Autonomic Computing
Research Issues, Challenges and Opportunities

S. Hariri and M. Parashar
AICCSA’03 Autonomic Computing Tutorial
July, 2003

Issues, Challenges and Opportunities

• Programming paradigms and development models
  - Autonomic component/service definition and construction
  - Autonomic application development
  - Policies/constraints definition and representation
• Application construction, execution and management
  - Dynamic (policy/rule-based) configuration, execution and optimization
  - Fault tolerance
• Autonomic Middleware Services
  - Messaging Services
  - Discovery Services
  - Security Services
  - Management Services
  - ...

Software Composition Models

Component Model  Composition Technology

Component Integration Environment
Component Model

- A distributed component-oriented model
  - An architecture for defining autonomic components and their interactions
  - A technology for deploying and executing autonomic components
  - Services for component lifecycle control, (de)activation, security, persistence and events

Autonomic Components/Services

- What is an autonomic component
- How to construct an autonomic component
- Runtime environment of autonomic components
- Examples of component based systems

Defining an Autonomic Component

- An autonomic component is the fundamental atom of autonomic applications and systems. It is a modular unit of composition with contractually specified interfaces, explicit context dependencies and mechanisms for self-management, responsible for providing its services, constraints (system resource requirements, performance requirements etc.), managing its own behavior in accordance with context, rules and policies, and interacting with other autonomic components.
Autonomic Components

- Component
  - Encapsulation
  - Information hiding
  - Modularity
- Autonomic Component
  - self/context aware
  - self managing (configuring, adapting, optimizing, protecting, healing)
    - autonomic components provide embedded mechanisms for self-management based on local state and context information to meet behavior and performance requirements
  - open
    - comply with standards

Constructing Autonomic Components

- Autonomic components export information and policies about their behavior, resource requirements, performance, interactivity and adaptability to system and application dynamics
  - functional aspects
    - abstracts component functionality, such as order of interpolation (linear, quadratic, etc.)
    - used by the compositional engine to select appropriate components based on application requirements
  - operational aspects
    - abstracts a component's operational behavior, including computational complexity, resource requirements, and performance (scalability)
    - used by the configuration and runtime engines to optimize component selection, mapping and adaptation
  - control aspect
    - describes the adaptability of the component and defines sensors/actuators and policies for management, interaction and control.

- Autonomic components encapsulate access policies, rules, a rule agent, and an access agent
  - enables components to consistently and securely configure, manage, adapt and optimize their execution based on rules and access policies
  - rules/policies can be dynamically defined (and changed) in terms of the component’s interfaces (based on access policies) and system and environmental parameters
  - rule execution may change the state, context and behavior of a component, and can generate events to trigger other rule agents
  - rule agent manages rule execution and resolves rule conflicts
Self-Management Approaches

- **Passive:**
  - Provide sensors for external accesses to collect component information
  - Provide actuators for external operations to control component behavior
- **Active:**
  - Collect external (local) status information through self-observation or collective-observation. Collect internal status information through sensors
  - Corresponding actions are issued based on this information in accordance with defined rules/policies/constraints
- **Proactive:**
  - Automatically adjust behavior in anticipation of future problems, needs or changes, based on history and/or predictive functions.

Aspects Description

- **Functional Aspect**
  - WSDL is an XML document that describes web services
  - SIDL is an IDL that describes the calling interfaces for a scientific library
  - Others:
    - HTML (too generic & simple), Programming language (suitability to description), Unstructured language (lack of semantics)
- **Control Aspect**
  - similar to functional aspect
- **Operational Aspect**
  - Describe resource and performance requirements - XML
  - Describe security policies - XACML
  - Describe rules - RuleML, ERML

Autonomic Components - Issues

- **Management Issues**
  - Naming and discovery, life cycle, failure recovery
- **Communication/Interaction Issues**
  - Communication mechanism, messaging model
- **Collaboration Issues**
  - Conflict resolution, load balancing, migration
**Autonomic Component Management**

- **Naming and discovery service**
  - Component registers itself to export its services, interfaces, aspects, sensors & actuators. Component can be discovered through keywords, name, or fuzzy matching.
- **Lifecycle service**
  - During its lifetime, component should be available and consistent. Component can be terminated securely.

