Enabling Autonomic Grid Applications: Requirements, Models and Infrastructure*

M. Parashar, Z. Li, H. Liu, V. Matossian, and C. Schmidt

The Applied Software Systems Laboratory,
Rutgers University, Piscataway NJ 08904, USA

Abstract. The increasing complexity, heterogeneity and dynamism of emerging pervasive Grid environments and applications has necessitated the development of autonomic self-managing solutions, which are based on strategies used by biological systems to deal with similar challenges of complexity, heterogeneity, and uncertainty. This paper introduces Project AutoMate and describes its key components. The overall goal of Project Automate is to investigate conceptual models and implementation architectures that can enable the development and execution of such self-managing Grid applications. Two applications enabled by AutoMate are also described.

1 Introduction

The emergence of pervasive wide-area distributed computing, such as pervasive information systems and computational Grid, has enabled a new generation of applications that are based on seamless aggregation and interactions. For example, it is possible to conceive of a new generation of scientific and engineering simulations of complex physical phenomena that symbiotically and opportunistically combine computations, experiments, observations, and real-time data, and can provide important insights into complex systems such as interacting black holes and neutron stars, formations of galaxies, and subsurface flows in oil reservoirs and aquifers, etc. Other examples include pervasive applications that leverage the pervasive information Grid to continuously manage, adapt, and optimize our living context, crisis management applications that use pervasive conventional and unconventional information for crisis prevention and response, medical applications that use in-vivo and in-vitro sensors and actuators for patient management, and business applications that use anytime-anywhere information access to optimize profits.

However, the underlying Grid computing environment is inherently large, complex, heterogeneous and dynamic, globally aggregating large numbers of independent computing and communication resources, data stores and sensor net-

* 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.

O. Babaoglu et al. (Eds.): SELF-STAR 2004, LNCS 3460, pp. 273–290, 2005.
© Springer-Verlag Berlin Heidelberg 2005

works. Furthermore, emerging applications are similarly complex and highly dynamic in their behaviors and interactions. Together, these characteristics result in application development, configuration and management complexities that break current paradigms based on passive components and static compositions. Clearly, there is a need for a fundamental change in how these applications are developed and managed. This has led researchers to consider alternative programming paradigms and management techniques that are based on strategies used by biological systems to deal with complexity, dynamism, heterogeneity and uncertainty. The approach, referred to as autonomic computing, aims at realizing computing systems and applications capable of managing themselves with minimal human intervention.

This paper has two objectives. The first is to investigate the challenges and requirements of programming Grid applications and to present self-managing applications as a means for addressing these requirements. The second is to introduce Project AutoMate, which investigates autonomic solutions to deal with the challenges of complexity, dynamism, heterogeneity and uncertainty in Grid environments. The overall goal of Project AutoMate is to develop conceptual models and implementation architectures that can enable the development and execution of such self-managing Grid applications. Specifically, it investigates programming models, frameworks and middleware services that support definition of autonomic elements, the development of autonomic applications as dynamic and opportunistic compositions of these autonomic elements, and the policy, content and context driven execution and management of these applications.

In this paper we introduce AutoMate and its key components, and describe their underlying conceptual models and implementations. Specifically we describe the Accord programming system, the Rudder decentralized coordination framework, and the Meteor content-based middleware providing support for content-based routing, discovery and associative messaging. We also present two autonomic Grid applications enabled by AutoMate. The first application investigates the autonomic optimization of an oil reservoir by enabling a systematic exploration of a broader set of scenarios, to identify optimal locations based on current operating conditions. The second application investigates the autonomic simulations and management of a forest fire propagation based on static and dynamic environment and vegetation conditions.

The rest of this paper is organized as follows. Section 2 outlines the challenges and requirements of Grid computing. Section 3 introduces Project AutoMate, presents its overall architecture and describes its key components, i.e., the Accord programming framework, the Rudder decentralized coordination framework and the Meteor content-based middleware. Section 4 presents the two illustrative Grid applications enabled by AutoMate. Section 5 presents a conclusion.

2 Grid Computing – Challenges and Requirements

The goal of the Grid concept is to enable a new generation of applications combining intellectual and physical resources that span many disciplines and

organizations, providing vastly more effective solutions to important scientific, engineering, business and government problems. These new applications must be built on seamless and secure discovery, access to, and interactions among resources, services, and applications owned by many different organizations.

Attaining these goals requires implementation and conceptual models [1]. Implementation models address the virtualization of organizations which leads to Grids, the creation and management of virtual organizations as goal-driven compositions of organizations, and the instantiation of virtual machines as the execution environment for an application. Conceptual models define abstract machines that support programming models and systems to enable application development. Grid software systems typically provide capabilities for: (i) creating a transient "virtual organization" or virtual resource configuration, (ii) creating virtual machines composed from the resource configuration of the virtual organization (iii) creating application programs to execute on the virtual machines, and (iv) executing and managing application execution. Most Grid software systems implicitly or explicitly incorporate a programming model, which in turn assumes an underlying abstract machine with specific execution behaviors including assumptions about reliability, failure modes, etc. As a result, failure to realize these assumptions by the implementations models will result in brittle applications. The stronger the assumptions made, the greater the requirements for the Grid infrastructure to realize these assumptions and consequently its resulting complexity. In this section we first highlight the characteristics and challenges of Grid environments, and outline key requirements for programming Grid applications. We then introduce autonomic self-managing Grid applications that can address these challenges and requirements.

