A Decentralized Agent Framework for Dynamic Composition and Coordination for Autonomic 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

Abstract

This paper introduces a peer-to-peer agent framework for supporting autonomic applications in decentralized distributed environments. The framework provides agents to discover, compose, and control elements. It defines 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 a scalable decentralized shared-space based substrate. The implementation and experimental evaluation of the system are also presented.

1 Introduction

Autonomic systems capable of managing themselves are appearing as a new generation of applications in emerging distributed decentralized environments, such as pervasive information systems and the global computational Grid infrastructure. However, these applications are complex with highly dynamic computational and interaction behaviors, and when combined with the uncertainty of the underlying infrastructure, result in significant development and management challenges. Formulations of autonomic applications and systems have viewed them as compositions of discrete composable elements. Enabling these element-based self-managing systems/applications presents many conceptual and implementation challenges that span all levels, including the programming models, runtime, middlewares, and operating systems.

Key challenges include the decentralized and dynamic composition and coordination of distributed elements. Dynamic composition enables applications or parts of an application to be dynamically composed from discrete elements to meet the changing requirements, deal with element failures, optimize performance, etc. Enabling dynamic composition is difficult in distritbuted environments because the available elements are heterogenous, dynamic, and may be numerous. It is infeasible to maintain common knowledge about the names/identifiers and addresses of the entities as well as the syntax and semantics of the interfaces. Coordination is the management of runtime dependencies and interactions among the elements in the system/application. In case of autonomic systems/applications, these dependencies and interactions can be various and complex, such as producer-consumer, peer-to-peer, collaborative, and so on. Further, the 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 extremely difficult.

This paper presents Rudder, a peer-to-peer agent framework for addressing the above challenges. Software agents provide effective high-level mechanisms for dealing with system adaptations as well as information discovery. For example, project MARE [12] exploits mobile agents to address resource discovery and configuration for mobile ad hoc environments. An agent-based marketplace model [2] is proposed to automate the service detection, selection and negotiation in wireless networks. MARS [5] adopts programmable tuple spaces to support coordination of mobile agents and has been exploited to support applications of Internet information retrieval, workflow management and E-Commence. However these systems are not appropriate for the logically decentralized, physically distributed environments, where the discovery and interactions must be addressed in a scalable and flexible manner.

The objective of Rudder is to enable the runtime element composition and coordination in peer-to-peer environments. This framework consists of software agents and agent interaction and negotiation protocols. The peer agents identify elements and locally control element behaviors. The

[*]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.

defined protocols enable the behaviors of individual agents to progress towards a consensus. In Rudder, new elements are dynamically identified and inserted into systems. The most appropriate adaptation plan is negotiated, decided, and enacted by multiple distributed cooperating agents. This framework is implemented and supported by a fully decentralized shared-space substrate COMET [7], 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 agent classification, coordination protocols, and implementation. Section 3 introduces application self-management behaviors enabled by Rudder. Experimental evaluations are demonstrated in section 4. Section 5 presents the conclusion and future work.

2 The Agent Framework

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

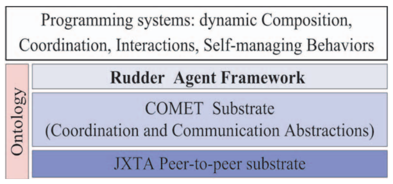

Programming systems: dynamic Composition, Coordination, Interactions, Self-managing Behaviors
Ontology
Rudder Agent Framework
COMET Substrate
(Coordination and Communication Abstractions)
JXTA Peer-to-peer substrate

Figure 1. A conceptual overview of the system.

2.1 Agent Classification

The framework defines two types of peer agents: Component Agent (**CA**) and Composition Agent (**CSA**). CAs represent discrete elements and use profiles to identify and control elements, which can be computational components or resource units (e.g., compute resources, instrument, data store). A profile consists of a set of functional and non-functional attributes (e.g. type, version, operations, etc.), which are semantically defined using an application-specific ontology described using an XML-based language.

