Autonomic Computing
Project AutoMate

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

AutoMate: Enabling Autonomic Applications
(http://automate.rutgers.edu)

• Objective:
  – 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

• Overview:
  – Definition of Autonomic Components:
    • definition of programming abstractions and supporting infrastructure that will
      enable the definition of autonomic components
    • autonomic components provide enhanced profiles or contracts that encapsulate
      their functional, operational, and control aspects
  – Dynamic Composition of Autonomic Applications:
    • mechanisms and supporting infrastructure to enable autonomic applications to be
      dynamically and opportunistically composed from autonomic components
    • compositions 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:
    • design, development, and deployment of key services on top of the Grid
      middleware infrastructure to support autonomic applications
    • a key requirement for autonomic behavior and dynamic compositions is the ability
      of the components, applications and resources (systems, services, storage, data) to
      interact as peers
AutoMate: Architecture

- Key components:
  - Autonomic programming framework (ICAC 2004)
  - Decentralized deductive engine and coordination middleware (Self* 2004, ICAC 2004)
  - Associative (content-based) Interactions (HPDC 2004)
  - P2P content-based discovery service (HPDC 2003, IEEE-IC 2004)
  - Dynamic access management engine (GCW 2003)

AutoMate System Stack

- Autonomic Applications (Autonomic Living, Ad hoc Control)
- Accord Autonomic Programming Framework
  - Coordinate Flows
  - Opportunistic Interactions
    - Discovery/Messaging (Content-DHT/Associative Rendezvous)
    - Content Network
  - Self Organizing Overlay
    - Ad Hoc Routing
    - Self Configuration
- Orbit Testbed
- Programming Model
- Meteor Middleware Stack
- Security: Ontology, Taxonomy, Authorization, Authentication, Trust

AutoMate: Architecture

AutoMate System Stack
AutoMate: Architecture

- AutoMate System Layer:
  - builds on the Grid middleware and OGSA and extends core Grid services to support autonomic behavior
  - provide specialized services such as peer-to-peer semantic messaging, events and notification
- AutoMate Component Layer:
  - addresses the definition, execution and runtime management of autonomic components
  - provides supporting services such as discovery, factory, lifecycle, context, etc
- AutoMate Application Layer:
  - builds on the component and system layers to support the autonomic composition and dynamic (opportunistic) interactions between components
- AutoMate Engines:
  - decentralized (peer-to-peer) networks of agents in the system
    - context-awareness engine composed of context agents and services and provides context information at different levels to trigger autonomic behaviors
    - deductive engine 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
    - trust and access control engine composed of access control agents and provides dynamic context-aware control to all interactions in the system
- AutoMate Portals:
  - provide users with secure, pervasive (and collaborative) access to the different entities
  - enable users to access resource, monitor, interact with, and steer components, compose and deploy applications, configure and deploy rules, etc

ACCORD: 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
Accord Programming Framework

• Autonomic component
  – Self-managed
  – Context-aware
• Dynamic composition
  – Runtime coordination/interaction of components

Autonomic Component (1)

• An autonomic component is described in three types of ports
  – Functional port defines a set of functions provided and used by the autonomic component
  – Control port defines the sensors/actuators for externally monitoring and controlling the autonomic component, and a set of constraints to control the access to the sensors and actuators
  – Operational port defines the interfaces to formulate, inject and manage rules, and encapsulate a set of rules used to manage the runtime behaviors of the component
Rule

- IF condition THEN then_actions ELSE else_actions

A logic combination of sensors, events, and functional interfaces
A sequence of sensors, actuators and functional interfaces

- Rules are categorized into
  - Behavior rules → manage the runtime behaviors of a component
  - Interaction rules → manage the interactions between components, between components and environments, and the coordination within an application
  - Security rules → control access to the functional interfaces, sensors/actuators and rule interfaces

Autonomic Component (2)

- Each autonomic component embeds a rule agent that is delegated to evaluate and execute rules in order to manage the execution of the component, and cooperates with other rule agents to fulfill application objectives
Dynamic Composition

• What is dynamic composition?
  – “Dynamic composition focuses on adapting running applications and changing their existing functionality either by adding new features or removing features”
• Dynamic composition includes
  – Dynamic plan generation
    • Generate plans at runtime based on dynamically defined composition objectives, their semantic descriptions, constraints, and available services and resources
    • E.g., Cardoso, SWORD, ICEINI, ACE
  – Dynamic plan execution
    • Integrate components together and execute them as specified in the composition plan
    • E.g., COMPOST, RUDDER

Dynamic Composition in Accord

• ACE (Accord Composition Engine) generates plans based on current context, goals, constraints and costs
  – It provides design-time customization and adaptation
• We will talk about dynamic plan execution in the following slides, in which “dynamic composition” specifically means “dynamic plan execution”
Our View of Dynamic Composition

- Replace a component
- Add a new component (interaction relationships changed)
- Change the interaction relationship
- Delete a component (interaction relationships changed)

Look At It Closer (1) Organization

- It defines how to organize autonomic components into an application though matching functional ports used by a component and provided by other components.
Look At It Closer (2) Communication

- It defines how the autonomic components talk with each other, via integrating basic communication paradigms (e.g., message sending/receiving) provided by underlying environment to rich semantic paradigms (e.g., RPC/RMI, shared-spaces) based on rules.

