the CSA begins evaluating bids after it receives all the bids; and (2) the CSA begins evaluating bids after it receives only 50% of the bids (i.e., Eva_r=0.5). Figure 6(a) plots the element discovery time. Once again, the plots show that element discovery scales well and is fairly independent of the system size - the discovery time increases by only about 28% when the number of matched profiles increases by 600%. Further, the plots in Figure 6(b) show that the execution overhead increases linearly with the number of CAs, which is expected. Also, the performance of evaluation process improves in case (2).



(a) (b)

Figure 6 Average CNP-based element selection and execution time.

5 Related Work

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

The TuCSoN (Omicini, A. & Zambonelli, F. 1999) and MARS (Cabri, G. *et al.* 1998) 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 and has a client-server architecture, making scalability a seriously limitation of these systems. Project MARE (Storey, M. *et al.* 2002) exploits mobile agents and the L^{2}imbo tuple space (Davies, N. *et al.* 1998) 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 (Bircher, E. & Braun, T. 2004) for wireless networks has been proposed and implemented on FIPA-OS (Poslad, S. *et al.* 2000) for automating service detection, selection, price and service feature negotiation. The negotiating based approach presented in (Soh, L. & Tsatsoulis, C. 2001) 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 (Shen, W. *et al.* 2002) to support system adaptations to changing computing needs and resources in Grid environments. However, the proposed approach is not implemented or evaluated.

6 Conclusion

This paper presented Rudder, a peer-to-peer agent framework for autonomic grid applications. This research investigated key issues in the developing and executing of autonomic systems and applications, which include dynamic discovery, selection, and composition of discrete elements as well as negotiation based system adaptive behaviors. A prototype implementation, operation, and evaluation were also discussed. Experimental results showed the scalability and flexibility of this framework as well as the feasibility of operating on wide area environment.

References

Bhat, V. et al. 2003 *Autonomic Oil Reservoir Optimization on the Grid*, In: Concurrency and Computation: Practice and Experience.

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

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

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

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

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

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

Liu, H. & Parashar, M. 2004 *A Component-based Programming Framework for Autonomic Applications*, In: Proceedings of 1st IEEE International Conference on Autonomic Computing.

Moon, B. & Jagadish, H. v. & Faloutsos, C. & Saltz, J.H. 2001 *Analysis of the Clustering Properties of the Hilbert Space-Filling Curve*,In: IEEE Transactions on Knowledge and Data Engineering, 13:1, pp. 124-141.

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

OWLSM 2005. http://ivs.tu-berlin.de/Projekte/owlsmatcher/

OWL-S 2004 *Semantic Markup for Web Services 1.1*. http://www.daml.org/services/owl-s/1.1

Poslad, S. & Buckle, P. & Hadingham,R. 2000 *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, pp.355-368.

Project JXTA. http://www.jxta.org.

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

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

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

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

Tang, S. 2004 *Matching Of Web Service Specifications Using DAML-S Descriptions*, Thesis.

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

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