# AutoMate: Enabling Autonomic Applications on the Grid*

M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen and M. Parashar
The Applied Software Systems Laboratory
Department of Electrical and Computer Engineering,
Rutgers, The State University of New Jersey,
automate@caip.rutgers.edu

B. Khargharia and S. Hariri
High Performance Distributed Computing Laboratory
Department of Electrical and Computer Engineering,
University of Arizona,
{bithika_k, hariri}@ece.arizona.edu

## Abstract

*The increasing complexity, heterogeneity and dynamism of networks, systems, services applications have made our computational/information infrastructure brittle, unmanageable and insecure. This has necessitated the investigation of a new paradigm for design, development and deployment based on strategies used by biological systems to deal with complexity, heterogeneity, and uncertainty, i.e. autonomic computing. This paper introduces the AutoMate project and describes its key components. The overall objective of AutoMate is to investigate key technologies to enable the development of autonomic Grid applications that are context aware and are capable of self-configuring, self-composing, self-optimizing and self-adapting. Specifically, it will investigate the definition of autonomic components, the development of autonomic applications as dynamic composition of autonomic components, and the design of key enhancements to existing Grid middleware and runtime services to support these applications.*

## 1 Introduction

The emergence of computational Grids [4] and the potential for seamless aggregation, integration and interactions has made it possible to conceive a new generation of realistic, scientific and engineering simulations of complex physical phenomena. These applications will symbiotically and opportunistically combine computations, experiments, observations, and real-time data, and will provide important insights into complex systems such as interacting black holes and neutron stars, formations of galaxies, subsurface flows in oil reservoirs and aquifers, and dynamic response of materials to detonations. However, the phenomenon being modeled by these applications is inherently multi-phased, dynamic and heterogeneous (in time, space, and state) requiring very large numbers of software components and very dynamic compositions and interactions between these components. Furthermore, the underlying Grid infrastructure is similarly heterogeneous and dynamic, globally aggregating large numbers of independent computing and communication resources, data stores and sensor networks. The combination of the two results 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 formulated, composed and managed so that their heterogeneity and dynamics can match and exploit the heterogeneous and dynamic nature of the Grid. In fact, we have reached a level of complexity, heterogeneity, and dynamism for which our programming environments and infrastructure are becoming unmanageable and insecure. 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, heterogeneity and uncertainty. The approach is referred to as autonomic computing [1]. An autonomic computing system is one that has the capabilities of being self-defining, self-healing,

*The work presented in this paper was supported in part by the National Science Foundation via grant numbers ACI 9984357 (CAREERS), EIA 0103674 (NGS) and EIA-0120934 (ITR), by DOE ASCI/ASAP (Caltech) via grant numbers PC295251 and 1052856, and the DOE Scientific Discovery through Advanced Computing (SciDAC) program via grant number DE-FC02-01ER41184.

PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP
ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE

self-configuring, self-optimizing, self-protecting, contextually aware, and open. The overall objective of the AutoMate project is to investigate key technologies to enable the development of autonomic Grid applications that are context aware and are capable of self-configuring, self-composing, self-optimizing and self-adapting. Specifically, it will investigate the definition of autonomic components, the development of autonomic applications as dynamic composition of autonomic components, and the design of key enhancements to existing Grid middleware and runtime services to support these applications. Specific issues addressed include:

*Definition of Autonomic Components:* The definition of programming abstractions and supporting infrastructure that will enable the definition of autonomic components. In addition to the interfaces exported by traditional components, autonomic components provide enhanced profiles or contracts that encapsulate their functional, operational, and control aspects. These aspects enhance the interfaces to export information and policies about their behavior, resource requirements, performance, interactivity and adaptability to system and application dynamics. Furthermore, they encapsulate sensors, actuators, access policies and a policy-engine. Together, aspects, policies, and policy engine allow autonomic components to consistently configure, manage, adapt and optimize their execution.

*Dynamic Composition of Autonomic Applications:* The development of mechanisms and supporting infrastructure to enable autonomic applications to be dynamically and opportunistically composed from autonomic components. The composition will be based on policies and constraints that are defined, deployed and executed at run time, and will be aware of available Grid resources (systems, services, storage, data) and components, and their current states, requirements, and capabilities.

*Autonomic Middleware Services:* The design, development, and deployment of key services on top of the Grid middleware infrastructure to support autonomic applications. One of the key requirements for autonomic behavior and dynamic compositions is the ability of the components, applications and resources (systems, services, storage, data) to interact as peers. Furthermore the components should be able to sense their environment. In this project, we extend the Grid middleware with (1) a peer-to-peer substrate, (2) context aware services, and (3) peer-to-peer deductive engines for composition, configuration and management of autonomic applications. An active peer-to-peer control network will combine sensors, actuators and rules to configure and tune components and their execution environment at runtime and to satisfy requirements and performance and quality of service constraints.