2.1 Characteristics of Grid Execution Environments and Applications

The key characteristics of Grid execution environments and applications are:

Heterogeneity: Grid environments aggregate large numbers of independent and geographically distributed computational and information resources, including supercomputers, workstation-clusters, network elements, data-storages, sensors, services, and Internet networks. Similarly, applications typically combine multiple independent and distributed software elements such as components, services, real-time data, experiments and data sources.

Dynamism: The Grid computation, communication and information environment is continuously changing during the lifetime of an application. This includes the availability and state of resources, services and data. Applications similarly have dynamic runtime behaviors in that the organization and interactions of the components/services can change.

Uncertainty: Uncertainty in Grid environment is caused by multiple factors, including (1) dynamism, which introduces unpredictable and changing behaviors that can only be detected and resolved at runtime, (2) failures, which have an increasing probability of occurrence and frequencies as system/application scales

increase; and (3) incomplete knowledge of global system state, which is intrinsic to large decentralized and asynchronous distributed environments.

Security: A key attribute of Grids is flexible and secure hardware/software resource sharing across organization boundaries, which makes security (authentication, authorization and access control) and trust critical challenges in these environments.

2.2 Requirements for Grid Programming Systems

The characteristics listed above impose requirements on the programming systems for Grid applications. Grid programming systems must be able to specify applications which can detect and dynamically respond during execution to changes in both, the state of execution environment and the state and requirements of the application. This requirement suggests that: (1) Grid applications should be composed from discrete, self-managing components which incorporate separate specifications for all of functional, non-functional and interaction-coordination behaviors. (2) The specifications of computational (functional) behaviors, interaction and coordination behaviors and non-functional behaviors (e.g. performance, fault detection and recovery, etc.) should be separated so that their combinations are composable. (3) The interface definitions of these components should be separated from their implementations to enable heterogeneous components to interact and to enable dynamic selection of components.

Given these features of a programming system, a Grid application requiring a given set of computational behaviors may be integrated with different interaction and coordination models or languages (and vice versa) and different specifications for non-functional behaviors such as fault recovery and QoS to address the dynamism and heterogeneity of applications and the environment.

2.3 Grid Computing Research

Grid computing research efforts over the last decade can be broadly divided into efforts addressing the realization of virtual organizations and those addressing the development of Grid applications. The former set of efforts have focused on the definition and implementation of the core services that enable the specification, construction, operation and management of virtual organizations and instantiation of virtual machines that are the execution environments of Grid applications. Services include (1) security services to enable the establishment of secure relationships between a large number of dynamically created subjects and across a range of administrative domains, each with its own local security policy, (2) resource discovery services to enable discovery of hardware, software and information resources across the Grid, (3) resource management services to provide uniform and scalable mechanisms for naming and locating remote resources, support the initial registration/discovery and ongoing monitoring of resources, and incorporate these resources into applications, (4) job management services to enable the creation, scheduling, deletion, suspension, resumption, and synchronization of jobs, (5) data management services to enable accessing, managing, and transferring of data, and providing support for replica management

and data filtering. Efforts in this class include Globus [2], Unicore [3], Condor [4] and Legion [5]. Other efforts in this class include the development of common APIs, toolkits and portals that provide high-level uniform and pervasive access to these services. These efforts include the Grid Application Toolkit (GAT) [6], DVC [7] and the Commodity Grid Kits (CoG Kits) [8]. These systems often incorporate programming models or capabilities for utilizing programs written in some distributed programming model. For example, Legion implements an object-oriented programming model, while Globus provides a capability for executing programs utilizing message passing.

The second class of research efforts, which is also the focus of this paper, deals with the formulation, programming and management of Grid applications. These efforts build on the Grid implementation services and focus on programming models, languages, tools and frameworks, and application runtime environments. Research efforts in this class include GrADS [9], GridRPC [10], GridMPI [11], Harness [12], Satin/IBIS [13] [14], XCAT [15] [16], Alua [17], G2 [18], J-Grid [19], Triana [20], and ICENI [21].

These systems have essentially built on, combined and extended existing models for parallel and distributed computing. For example, GridRPC extends the traditional RPC model to address system dynamism. It builds on Grid system services to combines resource discovery, authentication/authorization, resource allocation and task scheduling to remote invocations. Similarly, Harness and GridMPI build on the message passing parallel computing model, Satin supports divide-and-conquer parallelism on top of the IBIS communication system. GrADS builds on the object model and uses reconfigurable object and performance contracts to address Grid dynamics, XCAT and Alua extend the component based model. G2, J-Grid, Triana and ICENI build on various service based models. G2 builds on .Net [22], J-Grid builds on Jini [23] and current implementations of Tirana and ICENI build on JXTA [24]. While this is natural, it also implies that these systems implicitly inherit the assumptions and abstractions that underlie the programming models of the systems upon which they are based and thus in turn inherit their assumptions, capabilities and limitations.

2.4 Self-managing Applications on the Grid

As outlined above, the inherent scale, complexity, heterogeneity, and dynamism of emerging Grid environments result in application programming and runtime management complexities that break current paradigms. This is primarily because the programming models and the abstract machine underlying these models makes strong assumptions about common knowledge, static behaviors and system guarantees that cannot be realized by Grid virtual machines and which are not true for Grid applications. Addressing these challenges requires redefining the programming framework to address the separations outlined above. Specifically, it requires (1) static (defined at the time of instantiation) application requirements and system and application behaviors to be relaxed, (2) the behaviors of elements and applications to 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, (3) required common knowledge be expressed semantically (ontology and taxonomy) rather than in terms of names, addresses and identifiers, and (4) the core enabling middleware services (e.g., discovery, messaging) be driven by such a semantic knowledge. In the rest of this paper we describe Project AutoMate, which attempts to address these challenges by enabling autonomic self-managing Grid applications.

