Autonomic Computing Tutorial, ICAC 2004

Autonomic Computing Tutorial, ICAC 2004

Research Issues

Manish Parashar
The Applied Software Systems Laboratory
Rutgers, The State University of New Jersey
http://automate.rutgers.edu

Salim Hariri
High Performance Distributed Computing Laboratory
The University of Arizona
http://www.ece.arizona.edu/~hpdc
ICAC 2004 Autonomic Computing Tutorial
May 16, 2004

Research Issues and Challenges

Scale, Complexity, Dynamism, Heterogeneity, Unreliability, Uncertainty

- Defining autonomic elements
  - Programming paradigms and development models/frameworks
    - Autonomic component/service definition and construction
    - Policies/constraints definition, representation, enforcement
  - Constructing autonomic systems/applications
    - Composition, coordination, interactions models and infrastructures
      - Dynamic (policy/rule-based) configuration, execution and optimization
      - Dynamic (opportunistic) interactions, coordination, negotiation
  - Execution and management
    - Runtime and middleware services
      - Discovery, Interaction/coordination, Security, Management, ...
    - Security, protection
    - Fault tolerance, reliability, availability, ...

Autonomic Computing Tutorial, ICAC 2004
Programming Models and Frameworks

- A programming framework consists of 3 components
  - A programming model that defines a set of abstractions that programmer uses to define the behavior of application elements and their interactions.
  - An underlying virtual machine that defines the execution context for the applications and embodies the assumptions made by the model about the capabilities, behaviors and qualities of services of the underlying environments.
  - An infrastructure that provides the services necessary for creating, managing and destroying the virtual machine and for realizing the virtualization assumed by the programming model.

Elements of a Programming Model

- Entity model
  - Defines the first-class citizens, for example, processes objects, and services.
- Naming and discovery model
  - Defines how those entities know of each other and how they find each other.
- Composition model
  - Describes how to integrate entities into an application.
    - A composition model consists of
      - Organization model describes where the interaction is from/to.
      - Communication model describes how the entities talk with each other.
      - Coordination model describes when they talk with each other.
- Failure handling
  - Provides failure detection, recovery or tolerance for entities and applications.
## Requirements for Programming Frameworks

<table>
<thead>
<tr>
<th>Entity</th>
<th>Heterogeneity</th>
<th>Dynamism</th>
<th>Unreliability</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Context-aware</td>
<td>Context-aware, adaptive</td>
<td>Context-aware, adaptive</td>
<td>Independent authorization and authentication, encapsulation</td>
</tr>
<tr>
<td>Naming and discovering</td>
<td>Implementation- and context-independent</td>
<td>Dynamic</td>
<td>Descriptive names including state information</td>
<td>Descriptive names including access privileges</td>
</tr>
<tr>
<td>Communication</td>
<td>Conforming to a uniform syntax and semantics</td>
<td>Dynamically creating, deleting, and modifying communication relationships</td>
<td>Manage failure while guaranteeing required semantics</td>
<td>Access control, message integrity, trust</td>
</tr>
<tr>
<td>Lifecycle management</td>
<td>Uniform mechanism to create/terminate entities, and maintain lifetime</td>
<td>Capable of creating and deleting entities at runtime</td>
<td>Safe creation and deletion of entities</td>
<td>Security requirements instantiated with entity creation</td>
</tr>
<tr>
<td>Failure handling</td>
<td>Abstract failure description</td>
<td>Failure sets need to be enlarged to include the failures of performance degrading, denial of service etc. caused by dynamism, unreliability and security</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Capabilities and Limitations of Current Programming Models and Frameworks

<table>
<thead>
<tr>
<th>Capabilities</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component frameworks (e.g., RPC, RMI, MPI, PVM)</td>
<td>Heterogeneity, limited dynamism</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed object frameworks (e.g., CORBA)</td>
<td>Heterogeneity, limited dynamism</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Component based frameworks (e.g., CCA)</td>
<td>Heterogeneous, limited dynamism</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Service models (e.g., Web service model, Grid service model)</td>
<td>Heterogeneous, dynamism</td>
</tr>
</tbody>
</table>

* Note that the environments do not change during the lifetime of the applications.
Component/Service-based Programming

- Component Model
- Composition Technology
- Component Integration Environment

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 provides 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, and a manager
  - 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, attributes, or fuzzy matching.
- Lifecycle service
  - During its lifetime, component should be available and consistent.
  - Manage termination/failure of components
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
  – Dynamic lock mechanisms

• 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

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
  - Adopting suitable execution model for composed application
  - Component communication, interaction and reconfiguration
    - dynamic, opportunistic, ephemeral, p2p/hierarchical/c-s, …
- 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.
    - DySCo, SWORD, Symphony
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 rule/policy/constraint definition and deployment
- Adaptive rule/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
Coordination Middleware

- Support coordination and communication between autonomic elements and agents
  - Enable dynamic and opportunistic choreography and interactions of components and services
  - Enable applications to be dynamically composed and configured to adapt to the environment and current requirements
  - Provide mechanisms for dynamically defining, configuring, routing and executing policies and rules
- Examples
  - World Wide Web: PageSpace, JavaSpace, TSpace, MARS, TuCSoN
  - Mobile Ad hoc Network: JEDI, Lime, TOTA
  - Peer-to-Peer Network: PeerSpace, PeerWare, Rudder

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
- **Immediate Cause**
  - Resource depletion faults
  - Logic faults
  - Physical faults
- **Effects**
  - Value Faults
  - Timing 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

Yoshihiro TOHMA, Tokyo Denki University

- 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
Messaging Services

- Messaging semantics
  - Address or content based
  - Deterministic, probabilistic, opportunistic
  - Point-to-point, rendezvous, publisher/subscriber, rpc, epidemic, …

- Messaging architectures
  - Server based, p2p, …

- Message format
  - XML, custom, …

- Existing Messaging Frameworks include
  - Meteor/AR, I3, ICENI, Xevents/Xmessages, Pawn, NaradaBrokering, Java Message Service, IBM MQSeries, …

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
    - Pro: overlay 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
    - Pro: can support complex queries
    - Cons: not scalable
  - **Data-lookup (CAN, Chord, Pastry, etc)**
    - Structured overlay, Internet-scale DHT
    - Pro: efficient lookup with guarantees
    - Cons: complex queries not supported
  - **Structured keyword search (Squid):**
    - Structured overlay, extend data-lookup protocols
    - Examples:
      - Distributed Inverted Indexes
      - Space Filling Curves

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, (Context-aware) Access Control,
    - Vulnerability/Intrusion/Attack 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
### 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
Autonomic Computing Research Challenges

- Computing research has evolved in many isolated, loosely coupled research fields as (security, fault tolerance, high performance, AI, network, agent systems, etc.)
- Design and Development of computing systems and applications that provide high performance, secure, fault-tolerance, and intelligent face challenging research problems
  - Programming Paradigms
  - Middleware/Software Systems

Autonomic Programming Paradigm Research Issues

- Autonomic components and applications – how do we express the autonomic properties such as self-healing, self-optimizing, self-protection, etc. and how to integrate that with new or existing components and applications
- Adaptive Compositions of Applications – how can we dynamically add, delete, and change the algorithm used to implement each component at runtime
- Automating Existing Applications/Tasks – how do you convert an existing application and/or a task to be autonomic