In this paper we introduce the AutoMate architecture and describe its key components. The rest of this paper is organized as follows. Section 2 gives an overview of the design and architecture of AutoMate. Section 3 describes autonomic components in AutoMate. Section 3.2 describes autonomic compositions and interactions in AutoMate. Section 4 presents an overview of the structure and operation of the RUDDER deductive engine. Section 5 presents the design and operation of the dynamic context aware access control engine. Section 6 describes the Pawn P2P messaging substrate. Section 7 presents SQUID, a P2P system of flexible information discover. Section 8 briefly discusses the use of AutoMate in enabling autonomic science and engineering. Finally Section 9 presents some concluding remarks.

## 2 AutoMate: An Autonomic Component Framework for Grid Applications

The overall research objective of this project is to develop and deploy AutoMate, a framework for enabling autonomic Grid applications. Our technical approach builds on three fundamental concepts:

- Separation of policy from mechanism distilling out the aspects [5, 13] of components and enabling them to orchestrate a repertoire of mechanisms for responding to the heterogeneity and dynamics, both of the applications and the Grid infrastructure. The policies that drive these mechanisms are specified separately. Examples of mechanisms are alternative numerical algorithms, domain decompositions, and communication protocols; an example of a policy is to select a latency-tolerant algorithm when network load is above certain thresholds.

- Context, constraint and aspect based composition techniques applied to applications and middleware as an alternative to the current processes for translating the application's dynamic requirements for functionality, performance, quality of service, into sets of components and Grid resource requirements.

- Dynamic, proactive, and reactive component management to optimize resource utilization and application performance in situations where computational characteristics and/or resource characteristics may change. For example, if adaptive mesh refinement increases computational costs, we may negotiate to obtain additional resources or to reduce resolution, depending on resource availability and user preferences.

Building on these fundamental concepts, AutoMate addresses fundamental issues and provides key solutions



PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE

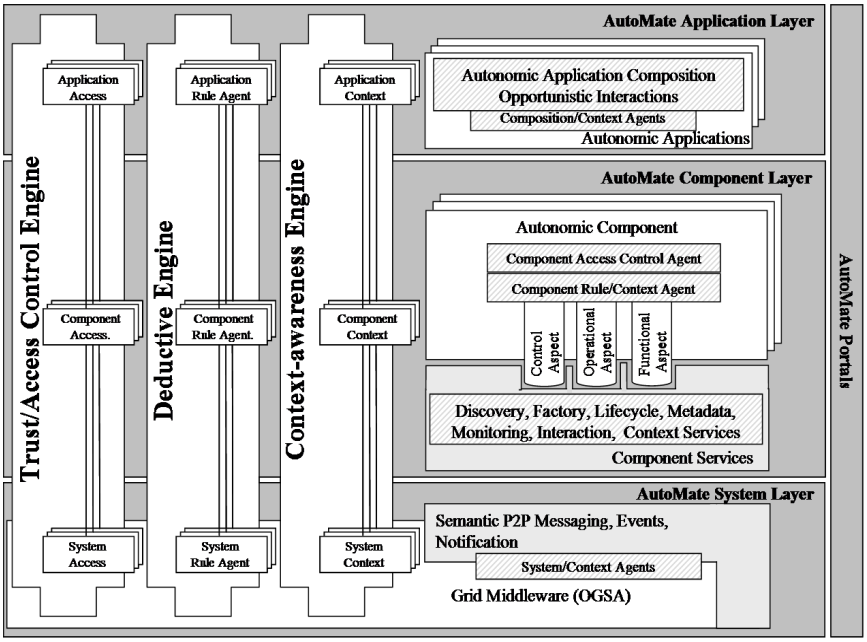


**Figure 1. AutoMate Architecture Diagram**

in the autonomic formulation, composition, and runtime management of applications on the Grid.

A schematic of the overall architecture is presented in Figure 1. AutoMate builds on the emerging Grid infrastructure and extends the Open Grid Service Architecture (OGSA)[21]. AutoMate is composed of the following components:

**AutoMate System Layer:** The AutoMate system layer builds on the Grid middleware and OGSA and extends core Grid services (security, information and resource management, data management) to support autonomic behavior. Furthermore, this layer provides specialized services such as peer-to-peer semantic messaging, events and notification.

**AutoMate Component Layer:** The AutoMate component layer addresses the definition, execution and runtime management of autonomic components. It consists of AutoMate components that are capable of self configuration, adaptation and optimization, and supporting services such as discovery, factory, lifecycle, context, etc. (which builds on core OGSA services).

**AutoMate Application Layer:** The AutoMate application layer builds on the component and system layers to support the autonomic composition and dynamic (opportunistic) interactions between components.

**AutoMate Engines:** The AutoMate engines are decentralized (peer-to-peer) networks of agents in the system. The context-awareness engine is composed of context agents and services and provides context information at different levels to trigger autonomic behaviors. The deductive engine is composed of rule agents which are part of the applications, components, services and resources, and provides the collective decision making capability to enable autonomic behavior. Finally, the trust and access control engine is composed of access control agents and provides dynamic context-aware control to all interactions in the system.