3 Project AutoMate: Enabling Self-managing Grid Applications

Project AutoMate [25] investigates autonomic solutions that are based on the strategies used by biological systems to deal with similar challenges of complexity, dynamism, heterogeneity and uncertainty. The goal is to realize systems and applications that are capable of managing (i.e., configuring, adapting, optimizing, protecting, healing) themselves. Project AutoMate aims at developing conceptual models and implementation architectures that can enable the development and execution of such self-managing Grid applications. Specifically, it investigates programming models, frameworks and middleware services that support the definition of autonomic elements, the development of autonomic applications as the dynamic and opportunistic composition of these autonomic elements, and the policy, content and context driven definition, execution and management of these applications.

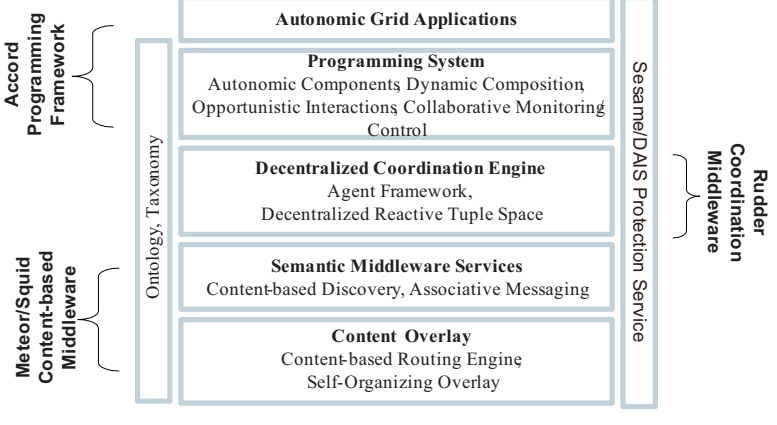


Fig. 1. A schematic overview of AutoMate.

A schematic overview of AutoMate is presented in Figure 1. Components of AutoMate include the Accord [26] programming system, the Rudder [27] decentralized coordination framework, and the Meteor [28] content-based middleware

providing support for content-based routing, discovery and associative messaging. Project AutoMate additionally includes the Sesame [29] context-based access control infrastructure, the DAIS [30] cooperative-protection services and the Discover collaboratory [31, 32] services for collaborative monitoring, interaction and control, which are not described here.

3.1 Accord, a Programming Framework for Autonomic Applications

The Accord programming system [26] addresses Grid programming challenges by extending existing programming systems to enable autonomic Grid applications. Accord realizes three fundamental separations: (1) a separation of computations from coordination and interactions; (2) a separation of non-functional aspects (e.g. resource requirements, performance) from functional behaviors, and (3) a separation of policy and mechanism - policies in the form of rules are used to orchestrate a repertoire of mechanisms to achieve context-aware adaptive runtime computational behaviors and coordination and interaction relationships based on functional, performance, and QoS requirements. The components of Accord are described below.

Accord Programming Model: Accord extends existing distributed programming models, i.e., object, component and service based models, to support autonomic self-management capabilities. Specifically it extends the entities and composition rules defined by the underlying programming model to enable computational and composition/interaction behaviors to be defined at runtime using high-level rules. The resulting *autonomic elements* and their *autonomic composition* are described below. Note that other aspects of the programming model, i.e., operations, model of computation and rules for composition are inherited and maintained by Accord.

Autonomic Elements: An autonomic element extends programming elements (i.e., objects, components, services) to define a self-contained modular software unit with specified interfaces and explicit context dependencies. Additionally, an autonomic element encapsulates rules, constraints and mechanisms for self-management, and can dynamically interact with other elements and the system. An autonomic element is illustrated in Figure 2 and is defined by 3 ports:

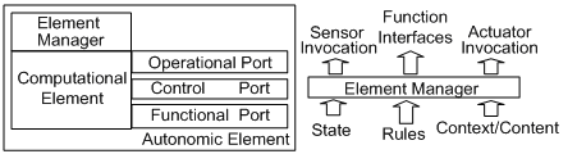


Fig. 2. An autonomic component

The **functional port** (Γ) defines a set of functional behaviors γ provided and used by the element. $\gamma \in \Omega \times \Lambda$, where Ω is the set of inputs and Λ is the set of outputs of the element, and γ defines a valid input-output set.

The **control port** ($\sum$) is the set of tuples (σ, ξ), where σ is a set of sensors and actuators exported by the element, and ξ is the constraint set that controls access to the sensors/actuators. Sensors are interfaces that provide information about the element while actuators are interfaces for modifying the state of the element. Constraints are based on state, context and/or high-level access polices.

The **operational port** (Θ) defines the interfaces to formulate, dynamically inject and manage rules that are used to manage the runtime behavior of the elements and the interactions between elements, between elements and their environments, and the coordination within an application.

Each autonomic element is associated with an element manager (possibly embedded) that is delegated to manage its execution. The element manager monitors the state of the element and its context, and controls the execution of rules. Note that element managers may cooperate with other element managers to fulfill application objectives.