Look At It Closer (3) Coordination

- It defines the sequence that components interact with each other, describing the data dependencies and control dependencies between components:
  - Data dependencies can be handled by communication rules.
  - Control dependencies are described in the following 4 control structures:
    1. conditional branch
    2. loop
    3. sequence
    4. parallel execution
How does Accord support Dynamic Composition?

1. Decompose the workflow into interaction rules
2. Then inject these rules into corresponding autonomic components

Individual autonomic components establish interactions based on its interaction rules – local behavior
Add or delete components: insert or delete related interaction rules into or from corresponding autonomic components – local behavior
Modify existing interactions: modify correspond interaction rules – local behavior

The application is executed in a distributed and possible parallel manner

How does Accord support Self-Management?

One of the components involved in this rule is dynamically selected as the manager:

The manager decompose the rule into triggers, inject triggers into other involved autonomic components, collects trigger results, evaluate the rule and invoke actions if the condition is satisfied
Prototype Implementation based on CCA Caffeine

- Our prototype implementation introduces self-management features into CCA Caffeine
  - Enable CCA components to automatically change the runtime behaviors

Connection of ports is described in a script

CCA components implement the use/provide ports

An Illustrative Example: The Kaps Problem

Start the whole procedure

Automatically select the optimal solver based to solve the equation on the condition number R (received from Eigenvalue component) and the selection rules

Calculates the condition number R of the equation

Provide the Kaps equation to be solved
Prototype Implementation based on OGSA

- Dynamical replacement of Grid services
- Dynamical addition and deletion of Grid services
- Dynamic modification of interactions between Grid services

An Illustrative Example: The Oil Reservoir Application
Add A New Component

```plaintext
IF cellChangeMsg is received THEN
  assign cellChangeMsg to input
  invoke updateCell with input

IF isFighterWork()==true THEN
  send cellChangeMsg to Rothermel
```

Change The Interaction Relationship

```plaintext
Rule1:
IF isSystemCongested()==true
THEN setThreshold with 0.5
ELSE setThreshold with 0.3

Rule2:
IF isBalanced()==false
THEN sendLoadMsg to DSM
```
**Self-management: Runtime fine-tuned functional behaviors**

**One component involved**
- IF isSystemOverLoaded()==true
  THEN invoke graphAlgorithm
  ELSE invoke greedyBlockAlgorithm

**Multiple components involved**
- IF CRM.isThreshold()>50
  THEN invoke DSM.greedyBlockAlgorithm
- IF Trigger
  THEN invoke greedyBlockAlgorithm

**ACCORD: Autonomic Composition Engine**
- Dynamically synthesize a service/composition plan at runtime based on dynamically defined goals, constraints and context
  - annotate services/components with semantic information describing its functionality and interfaces
  - use relational algebra to choreograph ad hoc interactions
  - use constraints to define and evaluate composition/service plans
Algorithm

[Initialization]
1. Each service description document is parsed for metadata
2. Semantic information is used to enhance service description
3. Composition request is made by the composer, consists of
   1. Composition Objective
   2. Semantic metadata
   3. Semantic threshold value i.e. the degree of correlation expected
   4. Constraints
   5. Start and target operations/services

[Selection]
1. Select appropriate services based on semantic matching
2. Executing constraints to refine services selection and composition

Algorithm

[Plan Generation]
1. All possible ad-hoc interactions are formulated
2. Service Graph is constructed, where
   1. Each operation acts as vertices of abstract graph
   2. The output argument types of operation are matched with the input parameters. If matching is correct, interaction link is created between operations
3. Constraints are executed to enable or disable inconsistent interactions
4. Initial and final operations are selected/specified based on semantic information of the composition and composition graph is created
5. Composition plan(s) is(are) generated or status is returned
   1. Path in the interaction graph from source to destination operation corresponds to sequence of required message invocations
   2. Operations lying on the path correspond to participating services
   3. Scenarios where multiple composition plans can exist, the cost factor is evaluated for each path and least cost path is selected
**ACCORD: ACE - Prototype operation**