In addition to these layers, AutoMate portals provide users with secure, pervasive (and collaborative) access to the different entities. Using these portals users can access resource, monitor, interact with, and steer components, compose and deploy applications, configure and deploy rules, etc. AutoMate leverages the experiences and technologies developed as part of the DISCOVER/DIOS [7, 8] computational collaboratory project (http://www.discoverportal.org). The different components are described in the following sections.

# 3 ACCORD: An Autonomic Component Framwork

## 3.1 Autonomic Components in AutoMate

Autonomic components in AutoMate export information and policies about their behavior, resource requirements, performance, interactivity and adaptability to system and application dynamics. In addition to the functional interfaces exported by traditional components, AutoMate components provide semantically enhanced profiles or contracts that encapsulate their functional, operational, and control aspects. A conceptual



PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP
ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE

overview of an AutoMate component is presented in Figure 2. The functional aspect specification abstracts

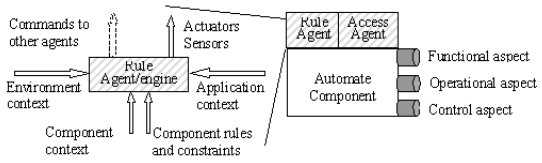


**Figure 2.** An AutoMate Component

component functionality, such as order of interpolation (linear, quadratic, etc.). This functional profile is then used by the compositional engine to select appropriate components based on application requirements. The operational aspect specification abstracts a component's operational behavior, including computational complexity, resource requirements, and performance (scalability). This profile is then used by the configuration and runtime engines to optimize component selection, mapping and adaptation. Finally, the control aspect describes the adaptability of the component and defines sensors/actuators and policies for management, interaction and control.

AutoMate components also encapsulate access policies, rules, a rule agent, and an access agent that allow the components to consistently and securely configure, manage, adapt and optimize their execution based on rules and access policies. The access agent is a part of the AutoMate access control engine and the underling dynamic access control model, and manages access to the component based on its current context and state (as discussed in Section 5). The rule agent is part of RUDDER, the AutoMate deductive engine (see Section 4) and manages local rule definition, evaluation and execution at the component level. Rules can be dynamically defined (and changed) in terms of the component's interfaces (based on access policies) and system and environmental parameters. Execution of rules can change the state, context and behavior of a component, and can generate events to trigger other rule agents. The rule agent is also responsible for managing, resolving rule conflicts using rule priorities and a dynamic rule-lock mechanism.

AutoMate components build on DIOS/DIOS++ [12, 6] which provides mechanisms to directly enhance traditional computational objects/components with sensors, actuators, rules, a control network that connects and manages the distributed sensors and actuators, and enables external discovery, interrogation, monitoring and manipulation of these components at runtime, and a distributed rule-engine that enables the runtime definition and deployment for managing and adapting application components. Application components may be distributed (spanning many processors) and dynamic (be created, deleted, changed or migrated at runtime). Access to a component's sensors and actuators is governed by its local access control policies along with global application level policies. Rules can be dynamically composed using sensors and actuators exported by application components. These rules are automatically partitioned and deployed onto the appropriate components using the control network, and evaluated by the distributed deductive engine.

### 3.2 Autonomic Compositions in AutoMate

Applications are typically composed with well defined objectives. In case of autonomic applications, however, these objectives can dynamically change based on the state of the application and/or the system. As a result, we need to dynamically select components and compose them at runtime based on current objectives. Together, the profiles, policies, and rules allow autonomous components to consistently and securely manage and optimize their executions. Furthermore, they enable applications to be dynamically composed, configured and adapted. Dynamic application work-flows [9] can be defined to select the most appropriate components based on user/application constraints (highest-performance, lowest cost, reservation, execution time upper bound, best accuracy), on the current applications requirements, to dynamically configure the component's algorithms and behavior based on available resources or system and/or applications state, and to adapt this behavior if necessary. The AutoMate dynamic composition model may be viewed as transforming a given composition or workflow into a new one by adding or modifying interactions and participating entities. Its primary goal is to enable dynamic (and opportunistic) choreography and interactions of components and services to react to the heterogeneity and dynamics of the application and underlying execution environment to produce the desired user objectives.

The AutoMate dynamic composition model is context aware and is based on policies and constraints that are defined, deployed and executed at runtime (see Figure 3). Composition policies and constraints are defined as simple rules and execute on the distributed deductive engine (Section 4) - i.e. there is no central authority that manages the composition process. These rules are defined in terms of the interfaces and aspects [5] exported by AutoMate components, the current context of the scenario and the overall objective of the application. Rules are simple and non-recursive, and can be composed and aggregated in a consistent way - based on logic and constraint based programming techniques [10]. Users can define and deploy rules at runtime provided they have



PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE

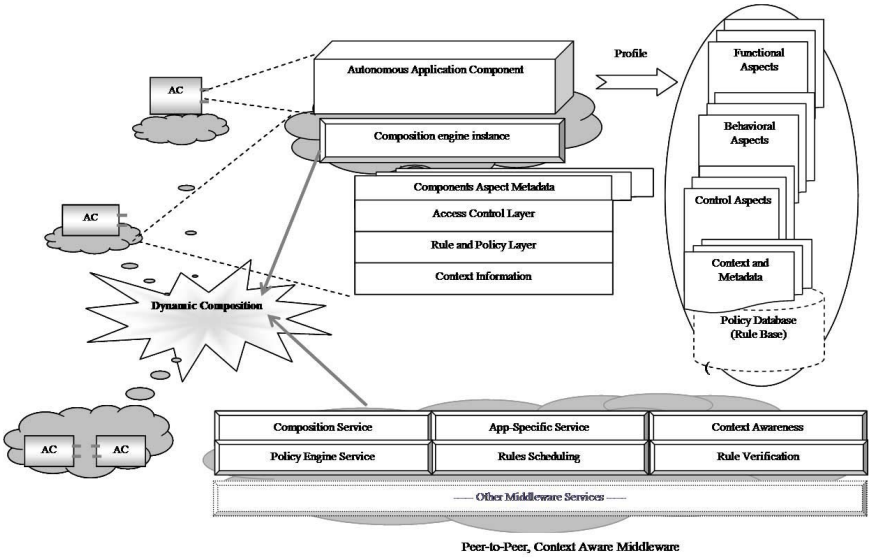


**Figure 3. Autonomic Compositions in AutoMate**

the required privileges, and the rules inherit the priorities and privileges of their owners [6]. Rules execute in a distributed fashion on a peer-to-peer deductive shell exported by the autonomic middleware as described below. Firing of rules causes the components to adapt, optimize, interact and compose. Composition metadata [9] is defined locally at the component level or globally at the application or the middleware level using a standard representation.

## 4 RUDDER: The Deductive Engine

RUDDER provides the core capabilities for supporting autonomic compositions, adaptations, and optimizations. It is a decentralized deductive engine composed of distributed specialized agents (component rule agents, composition agents, context agents and system agents) that exist at different levels of the system, and represents their collective behavior. It provides mechanisms for dynamically defining, configuring, modifying and deleting rules. Furthermore it defines an XML schema for composing rules and provides mechanisms for deploying and routing rules, decomposing and distributing them to relevant agents, and for coordinating the execution of rules. It also manages conflict resolutions within a single entity and across entities.

Figure 4 presents a schematic overview of RUDDER. It builds on AutoMate and Grid services and the underlying semantic messaging infrastructure. Rules can be dynamically injected into the system and are routed by the messaging substrate to the appropriate agents. Furthermore, the agents may hierarchically decompose a rule and distribute it to peer agents. For example, an application level rule may be decomposed into sub-rules that are assigned to its components. The components rules may be further decomposed into rules for the underlying systems entities.

RUDDER rules are defined using an XML rule schema and consists of the following tags: (<RULE identifier>, <priority>, <ON events>, <USE component/service>, <IF conditions>, <THEN actions>, and <ELSE default actions>). The *RULE identifier* uniquely identifies the rule in the system. The *priority* is assigned to the rule either by the user or the system. The event(s) specified within the *On* clause defines the trigger(s) for the rule. The *USE* clause specifies the component or service that will be evaluated when the rule is triggered. Finally, the *IF-THEN-ELSE* clause is used to specify conditions and actions. Local rule agents maintain rules and manage their creation, deletion, modification, activation and deactivation, and execution. When a rule is triggered and the associated conditional expression is satisfied, the set of actions specified are executed. A RUDDER agent supports learning mechanisms to enable dynamic self configuration and adaptation of rules and automatic definition of new rules based on component and application behavior. It also provides algorithms and mechanisms to combine the new rules with the current set of rules, and to resolve conflicts between rules within the entity.

Figure5 shows a sample RUDDER rule for managing distributed adaptive mesh refinement (AMR) application using GrACE [2]. In these applications, partitioners are used to dynamically partition the underlying adap-



PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP
ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE

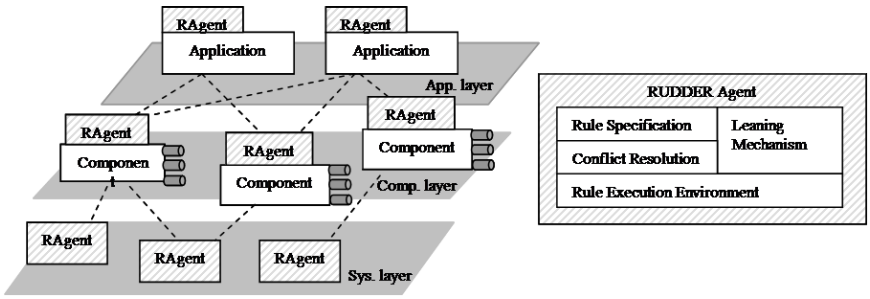


**Figure 4. The RUDDER Deductive Engine**

```
RULE GrACE_Switch_Partitioner
ON EVENT (Request_Switch_Partitioner)
USE Context_Service – Define_System_State
USE Component_Aspects – Define_Application State
If (Computation-communication ratio > cc_threshold) THEN
ACTION (PostEvent "PartitionerType=ISP")
ELSE
ACTION (PostEvent "PartitionerType=G-MISP+SP")
END IF
```