Rules in Accord: Rules incorporate high-level guidance and practical human knowledge in the form of if-then expressions, i.e., IF *condition* THEN *action*, similar to production rule, case-based reasoning and expert systems. *Condition* is a logical combination of element (and environment) sensors, function interfaces and events. *Action* consists of a sequence of invocations of element actuators and/or system actuators, and other interfaces. A rule fires when its condition expression evaluates to be true and causes the corresponding actions to be executed. A priority based mechanism is used to resolve conflicts. Two classes of rules are defined: (1) *behavioral rules* that control the runtime functional behaviors of an autonomic element (e.g., the dynamic selection of algorithms, data representation, input/output format used by the element), and (2)*interaction rules* that control the interactions between elements, between elements and their environment, and the coordination within an autonomic application (e.g., communication mechanism, composition and coordination of the elements). Note that behaviors and interactions expressed by these rules are defined by the model of computation and the rules for composition of the underlying programming model.

Behavioral rules are executed by an element manager embedded within a single element without affecting other elements. Interaction rules define interactions among elements. For each interaction pattern, a set of interaction rules are defined and dynamically injected into the corresponding elements. The coordinated execution of these rules results in the desired interaction and coordination behaviors between the elements.

Autonomic composition in Accord: Dynamic composition enables relationships between elements to be established and modified at runtime. Operationally, dynamic composition consists of a composition plan or workflow generation and execution. Plans may be created at runtime, possibly based on dynamically defined objectives, policies, and the context and content of applications and systems. Plan execution involves discovering elements, configuring them and defining interaction relationships and mechanisms. This may result in elements being

added, replaced or removed or the interaction relationships between elements being changed.

In Accord, composition plans may be generated using the Accord Composition Engine (ACE) [33] or using other approaches, and are expressed in XML. Element discovery uses the Meteor content-based middleware and specifically the Squid discovery service [34]. Plan execution is achieved by a peer-to-peer control network of element managers and agents within Rudder [27]. A composition relationship between two elements is defined by the control structure (e.g., loop, branch) and/or the communication mechanism (e.g., RPC, shared-space) used. A composition agent translates this into a suite of interaction rules, which are then injected into corresponding element managers. Element managers execute the rules to establish control and communication relationships among these elements in a decentralized and parallel manner. Rules can be similarly used for addition or deletion of elements. Note that the interaction rules must be based on the core primitives provided by the system. Accord defines a library of rule sets for common control and communications relationships between elements. The decomposition procedure will guarantee that the local behaviors of individual elements will coordinate to achieve the application's objectives. Runtime negotiation protocols provided by Accord address runtime conflicts and conflicting decisions caused by a dynamic and uncertain environment.

Accord Implementation Issues: The Accord abstract machine assumes the existence of common knowledge in the form of an ontology and taxonomy that defines the semantics for specifying and describing application namespaces, element interfaces, sensors and actuators, and the context and content of systems and applications. This common semantics is used for formulating rules for autonomic management of elements and dynamic composition and interactions between the elements. Further, the abstract machine assumes time-asynchronous system behavior with fail-stop failure modes. Finally, Accord assumes the existence of an execution environment that provides (1) an agent-based control network, (2) support for associative coordination, (3) services for content-based discovery and messaging, (4) support of context-based access control and (4) services for constructing and managing virtual machines for a given virtual organization. These requirements are addressed respectively by Rudder, Meteor, Sesame/DAIS and the underlying Grid middleware on which it builds.

Accord decouples interaction and coordination from computation, and enables both these behaviors to be managed at runtime using rules. This enables autonomic elements to change their behaviors, and to dynamically establish/terminate/change interaction relationships with other elements. Deploying and executing rules does impact performance, however, it increases the robustness of the applications and their ability to manage dynamism. Further, our observations indicate that the runtime changes to interaction relationships are infrequent and their overheads are relatively small. As a result, the time spent to establish and modify interaction relationships is small as compared to typical computation times. A prototype implementation and evaluation of its performance overheads is presented in [35, 36].

3.2 Rudder Coordination Framework

Rudder [27] is a scalable coordination middleware for supporting self-managing applications in decentralized distributed environments. The goal of Rudder is to provide the core capabilities for supporting autonomic compositions, adaptations, and optimizations. Rudder consists of two key components: (1) COMET, a fully decentralized coordination substrate that enables flexible and scalable coordination among agents and autonomic elements, and (2) an agent framework composed of software agents and agent interaction and negotiation protocols. The adaptiveness and sociableness of software agents provides an effective mechanism for managing individual autonomic elements and their relationships in an adaptive manner. This mechanism enables appropriate application behaviors to be dynamically negotiated and enacted by adapting classical machine learning, control, and optimization models and theories. The COMET substrate provides the core messaging and eventing services for connecting agent networks and scalably supporting various agent interactions, such as mutual exclusion, consensus, and negotiation. Rudder effectively supports the Accord programming framework and enables autonomic self-managing applications.

The COMET Substrate: COMET provides a global virtual shared coordination space associatively accessible by all peer agents, and the access is independent of the physical location of the tuples or identifiers of the host. The virtual coordination space builds on an associative messaging substrate and implements a distributed hash table, where the index space is directly generated from the semantic information space (ontology) used by the coordinating entities. COMET also supports dynamically constructed, transient spaces to enable context locality to be explicitly exploited for improved performance.