**Autonomic Component Communication**

- **Communication mechanisms**
  - RMI
  - Shared spaces
  - SOAP
  - RPC
  - …
- **Messaging model**
  - Push or pull model
  - Publisher and subscriber model
  - …
- **Synchronous / asynchronous communication**

**Autonomic Component Collaboration**

- **Load balancing**
  - Component migration
- **Conflict resolution**
  - Priority
  - Dynamical lock mechanism
- **Failure recovery**
  - Hot-swapping
  - Checkpoint/Rollback mechanism
**Autonomic Components – Key Challenges**

- Autonomic components formulation
  - Models, formalisms, meta-data specifications,
- Lifecycle of an autonomic component
  - Design & implementation, testing & verification, deployment, bootstrapping, setup, operation, updating, termination
- Lifecycle of autonomic component relationship
  - Specify & register, locate, negotiate, provision, operate, terminate

**Examples of component based systems**

- CCA (www.ccaforum.org)
  - Components interact through provide/use ports
  - Provide language interoperability through SIDL
  - Framework provides the means to hold and compose components into applications, and also provides a set of standard services to components
  - Based on the CCM
- ABLE (IBM)
  - Built on JavaBeans model, ABLE has component library of data access, machine learning, machine reasoning, and optimization algorithms packaged as JavaBeans (AbleBeans).
  - ABLE provides GUI for creating and configuring AbleBeans, and for constructing and testing the agents built from them.
  - ABLE also provides an agent platform for deploying agents across a distributed computing system.
### Autonomic Applications

### Developing Autonomic Application - Issues

- Specification of the autonomic application
  - Representing/specifying the abstract application
- Components discovery, selection, and composition
  - Automatically search for and select appropriate components
- Application execution
  - Adapting suitable execution model for composed application
  - Component communication, interaction and reconfiguration
    - Dynamic, opportunistic, ephemeral, p2p/hierarchical, etc.
- Application monitoring, control, behavior prediction
  - Context information collection and analysis
  - Proactive system configuration
- Reliability, fault tolerance, availability

### Composition Technology

- Composition model
  - Static composition: compositions defined at design time
  - Dynamic composition: compositions defined at runtime
- Composition description
  - Petri nets
  - Workflow language
    - WSFL, GSFL, GridFlow, etc.
Existing support

- **Web Services Domain**
  - WSFL and WSDL (IBM): XML language for the description of Web Services compositions
  - BPML and WSCI: complementary XML languages designed to allow for complete definition of a business process model
  - XLANG (Microsoft): provides an extension service element to WSDL that describes how the process will work as part of a business flow
  - JSFL: XML based notation for describing composite jobs made up of interacting services
  - ebXML: essentially has a specification to handle every aspect of the electronic-business transaction, from describing simple service endpoint definition, to describing the entire Business Process Workflow for the transaction
  - BPMN: provides a graphical notation for expressing business processes in a Business Process Diagram (BPD)

Existing support, contd.

- **Grid Services Domain**
  - GSFL and GSDL: XML-based language that allows the specification of workflow descriptions for Grid services in the OGSA framework.
  - XCAT Application Factories: address workflow issues for Grid-based components within the CCA framework, allowing components to be connected to each other dynamically

Existing support, contd.

- **WSCL**
  - A conversation language framework under development by the Hewlett-Packard Company, for modeling the sequencing of the interactions or operations of one interface

- **DAGMan**
  - meta-scheduler for Condor that manages dependencies between jobs

- **Chimera Virtual Data System (GriPhyN)**
  - It considers compositions as graphs of services.

- **Associative broadcast based Coordination model**
  - Integrated the coordination with composition
  - Associative broadcast is used to target messages to processes in specific states, which are in turn used to form composition.
Existing support, contd.

• DySCo
  – enables dynamic service composition and is based on the idea of functional incompleteness and multi-party orchestration

• SWORD
  – uses a rule-based expert system to find composition plans and each service is represented as a logical rule that expresses the inputs and outputs associated with it.