CAs manage the computations performed locally within elements and their interactions and operations. Transiently generated CSAs dynamically discover and compose the elements as applications. The agents individually enable the 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 [8] 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). CSAs enable applications to dynamically change flows and elements using workflow-selection and element-selection rules. Workflow-selection rules are used to select appropriate composition plans (e.g., with lowest cost) to enact. Element-selection rules are used to select suitable elements. However, the individual decision may be dependent on others, conflict to others, or not optimal for the global performance. Whenever such situation exists, these agents need to negotiate to determine an alternative satisfying both local and global preferences.

2.2 Coordination Protocols

A set of protocols are defined in Rudder for agents to control 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 agent uses this protocol to register the element's profile. The agent maintains the profile to be consistent with the element and deletes the profile when the element terminates. An element can be discovered by agents semantically using attributes in the element's profile.

The *Control Protocol* allows agents to query the states of elements and control their behaviors. A control message sent to an element includes a list of attributes and/or an operation which are predefined in the element's profile, and then executed by the element. A "heart-beat" message is periodically sent to all the registered elements. The elements must respond to this message to identify its liveness. If it does not respond within a predefined interval, its profile will be removed from the system.

The *Interaction Protocol* allows a group of agents to interact, cooperate and negotiate. Existing agent interaction protocols [3], e.g., the Contract-Net Protocol(CNP) [11], can be supported and customized for specific application or system context.

2.3 The COMET Substrate

COMET is a scalable peer-to-peer content-based coordination space. This space 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 [6] coordination primitives.

2.4 Implementation

The current prototype of Rudder has been implemented on Project JXTA [9], 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 discovery protocols are implemented using COMET Rd and RdAll operations, in which the element profile is encapsulated as a template in the format of XML string and one or all of the matched profiles will be returned. The control protocols are supported by direct peer-to-peer messaging, and the interaction protocols are implemented using appropriate messaging for specified communication patterns.

3 Enabling Self-Management Behaviors

A critical issue in enabling self-management is deciding on the most appropriate behavior to enact. 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 [10], 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 2 , 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.

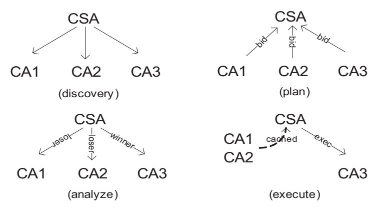


Figure 2. Dynamic composition using CNP-based negotiation.

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 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.

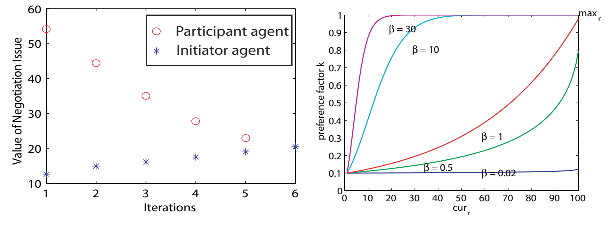
3.2 Enabling Multi-stage Self-Adaptation

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 [4] based on the following function:

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

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.



(a) (b)

Figure 3. Example of two agent negotiation.

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 3(b). As shown in Figure 3(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.

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. Each machine serves 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.

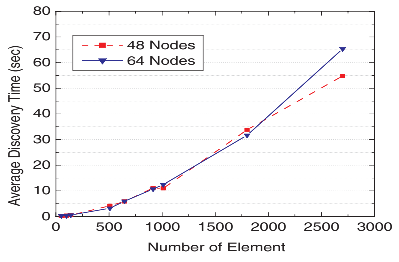
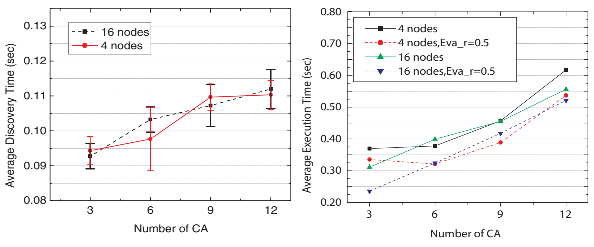


Figure 4. Scalability of element discovery.