**Figure 5. A Sample RUDDER Rule**

tive grid hierarchy. The most appropriate partitioning algorithm and its configuration depends on application parameters, the current state of adaptive grid hierarchy and state of the system [19]. The sample rule manages the autonomic selection of the appropriate partitioners to match system and application state. The rule is triggered when an event *Request_Switch_Partitioner* arrives at the relevant rule agent. Each time the rule is triggered, it uses context awareness services and component aspects to get the current system and application state (current refinement characteristics and dynamics). It uses this information to compute the overall computation-communication ratio to characterize the application state as computation or communication dominate[19]. The condition checks the computation-communication ratio against a user defined threshold and an event is posted to set the appropriate partitioner type. The threshold value (*cc_threshold*) can be dynamically changed and may be adjusted by the rule agent based on recent history.

## 5 SESAME: Dynamic role-based access control engine

A key requirement of autonomic applications is the support for dynamic, seamless and secure interactions between the participating entities, i.e. components, services, application, data, instruments, resources and users. Ensuring interaction security requires a fine grained access control mechanism. Furthermore, in the highly dynamic and heterogeneous Grid environment, the access rights of an entity depends on the entity's privileges, capabilities, context and state. For example, the ability of a user to access a resource or steer a component depends on users' privileges (e.g. owner), current capabilities (e.g. resources available), current context (e.g. secure connection) and the state of the resource or component.The AutoMate Access Control Engine addresses these issues and provides dynamic access control to users, applications, services,components and resources. The engine is composed of access control agents associated with various entities in the system. The underlying dynamic role based access control mechanism extends the RBAC (Role Based Access Control) model[3, 17] to make access control decision based on dynamic context information. The access control engine dynamically adjusts *Role Assignments* and *Permission Assignments* as illustrated in Figure 6.

The subject is the entity which requests service from another entity. In AutoMate, the subject may be a user, application, service or component. The respective context agent is responsible for collecting an entity's current context information such as the state and current execution environment of a component or an application. Based on this context information, the access control agent dynamically adjusts the *user-role* and *role-permission* relationships to dynamically grant appropriate access permissions. Note that the access control agent (and context agent) is authenticated and delegated by the authority service (e.g. a Grid Authority Service).In our approach, each component is assigned a role subset (by the authority service) from the entire role set. Similarly the component has permission subsets for each role that will access the component. During a secure interaction, state machines are maintained by the access control agent at the subject (*Role State Machine*) to navigate the role subset, and the object (*Permission State Machine*) to navigate the permission subset for each active role. The state machine consists of state variables (role, permission), which encode its state, and commands, which transform its state. These state machines define the currently active role and its assigned permissions and navigate the role/permission subsets to react



PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP
ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE

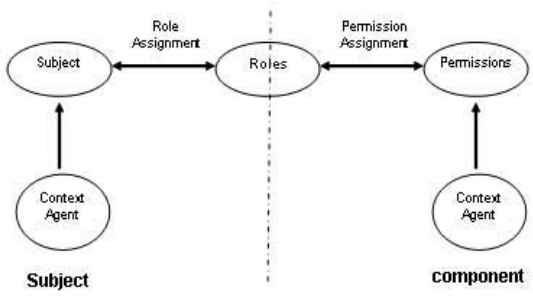


**Figure 6. Dynamic Access Control Model**

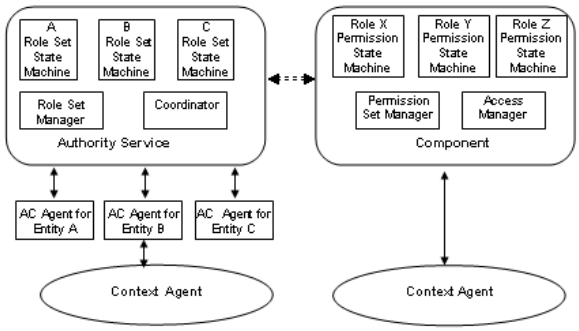


**Figure 7. Dynamic Access Control in AutoMate**

to changes in the context.

The operation of dynamic access control engine at the component layer is illustrated in Figure 7. This figure shows three AutoMate components, A, B, and C, each with their own access control agents and state machines assigned to them by the authority service. The access control agent maintains the role state machine for each component and defines its active role based on its current context. When the subject component accesses another component, it will first get its current role from its role state machine, and then use this role to access the component. At the accessed component, a permission state machine is defined (if it does not already exist) for the active role. For example, active roles X, Y, and Z have their own permission state machines at component. The access control agent at the accessed component will maintain this permission state machine to define the current permissions for a role based in its current context and state.