• Symphony
  – Java based composition and manipulation framework based on the Sun JavaBeans component architecture

Autonomic Application Composition

• Dynamically and opportunistically composed from autonomic components
  – ad hoc, negotiated, …

• Composition based on policies/rules/constraints defined by system and/or application

• Composition will be aware of available resources, components and their current states and capability

Dynamic Rule-based Composition

• Motivation:
  – Autonomic applications dynamically change based on the state of the system
  – Runtime access to and modifications of components and application
  – Composition described by scripts or programs limit the level of flexibility
  – Rules defined by system/user enable automatically adaptive composition
Dynamic Rule-based Composition

- Challenges:
  - How to specify the changes in the objective to create dynamic composition
  - How to guarantee consistency of environment after submitting change
    - Deadlock detection/prevention
    - Serious errors tolerance, such as no termination
  - How to define and deploy the rules to satisfy the application requirement and optimize performance

Component Integration Environment

- Component selection
- Component communication
- Runtime rules/policy/constraint definition and deployment
- Adaptive rules/policy/constraint modification and execution
- Firing of rules causes components/application to adapt, optimize, interact and compose
- Proactive system management based on automatically generated rules and constraints

Fault Tolerance and Autonomic Computing
Fault Tolerance in Autonomic Computing

- Autonomic Fault Detection
  - Automatic determine that a fault occurred
- Autonomic Fault Diagnosis
  - Automatic analysis the cause of fault
- Autonomic Fault Containment
  - Prevent the propagation of fault
- Autonomic Fault Masking
  - Ignore the service from failed component
- Autonomic Fault Compensation
  - Compensate the service provided by the failed component
- Autonomic Fault Repair
  - Remove the fault from system

Fault Classes

- Locality
  - Atomic Component Faults
  - Composite Component Faults
  - System Level Faults
  - External Faults
- Effects
  - Value Faults
  - Timing Faults
- Immediate Cause
  - Resource depletion faults
  - Logic faults
  - Physical faults
- Ultimate Cause
  - Specification fault

Fault Tolerance: Challenge

- Heterogeneous network
  - Every computing domain is autonomic
- Huge amount of information
  - Boundary-less information system
- Computing units are highly distributed
  - Non Centralized management
- Detection/notification of abnormality is hard
  - Dynamic change in network topology
- Need for reliable fault tolerance communication facility
Example Approach: Consensus Among Neighbors

- Form a group with at least three providers
- Within a group, healthy members recognize crazy one
  - e.g. by consensus
- Healthy members make up a new group
  - excluding faulty member and invite a healthy provider
- No need of inquiry or notification
- Can take the advantage of design diversity
- Heavy communication overhead

Autonomic Middleware Services

- Messaging
- Discovery
- Security
- …
Messaging for Autonomic Computing

- The Messaging is responsible for
  - Transporting messages to/from endpoints, with or without guarantees
  - Defining the format of messages
  - Filtering and/or aggregating

- Messaging architectures are of two types
  - Publisher/Subscriber systems
    - e.g. SIENA, GRYPHYON, LeSubscribe
  - Point to Point systems
    - e.g. Email, RPC

- Existing Messaging Frameworks include
  - ICENI, Xevents/Xmessages, Pawn, NaradaBrokering, Java Message Service, IBM MQSeries

Messaging Semantics

- PUSH
  - Information is “pushed” to an interested node as soon as it is available

- PULL
  - Interested nodes query an information provider for the presence of data

- REQUEST/RESPONSE
  - Nodes query an information provider and wait for a response

- TRANSACTION
  - A Request/Response with certain associated guarantees

- FILTERED MULTICAST
  - Data is sent through filters to a group of interested nodes

Message Distribution

- How to distribute messages to interested nodes?
  - Multicast
  - Flooding/Broadcast
  - Overlay Network Routing
    - CHORD, CAN, TAPESTRY, SQUID
  - Filtered Propagation

- How to guarantee that all subscribers to an information source have received a message?
  - No guarantees (Datagram Packets, UDP)
  - Per message acknowledgement
  - Aggregate acknowledgement (every x messages)
Messaging: SIENA

- Publisher/subscriber system
- Uses notification and subscription models to distribute information
- Uses filters and aggregating access points to provide scalability
- Content-based networking

Source: http://www.cs.colorado.edu/serl/siena/

Messaging: Gryphon

- Publish/subscribe middleware aimed at distributing large volumes of data in real time.
- Features
  - Topic based and content based publish/subscribe
  - Publish/subscribe system deployed on a public network cannot depend on homogenous router technology.
  - Use of tcp/ip or http.
  - Scale support to application growth.
  - Provide security and privacy features to a degree not mandated over private secured networks.
  - Client authentication, access controls and encryption/integrity of messages
- Implementation
  - Java Message Services (JMS) API.
  - Patented Matching Engine provides high speed content filtering.