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 to peers, matching the profile repository within the node memory, and returning matched results. 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 4 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: This experiment evaluates the Contract-Net Protocol based element selection, in which CAs represent elements randomly distributed on the peer nodes and a CSA attempts to find the best CA to execute a task. The task length is fixed and independent of element selection time. The tasks are generated through a Possion 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 which the task is assigned, excluding task length. This time includes task announcement, element selection, and result returning time. Figure 5(b) 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 5(b) , 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 5(a) . Once again, the discovery scales and is fairly independent of the system size - the discovery time increases only about 20% when the number of matched profiles increases 400%.



(a) (b)

Figure 5. Average CNP-based element selection and execution time.

5 Conclusion and Future Work

In this paper, we presented Rudder, a peer-to-peer agent framework enabling decentralized distributed autonomic applications. Agents dynamically discover, compose, control autonomic elements, and negotiate to execute appropriate self-managing behaviors. This research investigated key issues in the developing and executing of autonomic systems and applications, which include dynamic composition of discrete elements and negotiation based system adaption. The prototype implementation, operation, and evaluation were also discussed. Experimental results showed that this framework is scalable and flexible for supporting autonomic applications.

The future work is to extend this framework to support more autonomic behaviors, and adopt it to support an oil reservoir optimization application [1], which aims to autonomically determine the optimal locations and configurations of oil production and injection wells.

References

[1] V. Bhat, V. Matossian, M. Parashar, M. Peszynska, M. Sen, P.Stoffa, and M. F. Wheeler. Autonomic oil reservoir optimization on the grid. In *Concurrency and Computation: Practice and Experience, 2003*.

[2] E. Bircher and T. Braun. An agent-based architecture for service discovery and negotiations in wireless networks. In *Proceedings of 2nd International Conference on Wired/Wireless Internet Communications (WWIC 2004)*, Frankfurt an der Oder, Germany, February 04 - 06, 2004.

[3] S. Bussmann, N. R. Jennings, and M. Wooldridge. Re-use of interaction protocols for agent-based control applications. In *AOSE*, pages 73–87, 2002.

[4] P. F. C. Sierra and N. Jennings. A service-oriented negotiation model between autonomous agents. In *Proc. 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, pages 17–35, Ronneby, Sweden, 1997.

[5] G. Cabri, L. Leonardi, and F. Zambonelli. Reactive tuple spaces for mobile agent coordination. *Lecture Notes in Computer Science*, 1477, 1998.

[6] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–459, Apr. 1989.

[7] Z. Li and M. Parashar. Comet: A scalable coordination space for decentralized distributed environments. In *Proceedings of Second International Workshop on Hot Topics in Peer-to-Peer Systems*, San Diego, CA, USA, 2005.

[8] H. Liu, M. Parashar, and S. Hariri. A component-based programming framework for autonomic applications. In *Proceedings of 1st IEEE International Conference on Autonomic Computing*. IEEE Computer Society Press, 2004.

[9] Project JXTA. Internet: http://www.jxta.org.

[10] W. Shen, Y. Li, H. Ghenniwa, and C. Wang. Adaptive negotiation for agent-based grid computing. In *Proceedings of WorkShop Challenges02 in 1st International Conference on Autonomous Agents & Multiagent Systems*, Bologna, Italy, July 2002.

[11] R. G. Smith. The contract net protocol: high-level communication and control in a distributed problem solver. In *Distributed Artificial Intelligence*, pages 357–366. Morgan Kaufmann Publishers Inc., 1988.

[12] M. Storey, G. Blair, and A. Friday. Mare: resource discovery and configuration in ad hoc networks. *Mob. Netw. Appl.*, 7(5):377–387, 2002.