1. Composition request
   - Objective
   - Constraints
   - Semantic metadata

2. Connect and select services
   - based on constraints
   - based on keywords
   - based on input arguments

3. Create interaction links
   - using relational join based on semantic annotations

4. Synthesize composition plans as paths in the ad hoc service graph

5. Rank and return composition plans

**ACE Architecture**

- ACE Translator
- Graph Generator
- Constraint Analyzer
- Plan Generator

**ACCORD Composition Engine**

- SourceOperation: SourceService: SourceMessageName
- TargetOperation: TargetService: TargetMessageName
- CostOfLink: Valid: Level

- ServiceName: Objective: Keywords

- ServiceName: Operation: ParamOrder
- InputMessage: OutputMessage

- MessageName: ArgumentTypes: ArgumentName
**ACCORD: 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

---

**Enable Self-Managing Applications**

- Self-configuration - dynamic discovery, configuration and composition of components at run time
- Self-optimization - dynamic switching of workflows, components, and component interaction patterns, balancing of workload and resource utilization
- Self-healing - restarting or replacing failed components
- Self-protection - reactions defined to defend the system integrity from undesired operations of malicious agents
Rudder: A Rule-based Multi-agent Infrastructure for Supporting Autonomic Grid Applications

• Motivation
  – Self-managing context-aware autonomic applications change dynamically in heterogeneous pervasive Grid environments
  – These applications interact with environments and with each other to monitor, adapt and optimize their execution.
  – Supporting these applications requires suitable and effective coordination middleware services.

• Objective
  – Enable dynamic composition and coordination of components to manage changing application requirements and system context
  – Providing mechanisms for dynamically defining, configuring, deploying rules, and rule execution management

Rudder: System Architecture

• Agent framework
  – Component Agents, System Agents, Composition Agents
  – Control and configure components
  – Define composition rules: workflow-selection, component selection and task ordering rules

• A decentralized reactive tuple-space
  – Programmable behavior
  – Enable agent communication and coordination
  – Rule deploy and execution
**System Prototype**

- **JXTA Service based integration**
  - Agents implemented as JXTA application layer
  - Tuple space implemented as a JXTA module
  - Squid routing engine as a JXTA service enables complex keyword searches, guaranteeing query messages delivery
- **Programming language**
  - Tuples as XML document
  - Kernel Java

**Enabling Autonomic Oil Reservoir Optimization**

- **Goal**: maximize revenue from an oil field by optimizing wells’ placement and configurations
- **Three composition agents** for EM, Optimizer and IPARS
- **Autonomic optimization**
  - Component agents monitor and manage component executions
  - Composition agents proactively search and select available components and resources to satisfy current application objectives
  - System agents monitor the runtime utilization of the resource and dynamically balance workload
SQUID: A Decentralized Discovery Service

• Overview/Motivation:
  – Efficient information discovery in the absence of global knowledge of naming conventions is a fundamental problem in large, decentralized, distributed resource sharing environments such as the Grid.
  – Heterogeneous nature and large volume of data and resources, their dynamism and the dynamism of the Grid make the information discovery a challenging problem.

• Key features
  – P2P system that supports complex queries containing partial keywords, wildcards, and range queries.
  – 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 system can be used as a complement for current resource discovery mechanisms in Computational Grids (to enhance them with range queries).

SQUID: Design

• Overall architecture is a distributed hash table (DHT), similar to typical data lookup systems (e.g. Chord, CAN).
• Key innovation is a locality preserving, dimension reducing indexing scheme that effectively maps the multidimensional information space to physical peers.
  – data elements described using a sequence of keywords (common words in the case of P2P storage systems, or values of globally defined attributes - such as memory and CPU frequency - for resource discovery in computational grids)
    • 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
  – use Space Filling Curves to map documents that are local multi-dimensional index space to indices that are local in the 1-dimensional index space.
**SQUID: Operation**

(a) Network:

(b) Decoupled Interactions:

(c) Overlay network (e.g., chord):

(d) Application:

**Meteor Associative Rendezvous: Content-based Decoupled Interactions**

- **Associate Rendezvous Overlay**:
  - post(1)
  - retrieve(SEN_B, data)

- **Metadata Overlay**:
  - Overlay network (e.g., chord)

- **Application**:
  - Meteor Stack:
    - Content-based Routing (Squid)
    - P2P substrate

A schematic overview of the Meteor Stack:

- A PDA_A
- A SEN_B

Autonomic Computing Tutorial, ICAC 2004
Associative Rendezvous

- Content-based decoupled rendezvous interactions
  - publishers use semantic headers to publish information
  - information is maintained in the p2p messaging substrate
    - leases, TTL, garbage collection, replication, ...
  - subscribers can pull information using semantic selectors
    - support wildcards, ranges, partial keywords
    - matching messaging peers based on propositional logic
    - provides match guarantees
  - subscribers can register filtering/notification requests
  - programmable reactive behaviors at rendezvous nodes
  - supports multiple interaction semantics
    - 1-many, many-1, many-many, 1-1, anycast, etc.