Messaging: Gryphon

- Scalability
  - Brokers may be added into the network to provide support for additional clients.
- Availability
  - Responds to the failure of one broker in a network by rerouting traffic around the failed broker. Reconfiguration is automatic and requires no intervention by an administrator.
- Security
  - Supports access controls for limiting who may publish and subscribe to portions of the information space.
  - Supports SSL
**Messaging: Le Subscribe**

- A flexible LDAP-like publication model
- Subscriptions are conjunctions of predicates
  - Supports usual predicates plus is-kind-of and contains predicates
- Scalable and efficient matching supporting high rate of events with a large number of subscriptions
- Fast reactivity
  - Updates of the publication schema or current subscription set are immediately taken into account by the system.

Source: [http://www-caravel.inria.fr/LeSubscribe/](http://www-caravel.inria.fr/LeSubscribe/)

---

**Messaging Frameworks: ICENI**

- A component framework that separates the concerns of component interface, behavior and implementation into well defined XML schemas while enabling visualization and computational steering
- A federating Grid middleware built from Java and Jini that enables an organization resources to be exposed as services with well-defined usage and access control policies
- e-Science portals that utilize the component meta-data and services within ICENI to simplify access to Grid resources and applications
- Interoperability of the services and meta-data within ICENI to other service oriented architectures - such as the Open Grid Services Architecture (OGSA)
- The development of higher-level services, such as application mappers and resource brokers
Messaging Frameworks: NaradaBrokering

- Event brokering system designed to run on a large network of cooperating broker nodes
- Based on the distributed publish/subscribe paradigm and also supports the flexibility of centralized interactions as well as peer-to-peer (P2P) interactions over the edge
- JMS and JXTA compliant
- Support for raw RTP clients

Messaging: PAWN

- Publisher/Subscriber system supporting content-based networking
- Builds on Project JXTA
- Extends JXTA to provide distributed object controls
  - Serialization of Computational Object to XML
  - Remote Method Invocations (PawnRPC)
- Provides high-level messaging semantics for P2P applications on the Grid

Messaging: SOAP

- Simple Object Access Protocol Version 1.2
  - Lightweight protocol intended for exchanging structured information in a decentralized, distributed environment
  - Uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols
  - The framework has been designed to be independent of any particular programming model and other implementation specific semantics
  - Simple and extensible
- However, it does not provide
  - "reliability", "security", "correlation", "routing", and "Message Exchange Patterns"
**Issues: Messaging: Elections**

- Several peers may offer a similar service
  - Which one should be elected?

<table>
<thead>
<tr>
<th>Application</th>
<th>System</th>
<th>Middleware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare states between different service providers locally and come to a common decision for the definite elected peer</td>
<td>Coordinate messages</td>
<td>Distribute peer profiles to peers offering services for comparison and election of relevant algorithm</td>
</tr>
</tbody>
</table>

**Issues: Messaging: Consensus**

- Several peers may have a result for a similar computation
  - Which result should be considered final?
  - How to eliminate the potential differences between the results?

<table>
<thead>
<tr>
<th>Application</th>
<th>System</th>
<th>Middleware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receive output from other applications</td>
<td>Select messages to services computing similar results</td>
<td>Establish differences between results</td>
</tr>
</tbody>
</table>

**Issues: Messaging: Mutual Exclusion**

- When several peers require access to a resource
  - How to guarantee that the resource does not get accessed by several peers concurrently?

<table>
<thead>
<tr>
<th>Application</th>
<th>System</th>
<th>Middleware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock access to critical resource or critical region</td>
<td>Notify peers that requested access that the resource is currently locked and notify resource that other peers request access</td>
<td>Distribute resource information to interested peers</td>
</tr>
</tbody>
</table>
**Other Issues**

- **Selection**
  - Several peers may offer a similar service - Which one should be elected?

- **Consensus**
  - Several peers may have a result for a similar computation - Which result should be considered final?
  - How to eliminate the potential differences between the results?

- **Mutual Exclusion**
  - When several peers require access to a resource
  - How to guarantee that the resource does not get accessed by several peers concurrently?