COMET consists of layered abstractions prompted by a fundamental separation of communication and coordination concerns. It provides an associative communication abstraction and guarantees that content-based query messages, specified using flexible content descriptors, are fully served with bounded costs. This layer essentially maps the virtual information space in a deterministic way to the dynamic set of currently available peer nodes in the system, while maintaining content locality. The COMET coordination abstraction extends the traditional data-driven coordination model with event-based reactivity to changes in system state and data access operations. It defines a reactive tuple abstraction, which consists of additional components: a *condition* that associates *reaction* to events, and a *guard* that specifies how and when the reaction will be executed (e.g., immediately, once). The *condition* is evaluated on an access event. If it evaluates to true, the corresponding reaction is executed. The COMET coordination abstraction provides the basic Linda-like primitives, such as Out, In, and Rd. These basic operations operate on regular as well as reactive tuples and retain their Linda semantics.

The Agent Framework: The Rudder agent framework is composed of a dynamic network of software agents existing at different levels, ranging from individual system/application elements to the overall system/application. Agents

monitor the element states, manage element behaviors and dependencies, coordinate element interactions, and cooperate to manage overall system/application behaviors. An agent is a processing unit that perform actions based on rules, which are dynamically defined to satisfy system/application requirements. Further, agents use profiles which are used to identify and describe elements, interact with them and control them. A profile consists of a set of (functional and non-functional) attributes and operators, which are semantically defined using an application-specific ontology. The framework additionally defines a set of protocols for agent coordination and application/system management. Discovery protocols support the registering, unregistering, and discovery of system/application elements. Control protocols allow the agents to query element states, control their behaviors and orchestrate their interactions. These protocols include negotiation, notification, and mutual exclusion. The agent coordination protocol are scalably and robustly implemented in using the abstractions and service provided by COMET. COMET builds on an associative communication middleware, Meteor, which is described below.

3.3 Meteor: A Content-Based Middleware

Meteor [28] is a scalable content-based middleware infrastructure that provides services for content routing, content discovery and associative interactions. The Meteor stack consists of 3 key components: (1) a self-organizing content overlay, (2) a content-based routing engine and discovery service (Squid), and (3) the Associative Rendezvous Messaging Substrate (ARMS). The Meteor overlay is composed of Rendezvous Peer (RP) nodes, which may be any node on the Grid (e.g., gateways, access points, message relay nodes, servers or end-user computers). RP nodes can join or leave the network at any time. The content overlay provides a single operation, *lookup(identifier)*, which requires an exact content identifier (e.g., name). Given an identifier, this operation locates the peer node where the content should be stored or fetched.

Squid [34] is the Meteor content-based routing engine and decentralized information discovery service. It supports flexible content-based routing and complex queries containing partial keywords, wildcards, and ranges. Squid guarantees that all existing data elements that match a query will be found. The key innovation of Squid is the use of a locality preserving and dimension reducing indexing scheme, based on the Hilbert Space Filling Curve (SFC), which effectively maps the multidimensional information space to the peer identifier space. Keywords can be common words or values of globally defined attributes, depending on the nature of the application that uses Squid, and are based on common ontologies and taxonomies.

The ARMS layer [28] implements the Associative Rendezvous (AR) interaction paradigm. AR is a paradigm for content-based decoupled interactions with programmable reactive behaviors. Rendezvous-based interactions provide a mechanism for decoupling senders and receivers, in both space and time. Such decoupled asynchronous interactions are naturally suited for large, distributed,

and highly dynamic systems such as pervasive Grid environments. AR extends the conventional name/identifier-based rendezvous in two ways. First, it uses flexible combinations of keywords (i.e, keyword, partial keyword, wildcards and ranges) from a semantic information space, instead of opaque identifiers (names, addresses) that have to be globally known. Interactions are based on content described by these keywords. Second, it enables the reactive behaviors at the rendezvous points to be encapsulated within messages, therefore increasing flexibility and enabling multiple interaction semantics (e.g., broadcast multicast, notification, publisher/subscriber, mobility, etc.).

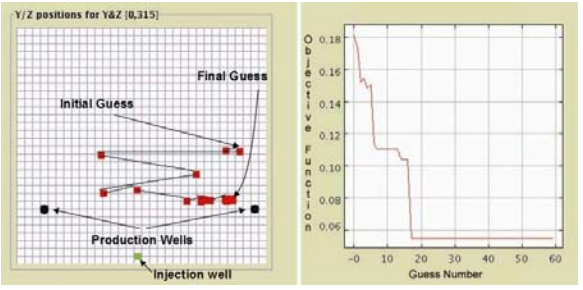
3.4 Current Status

The core components of AutoMate have been prototyped and are currently being used to enable self-managing applications in science and engineering. The initial prototype of Accord extended an object-oriented framework based on C++ and MPI. The current implementation extends the DoE Common Component Architecture (CCA) [37] and we are working on extending an OGSA-based programming system. Current prototypes of Rudder and Meteor build on the JXTA [24] platform and use existing Grid middleware services. Current applications include autonomic oil reservoir optimizations [31, 38], autonomic forest-fire management [39], autonomic runtime management of adaptive simulations [40], and enabling sensor-based pervasive applications [28]. The first two application are briefly described below. Further information about AutoMate and its components and applications can be obtained from http://automate.rutgers.edu/.