## 6 Pawn: A P2P Messaging Substrate

Pawn[11] is a peer-to-peer messaging substrate that builds on project JXTA[14] to support peer-to-peer interactions on the Grid. Pawn provides a stateful and guaranteed messaging to enable key application-level interactions such as synchronous/asynchronous communication, dynamic data injection, and remote procedure calls. It exports these interaction modalities through services at every step of the scientific investigation process, from application deployment, to interactive monitoring and steering, and group collaboration. A conceptual overview of the Pawn P2P substrate is presented in Figure 8 and is composed of peers (computing, storage, or user peers), network and interaction services, and mechanisms. These components are layered to represent the requirements stack enabling interactions in a Grid environment. The figure can be read from bottom to top as

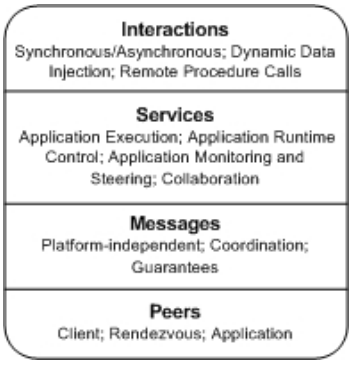


**Figure 8.** Pawn requirements stack

:"Peers compose messages handled by services through specific interaction modalities".

JXTA defines unicast pipes that provide a communication channel between two endpoints, and propagate pipes that can multicast a message to a peergroup. It also defines the Resolver Service that sends and receives messages in an asynchronous manner. The recipient of the message can be a specific peer or an entire peergroup. The pipe and resolver service use the available underlying transport protocol (TCP, HTTP, TLS) to transport messages from point to point. Pawn extends the pipe and resolver services to provide stateful and guaranteed messaging. This messaging is then used to enable the key application-level interactions such as synchronous/asynchronous communication, dynamic data injection, and remote procedure calls.

**Stateful Messages:** In Pawn, messages are platform-independent, and are composed of source and destination identifiers, a message type, a message identifier, a payload, and a handler tag. The handler tag uniquely identifies the service that will process the message. State is maintained by making every message a self-sufficient and self-describing entity that, in case of a link fail-



PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP
ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE

ure, can be resent to its destination by an intermediary peer without the need to be re-composed by its original sender. In addition, messages can include system and application parameters in the payload to maintain application state.

**Message Guarantees:** Pawn implements application-level communication guarantees by combining stateful messages and a per-message acknowledgment table maintained at every peer. FIFO message queues are used to handle all incoming and outgoing messages. Every outgoing message that expects a response is flagged in the table as awaiting acknowledgment. This flag is removed once the message is acknowledged. Messages contain a default timeout value representing an upper limit on the estimated response time. If an acknowledgment is not received and/or the timeout value expires, the message is resent. The message identifier is a composition of the destination and sender's unique peer identifiers. It is incremented for every transaction during a session (interval between a peer joining and leaving a peergroup) to provide application-level message ordering guarantees.

**Synchronous/Asynchronous communication:** Pawn combines JXTA communication semantics, synchronous (using blocking pipes) or asynchronous (using non-blocking pipes) interactions, with its stateful messaging and message guarantees mechanisms to provide reliable messaging enabling the desired higher-level application interactions.

**Dynamic Data Injection:** In Pawn, every peer advertisement contains a pipe advertisement, which uniquely identifies an input and output communication channel to the peer. This pipe is used by other peers to create an end-to-end channel to dynamically send and receive messages.

Every interacting peer implements a message handler that listens for incoming messages on the peer's input pipe channel. The message payload is passed to the application/service identified by the handler tag field at runtime.

**Remote Method Calls (PawnRPC):** The PawnRPC mechanism provides the low-level constructs for building applications interactions across distributed peers. Using PawnRPC, a peer can dynamically invoke a method on a remote peer by passing its request as an XML message through a pipe. The interfaces for the methods that are exported by a peer are published as part of the peer advertisement during peer discovery. The PawnRPC XML message is a composition of the destination address, the remote method name, the arguments of the method, and the arguments associated types. Upon receiving a PawnRPC message, a peer locally checks the credentials of the sender, and if the sender is authorized, the peer invokes the appropriate method and returns a response to the requesting peer. The process may be done in a synchronous or asynchronous manner. PawnRPC uses the messaging guarantees to assure delivery ordering, and stateful messages to tolerate failure.

## 7 SQUID: Decentralized Discovery Service

A fundamental problem in large, decentralized, distributed resource sharing environments such as the Grid, is the efficient discovery of information, in the absence of global knowledge of naming conventions. For example a document is better described by keywords than by its filename, a computer by a set of attributes such as CPU type, memory, operating system type than by its host name, and a component by its aspects than by its instance name. The heterogeneous nature and large volume of data and resources, their dynamism (e.g. CPU load) and the dynamism of the Grid make the information discovery a challenging problem. An ideal information discovery system has to be efficient, fault-tolerant, self-organizing, has to offer guarantees and support flexible searches (using keywords, wildcards, range queries). Decentralized peer-to-peer (P2P) systems, by their inherent properties (self-organization, fault-tolerance, scalability), provide an attractive solution.