**Opportunistic Interactions**

- **Interactions based on local goals and objectives**
  - local goals and objectives are defined as constraints to be satisfied
  - constraints can updated and new constraints can defined at any time

- **Dynamic and ad-hoc**
  - interactions use “semantic messaging” based on proximity, privileges, capabilities, context, interests, offerings, etc.

- **Opportunistic**
  - constraints are long-term and satisfied opportunistically (may not be satisfied)

- **Probabilistic guarantees and soft state**
  - no explicit synchronization
  - interaction semantics are achieved using feedback and consensus building
Discovery Mechanisms

- Centralized: based on client-server model
  - Centralized directories, they may synchronize periodically
  - Example: Universal Description, Discovery and Integration of Web Services (UDDI)

- Peer-to-Peer Systems
  - Unstructured (Gnutella-like):
    - Unstructured overlay network, use flooding
    - Pros: easy to maintain, supports complex queries
    - Cons: no search or cost guarantees

Discovery Mechanisms (contd.)

- Peer-to-Peer Systems
  - Hybrid (Napster):
    - Unstructured overlay network, use centralized directories for search
    - Pros: can support complex queries
    - Cons: not scalable

- Data-lookup (CAN, Chord, Pastry, etc)
  - Structured overlay, Internet-scale DHT
  - Pros: efficient lookup with guarantees
  - Cons: complex queries not supported

- Structured keyword search:
  - Structured overlay, extend data-lookup protocols
  - Examples:
    - Distributed Inverted Indexes
    - Space Filling Curves

Discovery: Data Lookup Systems

- CHORD
  - Uses consistent hashing to map data onto an identifier ring

- Content-Addressable Network (CAN)
  - Maps data on a d-dimensional Cartesian space

- PAISTRY
  - Uses Plaxton’s algorithm to perform classless routing in the overlay network

- SQUID
  - Maps documents in a d-dimensional space
  - Uses Space Filling Curves to store values onto available nodes
    - Locality preserving
    - Causality preserving
**Distributed Indexing**

- **Properties**
  - Parameter
  - Logical Path Length
  - Neighbor state
  - Routing Overhead (RDP)
  - Messages to insert
  - Mutability
  - Load-balancing

<table>
<thead>
<tr>
<th></th>
<th>Tapestry</th>
<th>Chord</th>
<th>CAN</th>
<th>Pastry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Log_b(N)</td>
<td>Log_b(N)</td>
<td>Dimen d</td>
<td>Base b</td>
</tr>
<tr>
<td>Logical Path Length</td>
<td>Log_b(N)</td>
<td>Log_b(N)</td>
<td>O(d⁺N¹/²)</td>
<td>Log_b(N)</td>
</tr>
<tr>
<td>Neighbor state</td>
<td>bLog_b(N)</td>
<td>Log_b(N)</td>
<td>O(d)</td>
<td>bLog_b(N) + O(b)</td>
</tr>
<tr>
<td>Routing Overhead (RDP)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)?</td>
</tr>
<tr>
<td>Messages to insert</td>
<td>O(Log_b(N))</td>
<td>O(Log_b(N))</td>
<td>O(d⁻¹⁺N¹/²)</td>
<td>O(Log_b(N))</td>
</tr>
<tr>
<td>Mutability</td>
<td>App-dep.</td>
<td>App-dep</td>
<td>Immut.</td>
<td>??</td>
</tr>
<tr>
<td>Load-balancing</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
</tbody>
</table>

*Designed as P2P Indices*

**Security Issues in Autonomic Computing**

- **Security Objectives**
  - Confidentiality: Prevention of unauthorized disclosure of information
  - Integrity: Prevention of unauthorized modification of information
  - Availability: Prevention of unauthorized withholding of information or resources

- **Security Issues**
  - Authentication
  - Authorization, Access Control
  - Intrusion Detection
  - Security Policy Definition and Reasoning
  - Resistant to Fraud and Persuasion
  - Privacy
  - Misc. Digital Signature, Non-repudiation
    - Crucial for e-commerce applications

**Authentication**

- Authenticate an entity (service/component/user) in the system
  - Password, Private Key (Kerberos)
- Trust between autonomic entities
  - PGP, X.509 PKI
- Key management
  - Key distribution, Key negotiation protocol
- Delegation
  - One entity delegate privilege to another entity.
Authorization, Access Control