Associative Rendezvous: An Illustrative Example
Cascading Local Behaviors (CLB)

- Pervasive applications as emergent opportunistic flows
  - Local behavior of pervasive element defined as context and content-based state transitions and interest/data messages
    - Content/context messages trigger local actions and/or generation of data/interest messages
  - Cascading of local content/context-based transitions result in opportunistic application flows

An Illustrative Example

Temperature Sensor
if temp > 80 then publish temp

Thermostat
If temp > 85
Then temp_control = on;
publish temp_control

Window Actuator
If temp_control == on
then close windows

Meteor AR
post(temp = 91, store)

Thermostat
notify(thermostat)
post(temp_control, store)

Turn temp_control on
post(temp_control, store)

Close window

Meteor AR
retrieve(temp_control)

Close window
Meteor Programming Framework and Content-based Middleware

- Self-organizing overlay
  - Rendezvous points: Chord, a ring overlay
- Content-based routing
  - Squid
- AR Messaging substrate
  - Profile manager
  - Matching engine
- CLB-based opportunistic flows
  - Local state machine
  - Message dispatcher

Pawn: A P2P Messaging Substrate

- Objective
  - Engineer a peer-to-peer messaging substrate that extends existing solutions to enable high-level interactions for scientific applications.
- Architecture
  - Peers, Messages, Services, Interactions
- Key Features
  - Stateful messages
  - Guaranteed messaging semantics
  - Publish/subscribe mechanisms across peer-to-peer domains
  - High-level messaging semantics
    - Sync/Async Messaging
    - PUSH (dynamic injection)
    - PawnRPC
- Built on Project JXTA
  - Pipes
  - Resolver
SESAME: Context Aware Access Management (CCS 03)

- **Objective:**
  - support dynamic, seamless and secure interactions between the participating entities (i.e. components, services, application, data, instruments, resources and users)
  - Autonomic Computing – Self Protecting (Context aware, Dynamic)

- **Issues:**
  - access rights in highly dynamic and heterogeneous Grid environments depends on the entity's privileges, capabilities, context and state
    - e.g. 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. location, time, secure connection) and the state of the resource or component

- **Approach**
  - extend Role Based Access Control (RBAS) to make access control decision based on dynamic context information
  - dynamically adjust Role Assignments and Permission Assignments based on context

SESAME: Operation

- Dynamically adjusts the user-role and role-permission relationships based on context information

- each component is assigned a role subset (by the authority service) from the entire role set on authentication
- each component maintains permission subsets for each role that will access the component
- during an interaction, state machines are maintained by the delegated 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
- state machines define the currently active role permissions
- access agent navigates the role/permission subsets to react to changes in the context
SESAME: Illustrative Example

- 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.

DAIS: Detecting DDoS Attacks by Information Sharing

- Distributed framework of detection nodes where heterogeneous systems can plug in and cooperate to achieve a better overall detection.
- Each local detection node detects traffic anomalies using profiles of normal traffic constructed using stream sampling algorithms.
- System uses the overlay network to share the detection information using gossip protocol based on epidemic algorithm across the Internet.
DAIS: Architecture for an Individual Detection Node

Individual Detection Node

- Respond to attack
- Detection decision

Local detection mechanism
- Local traffic measurement

Cooperative detection engine
- Message dissemination using gossip
- Neighbor nodes

The Team

- TASSL Rutgers University
  - Autonomic Computing Research Group
    - Viraj Bhat
    - Nanyan Jiang
    - Hua Liu (Maria)
    - Zhen Li (Jenny)
    - Vincent Matossian
    - Cristina Schmidt
    - Guangsen Zhang
  - Autonomic Applications Research Group
    - Sumir Chandra
    - Xiaolin Li
    - Li Zhang

- CS Collaborators
  - HPDC, University of Arizona
    - Salim Hariri
  - Biomedical Informatics, The Ohio State University
    - Tahsin Kurc, Joel Saltz
  - CS, University of Maryland
    - Alan Sussman, Christian Hansen

- Applications Collaborators
  - CSM, University of Texas at Austin
    - Malgorzata Peszynska, Mary Wheeler
  - IG, University of Texas at Austin
    - Minai Sen, Paul Stoffa
  - ASCI/CACR, Caltech
    - Michael Alvisi, Julian Cummings, Dan Meiron
  - CRL, Sandia National Laboratory, Livermore
    - Jaideep Ray, Johan Steensland