SQUID [18] supports decentralized information discovery in AutoMate. It is a P2P system that supports complex queries containing partial keywords, wildcards, and range queries, and guarantees that all existing data elements that match a query will be found with bounded costs in terms of number of messages and number of nodes involved. The key innovation is a dimension reducing indexing scheme that effectively maps the multi-dimensional information space to physical peers.

The overall architecture of SQUID is a distributed hash table (DHT), similar to typical data lookup systems [15, 20]. The key difference is in the way we map data elements[1] to the index space. In existing systems, this is done using consistent hashing to uniformly map data element identifiers to indices. As a result, data elements are randomly distributed across peers without any notion of locality. Our approach attempts to preserve locality while mapping the data elements to the index space. In our system, all data elements are described using a sequence of keywords (common words in the case of P2P storage systems, or values of globally defined attributes

[1]The term 'data element' is used to represent a piece of information that is indexed and can be discovered. A data element can be a document, a file, an XML file describing a resource, an URI associated with a resource, etc.



PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP
ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE

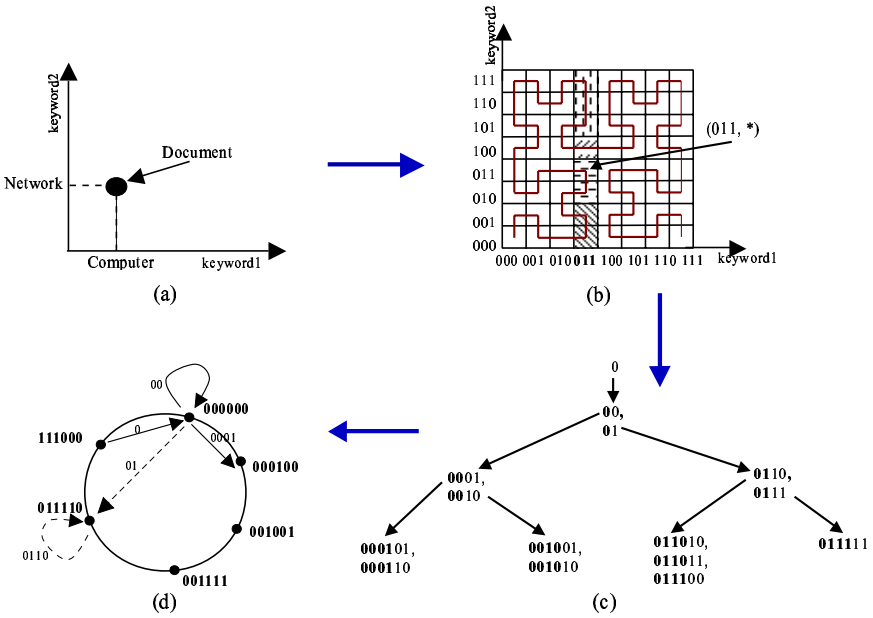


**Figure 9.** (a) A 2-dimensional keyword space. The data element *"Document"* is described by keywords *"Computer"* and *"Network"*; (b) Mapping the 2-dimensional space to a curve. The query (011, *) defines clusters on the curve (segments); (c) Recursive refinement of query (011, *) viewed as a tree. Each node is a cluster, and the bold characters are the cluster's prefixes; (d) Solving the query: embedding the leftmost tree path (solid arrows) and the rightmost path (dashed arrows) onto the overlay network topology.

- such as memory and CPU frequency - for resource discovery in computational grids). These keywords form a multidimensional keyword space where the keywords are the coordinates and the data elements are points in the space. Two data elements are "local" if their keywords are lexicographically close or they have common keywords. Thus, we map documents that are local in this multi-dimensional index space to indices that are local in the 1-dimensional index space, which are then mapped to the same node or to nodes that are close together in the overlay network. This mapping is derived from a locality-preserving mapping called Space Filling Curves (SFC) [16]. In the current implementation, we use the Hilbert SFC [16] for the mapping, and Chord [20] for the overlay network topology.

Note that locality is not preserved in an absolute sense in this keyword space; documents that match the same query (i.e. share a keyword) can be mapped to disjoint fragments of the index space, called clusters. These clusters may in turn be mapped to multiple nodes so a query will have to be efficiently routed to these nodes. SQUID optimizes the querying process using successive refinement and pruning of the queries. These optimizations significantly reduce the number of nodes queried. The overall operation of SQUID is presented in Figure 9.

Unlike the consistent hashing mechanisms, SFC does not necessarily result in uniform distribution of data elements in the index space - certain keywords may be more popular and hence the associated index subspace will be more densely populated. As a result, when the index space is mapped to nodes load may not be balanced. SQUID provides a suite of relatively inexpensive load-balancing optimizations and experimentally demonstrate that they successfully reduce the amount of load imbalance.