- Environment is heterogeneous and dynamic
- Large numbers of distributed entities
  - Centralized authorization is not sufficient
- Global name space has constraints.
- Access control should be context aware
  - DAC, MAC, RBAC
- Fine grained access control mechanism
  - Dynamic Role Based Access Control

DRBAC Model

Access Control Engine Architecture
Intrusion Detection

- Autonomic detection, without human intervention
- Proactively notify the vulnerability of the system
- Restore the compromised component automatically
- Provide security analysis service at the entity to make distributed intrusion detection feasible
- Will be a challenge without help of human
- IBM’s approach - BlueBox: Policy-driven, Host-Based Intrusion Detection System
  - Policy driven technique is similar with the Java sandboxing

Security Policy Definition and Reasoning

- Self protecting based on security policy
- Policy will be different across domain
  - Standard Policy Language will be crucial
- Policy composition
  - Based on predict logic; hard issue
- Policy conflict resolution
  - Without violating security policy of involved entities
- Policy negotiation between component
  - Standard protocol

Resistant to Fraud and Persuasion

- Attacker will harness the ability of an entity to subvert the system
  - Entity will ensure the execution of compromised policy with every resource at its disposal
- Security policy of component should be prevented from compromise
  - High level subversion has catastrophic damage to system security
- Properly secured, the security policies would improve the resistance of component to attack
  - Advantage from autonomic computing
Privacy

- Component private information should be protected
  - Definition of the private information is different
- Processing personal data without violating the privacy policy automatically
  - Reliable, secure, without human involvement
- Take complex political and geographical situations into account
  - Not a technique issue only, sociology, law, nationality related.
- Privacy negotiation
  - Significant challenge in real system

Bibliography

- Autonomic computing
  - "Autonomic computing and Grid", Pratap Pattnaik, Kattamuri Ekanadham, and Joefon Jann, Thomas J. Watson Research Center, Hawthorne, New York
  - "Autonomic computing: concepts, opportunities, and challenges"
  - "Autonomic computing: current research and future challenges", Jeff Kephart, IBM research

Bibliography (contd.)

- "Helping Computers Help Themselves," David Pescovitz, Contributing Editor, Special R&D Report
Bibliography (contd.)

- **Autonomic Computing Projects**
  - Autonomia: [http://www.ece.arizona.edu/~hpdc/projects/AUTONOMIA/](http://www.ece.arizona.edu/~hpdc/projects/AUTONOMIA/)
  - “DOA: Towards breaking the curse of dimensionality,” A. Hinneburg and D. A. Keim, Proceedings of the International Conference on Data Engineering, 1999

- **Application Specific Projects**
  - “A parallel CORBA component model for numerical code coupling”, Christian Perez, Theory and Practice, the 3rd International Workshop on Grid computing, number of 2536 of LNCS, Baltimore, Maryland, pages 88-98, November 2002
Bibliography (contd.)

• Component
  – “The Common Component Architecture (CCA) and XCAT” Lawrence Berkeley National Laboratory, March 2003 http://www.extreme.indiana.edu/xcat/

• Economics

• Programming

• Grid

• Middleware
Bibliography (contd.)

• Agent Systems

Bibliography (contd.)

• Workflow

Bibliography (contd.)

• Language
  – Rule Markup Language: http://www.21st-century.de/rml
  – SRML: http://www.icsi.berkeley.edu/SRML

Bibliography (contd.)

• Composition
Bibliography (contd.)

Discovery
- Gnutella webpage: http://gnutella.wego.com/
- Napster webpage: http://napster.com/
- "Universal Description, Discovery and Integration: UDDI", Technical White paper, September 2000

Security
- "BlueBox : A Policy-Driven, Host-Based Intrusion Detection System", Suresh N. Chari, Pau-Chen Cheng. NDSS'02
- "White-Box Access Control Models," Paul Samaha, E.C., Hai Pham, Charles Yianmu, IEEE Computer, 18(8), 2005
Bibliography (contd.)


- Messaging
  - “Flexible Information Discovery in Decentralized Distributed Systems,” Cristina Schmidt and Manish Parashar, Accepted for publication at HPDC 12, 2003

Bibliography (contd.)

• Management

• Rule-based Systems
Bibliography (contd.)

- Miscellaneous