4 Autonomic Grid Applications

4.1 Autonomic Oil-Reservoir Optimization

One of the fundamental problems in oil reservoir production is determining the optimal locations of the oil production and injection wells. However, the selection of appropriate optimization algorithms, the runtime configuration and invocation of these algorithms and the dynamic optimization of the reservoir remains a challenging problem. In this research we use AutoMate to support autonomic aggregations, compositions and interactions and enable an autonomic self-optimizing reservoir application. The application consists of: (1) sophisticated reservoir simulation components that encapsulate complex mathematical models of the physical interaction in the subsurface, and execute on distributed computing systems on the Grid; (2) Grid services that provide secure and coordinated access to the resources required by the simulations; (3) distributed data archives that store historical, experimental and observed data; (4) sensors embedded in the instrumented oilfield providing real-time data about the current state of the oil field; (5) external services that provide data relevant to optimization of oil production or of the economic profit such as current weather information or current prices; and (6) the actions of scientists, engineers and other experts, in the field, the laboratory, and in management offices.



(a)

(b)

Fig. 3. Autonomic optimization of the well placement problem using (a) VFSA algorithm (b) SPSA algorithm

The main components of the autonomic reservoir framework [31, 38] are (i) instances of distributed multi-model, multi-block reservoir simulation components, (ii) optimization services based on the Very Fast Simulated Annealing (VFSA) [31] and Simultaneous Perturbation Stochastic Approximation (SPSA) [38], (iii) economic modeling services, (iv) real-time services providing current economic data (e.g. oil prices) and , (v) archives of data that has already been computed, and (vi) experts (scientists, engineers) connected via pervasive collaborative portals.

The overall oil production process is autonomic in that the peers involved automatically detect sub-optimal oil production behaviors at runtime and orchestrate interactions among themselves to correct this behavior. Further, the detection and optimization process is achieved using policies and constraints that minimize human intervention. Policies are used to discover, select, configure, and invoke appropriate optimization services to determine optimal well locations. For example, the choice of optimization service depends on the size and nature of the reservoir. The SPSA algorithm is suited for larger reservoirs with relatively smooth characteristics. In case of reservoirs with many randomly distributed maxima and minima, the VFSA algorithm can be employed during the initial optimization phase. Once convergence slows down, VFSA can be replaced by SPSA. Similarly, policies can also be used to manage the behavior of the reservoir simulator, or may be defined to enable various optimizers to execute concurrently on dynamically acquired Grid resources, and select the best

well location among these based on some metric (e.g., estimated revenue, time or cost of completion).

Figure 3 illustrates the optimization of well locations using the VFSA and SPSA optimization algorithms for two different scenarios. The well positions plots (on the left in 3(a) and (b)) show the oil field and the positions of the wells. Black circles represent fixed injection wells and a gray square at the bottom of the plot is a fixed production well. The plots also show the sequence of guesses for the position of the other production well returned by the optimization service (shown by the lines connecting the light squares), and the corresponding normalized cost value (plots on the right in 3(a) and (b)).

4.2 Autonomic Forest Fire Management Simulation

The autonomic forest fire simulation, composed of *DSM (Data Space Manager), CRM (Computational Resource Manager), Rothermel, WindModel,* and *GUI elements*, predicts the speed, direction and intensity of the fire front as the fire propagates using static and dynamic environment and vegetation conditions. *DSM* partitions the forest represented by a 2D data space into sub spaces based on current system resources information provided by *CRM*. Under the circumstance of load imbalance, *DSM* re-partitions the data space. *Rothermel* generates processes to simulate the fire spread on each subspace in parallel based on current wind direction and intensity simulated by the *WindModel*, until no *burning* cells remain. Experts interact with the above elements using the *GUI* element.

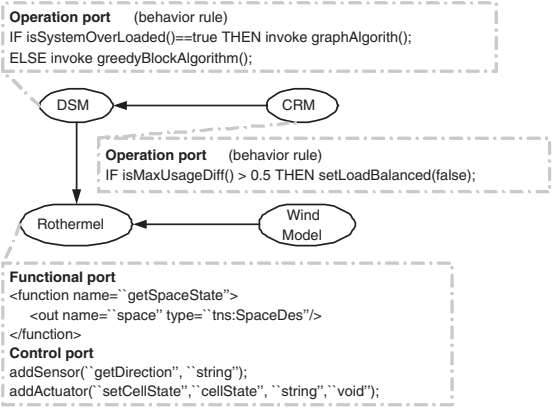


Fig. 4. Examples of the port definition and rules

We use the *Rothermel*, *DSM*, and *CRM* as examples to illustrate the definition of the Accord functional, control and operational ports, as shown in Figure 4. *Rothermel*, for example, provides *getSpaceState* to expose space information as part of its **Functional Port**, and provides the sensor *getDirection* to get the fire spread direction and the actuator *setCellState* to modify the state of a

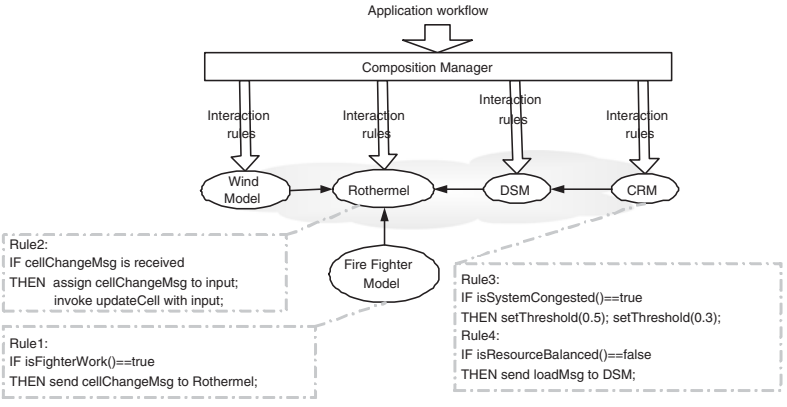


Fig. 5. Add a new component *Fire Fighter Model* and change the interaction relationship between *CRM* and *DSM*

specified cell as part of its **Control Port**. The *DSM* and *CRM* receive rules to manage their runtime behaviors through the **Operation Port**.