## 8 Autonomic Computational Science and Engineering using AutoMate

A goal of AutoMate is to support autonmic computational science and engineering. For example, it is currently being used to develop the ARMaDA framework for the autonomic runtime management of structured adaptive mesh refinement (SAMR) applications [2, 22]. These applications are heterogeneous and dynamic, and require runtime adaptations and optimizations. ARMaDA monitors, analyzes, characterizes the state of the SAMR application using aspects exported by the autonmic components. It also monitors and characterizes the state of the system using the context awareness engine. Current system and application state is then used to reactively and proactively manage application execution and improve overall application performance. Adaptation/optimizations include defining application parame-



PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP
ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE

ters, distribution strategies, partitioning and scheduling mechanisms, and communication/synchronization algorithms.

## 9 Conclusions and Current Status

The computational solutions addressed by the AutoMate project are based on fundamental innovations in the development, optimization and deployment of component-based Grid applications, thereby allowing the heterogeneity and dynamics of the applications to match that of the Grid and fully exploit its potential. These innovations will enable scientists to choreograph high performance, integrated end-to-end simulations that were never possible or attempted before. The key IT contributions are the methodology and associated technologies that enable the development of applications that can manage and exploit the dynamism and heterogeneity of the Grid, and that address the extremely serious problem of software complexity that is threatening both academia and industry.

We currently have working prototypes of each of the components presented in this paper, and are in the process of integrating them to support autonomic structured adaptive mesh refinement applications (SAMR) in science and engineering. Further information about AutoMate and its components can be obtained from http://automate.rutgers.edu/.

## References


[1] Autonomic computing: IBM's perspective on the state of information technology. http://researchweb.watson.ibm.com/autonomic/.

[2] S. Chandra and M. Parashar. ARMaDA: An adaptive application-sensitive partitioning framework for structured adaptive mesh refinement applications. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing Systems (PDCS 02)*, pages 446–451. ACTA Press, November 2002.

[3] M. J. Covington, M. J. Moyer, and M. Ahamad. Generalized role-based access control for securing future applications. In *In Proceedings of the 23rd National Information Systems Security Conference (NISSC)*, October 2000.

[4] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[5] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242, Berlin, Heidelberg, and New York, 1997. Springer-Verlag.

[6] H. Liu and M. Parashar. Dios++: A framework for rule-based autonomic management of distributed scientific applications. Submitted for publication, 2003.

[7] V. Mann, V. Matossian, R. Muralidhar, and M. Parashar. DISCOVER: An environment for Web-based interaction and steering of high-performance scientific applications. *Concurrency and Computation: Practice and Experience*, 13(8–9):737–754, 2001.

[8] V. Mann and M. Parashar. Engineering an interoperable computational collaboratories on the grid. *Concurrency and Computation: Practice and Experience, Special Issue on Grid Computing Environments*, 14(13-15):1569–1593, 2002.

[9] D. C. Marinescu. *Internet-Based Workflow Management: Towards a Semantic Web*. John Wiley & Sons, 2002.

[10] K. Marriott and P. J. Stuckey. *Programming with Constraints: an Introduction*. MIT Press, 1999.

[11] V. Matossian and M. Parashar. Enabling peer-to-peer interactions for scientific applications on the grid. Submitted for publication, 2003.

[12] R. Muralidhar and M. Parashar. A distributed object infrastructure for interaction and steering. In *Proceedings of the 7th International Euro-Par Conference*, volume 2150, pages 67–74, August 2001.

[13] A. Popovici, T. Gross, and G. Alonso. Dynamic weaving for aspect-oriented programming. In *Proceedings of the 1st international conference on Aspect-oriented software development*, pages 141–147. ACM Press, 2002.

[14] Project jxta. http://www.jxta.org.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[16] H. Sagan. *Space-Filling Curve*. Springer Verlag, May 1995.

[17] R. Sandhu, D. Ferraiolo, and R. Kuhn. The nist model for role-based access control: towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control*, pages 47–63. ACM Press, 2000.

[18] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. Accepted for publication at HPDC-12, 2003.

[19] J. Steensland, S. Chandra, and M. Parashar. An application-centric characterization of domain-based sfc partitioners for parallel samr. *IEEE Transactions on Parallel and Distributed Systems*, 13(12):1275–1289, 2002.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SICOMM'01 Conference*, pages 149–160, San Diego, California, August 2001.

[21] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman. Grid service specification, February 2002.

[22] H. Zhu, M. Parashar, J. Yang, Y. Zhang, S. Rao, and S. Hariri. Self-adapting, self-optimizing runtime management of grid applications using pragma. accepted for publication in the Proceedings of the NSF Next Generation Systems Program Workshop, IEEE/ACM International Parallel and Distributed Processing Symposium, April 2003.




PROCEEDINGS OF THE AUTONOMIC COMPUTING WORKSHOP FIFTH ANNUAL INTERNATIONAL WORKSHOP ON ACTIVE MIDDLEWARE SERVICES (AMS'03) ISBN 0-7695-1983-0/01 $17.00 © 2003 IEEE