Behavior rules can be defined at compile time or at runtime and injected into corresponding element managers to dynamically manage the computational behaviors of elements. As illustrated in Figure 4, *DSM* dynamically selects an appropriate algorithm based on the current system load and *CRM* will detect load imbalance when the maximal difference among resource usage exceeds the threshold according to the behavior rules shown.

The application workflow is decomposed by the *Composition Manager* into interaction rules, which are injected into individual elements. Therefore, addition, deletion and replacement of elements can be achieved using corresponding interaction rules. For example, a new element, *Fire Fighter Model*, modelling the behaviors of the fire fighters, is added to the application as shown in Figure 5, by inserting Rule1 into *Fire Fighter Model* and Rule2 into *Rothermel*. Similarly, changing an interaction relationship can be achieved by replacing the existing interaction rules with new rules. As shown in Figure 5, *CRM* dynamically decreases the frequency of notifications to *DSM* when the communication network is congested based on Rule3 and Rule4.

5 Conclusion

In this paper, we presented Project AutoMate and described its key components. Project AutoMate investigates solutions that are based on the strategies used by biological systems to deal with similar challenges of complexity, dynamism, heterogeneity and uncertainty. This approach, referred to as Autonomic Computing, aims at realizing systems and applications that are capable of managing (i.e., configuring, adapting, optimizing, protecting, healing) themselves. The overall goal of Project AutoMate is to investigate conceptual models and implementation architectures that can enable the development and execution of such self-

managing Grid applications. Specifically, it investigates programming models, frameworks and middleware services that support the definition of autonomic elements, the development of autonomic applications as the dynamic and opportunistic composition of these autonomic elements, and the policy, content and context driven definition, execution and management of these applications. Two case-study applications, autonomic oil reservoir optimization and autonomic forest fire management, enabled by AutoMate were also presented.

References

1. Parashar, M., Browne, J.C.: Conceptual and implementation models for the grid. Proceedings of the IEEE, Special Issue on Grid Computing (2005)
2. The globus alliance, http://www.globus.org.
3. Unicore forum, http://www.unicore.org.
4. Thain, D., Tannenbaum, T., Livny, M.: Condor and the Grid. John Wiley & Sons Inc. (2002)
5. Grimshaw, A.S., Wulf, W.A.: The legion vision of a worldwide virtual computer. Communications of the ACM **40** (1997) 39 – 45
6. Allen, G., Davis, K., Dolkas, K.N., Doulamis, N.D., Goodale, T., Kielmann, T., Merzky, A., Nabrzyski, J., Pukacki, J., Radke, T., Russell, M., Seidel, E., Shalf, J., Taylor, I.: Enabling applications on the grid: A Gridlab overview. International Journal of High Performance Computing Applications: Special issue on Grid Computing: Infrastructure and Applications (2003) to appear
7. Taesombut, N., Chien, A.: Distributed virtual computer (dvc): Simplifying the development of high performance grid applications. In: Workshop on Grids and Advanced Networks (GAN '04), IEEE Cluster Computing and the Grid (CCGrid2004) Conference, Chicago, IL USA (2004)
8. Laszewski, G.v., Foster, I., Gawor, J.: Cog kits: A bridge between commodity distributed computing and high-performance grids. In: ACM 2000 Conference on Java Grande, San Francisco, CA USA, ACM Press (2000) 97 – 106
9. Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., Johnsson, L., Kennedy, K., Kesselman, C., Mellor-Crummey, J., Reed, D., Torczon, L., Wolski, R.: The grads project: Software support for high-level grid application development. International Journal of High Performance Computing Applications **15** (2001) 327–344
10. Nakada, H., Matsuoka, S., Seymour, K., Dongarra, J., Lee, C., Casanova, H.: Gridrpc: A remote procedure call api for grid computing (2003)
11. Ishikawa, Y., Matsuda, M., Kudoh, T., Tezuka, H., Sekiguchi, S.: The design of a latency-aware mpi communication library. In: Proceedings of SWOPP03. (2003)
12. Migliardi, M., Sunderam, V.: The harness metacomputing framework. In: Proceedings of ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, TX, SIAM (1999)
13. Nieuwpoort, R.V.v., Maassen, J., Wrzesinska, G., Kielmann, T., Bal, H.E.: Satin: Simple and efficient java-based grid programming. Journal of Parallel and Distributed Computing Practices (2004)
14. Nieuwpoort, R.V.v., Maassen, J., Wrzesinska, G., Hofman, R., Jacobs, C., Kielmann, T., Bal, H.E.: Ibis: A flexible and efficient java-based grid programming environment. (Concurrency & Computation: Practice & Experience)

15. Govindaraju, M., Krishnan, S., Chiu, K., Slominski, A., Gannon, D., Bramley, R.: Xcat 2.0: A component-based programming model for grid web services. Technical Report Technical Report-TR562, Dept. of C.S., Indiana Univ (2002)
16. Krishnan, S., Gannon, D.: Xcat3: A framework for cca components as ogsa services. In: Proceedings of HIPS 2004, 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments. (2004)
17. Ururahy, C., Rodriguez, N.: Programming and coordinating grid environments and applications. In: Concurrency and Computation: Practice and Experience. Number 5 (2004)
18. Kelly, W., Roe, P., Sumitomo, J.: G2: A grid middleware for cycle donation using .net. In: Proceedings of The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications. (2002)
19. Mathe, J., Kuntner, K., Pota, S., Juhasz, Z.: The use of jini technology in distributed and grid multimedia systems. In: MIPRO 2003, Hypermedia and Grid Systems, Opatija, Croatia (2003) 148–151
20. Taylor, I., Shields, M., Wang, I., Philp, R.: Distributed p2p computing within triana: A galaxy visualization test case. In: International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France, IEEE Computer Society Press (2003)
21. Furmento, N., Hau, J., Lee, W., Newhouse, S., Darlington, J.: Implementations of a service-oriented architecture on top of jini, jxta and ogsa. In: Proceedings of UK e-Science All Hands Meeting. (2003)
22. Microsoft .net. (http://www.microsoft.com/net/)
23. Jini network technology. (http://wwws.sun.com/software/jini/)
24. Project jxta. http://www.jxta.org (2001)
25. Agarwal, M., Bhat, V., Li, Z., Liu, H., Matossian, V., Putty, V., Schmidt, C., Zhang, G., Parashar, M., Khargharia, B., Hariri, S.: Automate: Enabling autonomic applications on the grid. In: Autonomic Computing Workshop The Fifth Annual International Workshop on Active Middleware Services (AMS 2003), Seattle, WA USA (2003) 365–375
26. Liu, H., Parashar, M., Hariri, S.: A component-based programming framework for autonomic applications. In: 1st IEEE International Conference on Autonomic Computing (ICAC-04), New York, NY, USA, IEEE Computer Society Press (2004) 278 – 279
27. Li, Z., Parashar, M.: Rudder: A rule-based multi-agent infrastructure for supporting autonomic grid applications. In: 1st IEEE International Conference on Autonomic Computing (ICAC-04), New York, NY, USA, IEEE Computer Society Press (2004) 278 – 279
28. Jiang, N., Schmidt, C., Matossian, V., Parashar, M.: Content-based decoupled interactions in pervasive grid environments. In: First Workshop on Broadband Advanced Sensor Networks, BaseNets'04, San Jose, California (2004)
29. Zhang, G., Parashar, M.: Dynamic context-aware access control for grid applications. In: 4th International Workshop on Grid Computing (Grid 2003), Phoenix, AZ, USA, IEEE Computer Society Press (2003) 101 – 108
30. Zhang, G., Parashar, M.: Cooperative mechanism against ddos attacks. In: IEEE International Conference on Information and Computer Science (ICICS 2004), Dhahran, Saudi Arabia (2004)
31. Bhat, V., Matossian, V., Parashar, M., Peszynska, M., Sen, M., Stoffa, P., Wheeler, M.F.: Autonomic oil reservoir optimization on the grid. Concurrency and Computation: Practice and Experience, John Wiley and Sons (2003)

32. Mann, V., Matossian, V., Muralidhar, R., Parashar, M.: DISCOVER: An environment for Web-based interaction and steering of high-performance scientific applications. Concurrency and Computation: Practice and Experience **13** (2001) 737–754
33. Agarwal, M., Parashar, M.: Enabling autonomic compositions in grid environments. In: 4th International Workshop on Grid Computing (Grid 2003), Phoenix, AZ, USA, IEEE Computer Society Press (2003) 34 – 41
34. Schmidt, C., Parashar, M.: Enabling flexible queries with guarantees in p2p systems. IEEE Internet Computing **8** (2004) 19 – 26
35. Liu, H.: A component-based programming framework for autonomic grid applications. Ph.D. Proposal (2004)
36. Liu, H., Parashar, M.: Rule-based monitoring and steering of distributed scientific application. International Journal of High Performance Computing and Networking (IJHPCN) (2005)
37. Allan, B.A., Armstrong, R.C., Wolfe, A.P., Ray, J., Bernholdt, D.E., Kohl, J.A.: The cca core specifications in a distributed memory spmd framework. Concurrency and Computing:Practice and Experience, John Wiley and Sons **14** (2002) 323 – 345
38. Matossian, V., Parashar, M., Bangerth, W., Klie, H., Wheeler, M.: An autonomic reservoir framework for the stochastic optimization of well placement. Cluster Computing: The Journal of Networks, Software Tools, and Applications, Special Issue on Autonomic Computing, Kluwer Academic Press (to appear)
39. Khargharia, B., Hariri, S., Parashar, M.: vgrid: A framework for building autonomic applications. In: 1st International Workshop on Heterogeneous and Adaptive Computing– Challenges of Large Applications in Distributed Environments (CLADE 2003), Seattle, WA, USA, Computer Society Press (2003) 19–26
40. Chandra, S., Parashar, M., Hariri, S.: Gridarm: An autonomic runtime management framework for samr applications in grid environments. In: New Frontiers in High-Performance Computing, Proceedings of the Autonomic Applications Workshop, 10th International Conference on High Performance Computing (HiPC 2003). Elite Publishing, Hyderabad, India (2003) 286 – 295