Project AutoMate
Enabling Autonomic Applications

Manish Parashar, The AutoMate Group
The Applied Software Systems Laboratory
Rutgers, The State University of New Jersey
http://automate.rutgers.edu
Ack: NSF (CAREER, KDI, ITR, NGS), DoE (ASCI, CIT)

CAIP Research Review
November 12, 2003
Outline

• Autonomic computing – motivations and overview
• AutoMate: A framework of enabling autonomic applications
  – ACCORD: Autonomic component framework
  – RUDDER: Decentralized deductive engine
  – SESAME: Context sensitive dynamics access management
  – Pawn: Peer-to-Peer messaging infrastructure
  – SQUID: Decentralized discovery service
• Application Scenarios
  – Autonomic interactions oil reservoir optimization
  – Autonomic runtime management
  – Autonomic design
  – …

Conclusions
Current Trends:
Smaller/Cheaper/Faster/Powerful/Connected ….

• Explosive growth in computation, communication, information and integration technologies
  – *computing is ubiquitous, pervasive – communication is/will be*

• Pervasive “anytime-anywhere” access environments
  – *ubiquitous access to information via PCs, PDAs, Cells, smart appliances, etc. (billions of devices, millions of users)*
  – *peers capable of producing/consuming/processing information at different levels and granularities*
  – *embedded devices in clothes, phones, cars, mile-markers, traffic lights, lamp posts, refrigerators, medical instruments …*

• “On demand” computational/storage resources, services

• The emerging national/global Cyberinfrastructure
However …

• Developing application to utilize and exploit the emerging Cyberinfrastructure remains a significant challenge
  – Unprecedented scales, complexity, heterogeneity, dynamism and unpredictability, lack of guarantees
  – The problem: a level of complexity, heterogeneity, and dynamism for which our programming environments and infrastructure are becoming unmanageable, brittle and insecure
    • System size, heterogeneity, dynamics, reliability, availability, usability
  – Requires fundamental changes in how applications are formulated, composed and managed
    • Breaks current paradigms based on passive components and static compositions
    • autonomic components and their dynamic composition, opportunistic interactions, virtual runtime, …
  – Resonance - heterogeneity and dynamics must match and exploit the heterogeneous and dynamic nature of the cyberinfrastructure
Autonomic Computing?

- Nature has evolved to cope with scale, complexity, heterogeneity, dynamism and unpredictability, lack of guarantees
  - *self configuring, self adapting, self optimizing, self healing, self protecting, highly decentralized, heterogeneous architectures that work !!!*
  - *e.g. the autonomic nervous system*
    - tells you heart how fast to beat, checks your blood's sugar and oxygen levels, and controls your pupils so the right amount of light reaches your eyes as you read these words, monitors your temperature and adjusts your blood flow and skin functions to keep it at 98.6°F
    - coordinates - an increase in heart rate without a corresponding adjustment to breathing and blood pressure would be disastrous
    - is autonomic - you can make a mad dash for the train without having to calculate how much faster to breathe and pump your heart, or if you'll need that little dose of adrenaline to make it through the doors before they close
  - can these strategies inspire solutions?
  - of course, there is a cost
    - *lack of controllability, precision, guarantees, comprehensibility, …*
  - A.I. ? – duplication of human thought is not the ultimate goal
AutoMate: Enabling Autonomic Applications (http://automate.rutgers.edu)

- **Objective:**
  - To enable the development of autonomic applications that are context aware and are capable of self-configuring, self-composing, self-optimizing and self-adapting.
  - address uncertainty, unreliability, dynamism, incomplete knowledge

- **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 and Coordination of Autonomic Applications**
    - models, mechanisms and supporting infrastructure to enable autonomic applications to be dynamically and opportunistically composed from autonomic components
    - composition/coordination based on (dynamically defined) declarative policies, goals and constraints, and 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
    - enable components, applications and resources (systems, services, storage, data) to interact as peers
AutoMate: Design Principles

• Separation of policy from mechanism distilling out the aspects 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 ad-hoc 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.
AutoMate: Architecture

Key components:
- Accord: Autonomic application framework
- Rudder: Decentralized deductive engine
- Squid: P2P discovery service
- SESAME: Dynamic access control engine
- Pawn: P2P messaging substrate
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
  - using these portals users can access resource, monitor, interact with, and steer components, compose and deploy applications, configure and deploy rules, etc.
• 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 & coordination aspects
    • describes the adaptability of the component and defines sensors/actuators and policies for management, interaction and control.
ACCORD: Autonomic Components

- 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
ACCORD: Autonomic Component – Prototype implementation (EuroPar 03)
ACCORD: Autonomic Composition/Coordination 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 (extend WSDL with semantic metadata)
  - use relational algebra to choreograph ad hoc interactions at runtime
  - use constraints to define and evaluate composition/service plans
  - provide mechanism to evaluate and rank plans based on user defined cost models
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

- **Translator**
  - Parse WSDL description
  - Update relevant tables
  - Add semantic metadata
- **Graph Generator**
  - Define interactions links
  - Desirability factor is associated with link
- **Constraint Analyzer**
  - Valid links are selected
  - Represented as simple SQL queries
- **Plan Generator**
  - Appropriate plans are generated
  - Plans are ranked based on cost factor
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
RUDDER is a decentralized deductive engine composed of distributed specialized agents (component rule agents, composition/coordination agents, context agents and system agents) that exist at different levels of the system, and represents their collective behavior.

Objectives

- Providing mechanisms for dynamically defining, configuring, deploying rules, and rule conflicts management
- Runtime management services, supporting autonomic composition, adaptation, optimization and execution
RUDDER Architecture

Middleware Services
RUDDER: Agent Architecture

- Goal-directed focus: focus on the objective and choose the method to achieve it
- Context sensitivity: make decisions about what to try and retry based on present conditions

- Dynamically added, deleted and modified
- Defined using XML rule schema and consists of tags:<RULE identifier>,<priority>,<ON events>, <THEN plan-identifier>

**Agent**

- Message Queues
- Dispatcher

**State Machine**

- User Interface Aspect
- Data Aspect
- Sys Aspect

**Strategy**

- Beliefs
- Goals
- Rule Set (Conflict solver)

**Events**

- InternalEvent
- AdaptiveEvent
- ReactiveEvent

**Auto Process**

- Event driven reactive, adaptive behavior and goal directed proactive behavior
BDI Agent Model

- An agent has beliefs about the world and desires to satisfy, driving it to form intentions to act
  - Beliefs: about the environment and other agents
  - Desire or goals to achieve
  - Intention or plans to act upon to achieve its desires

Human → Belief, Desire, Intentions Agent

Beliefs - perceived understanding of the world

Goals or desires

Accumulated experience and behaviours

Execution Engine

Pre-compiled plans

Beliefs - database of perceived world knowledge

Goals or desires

Intentions - currently executing plans
SESAME: Context Aware Access Management

**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
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.
SQUID: A Decentralized Information Discovery/Coordination

• Overview/Motivation:
  – Efficient information discovery and coordinations in the absence of global knowledge of naming conventions is a fundamental problem in large, decentralized, distributed resource sharing environments such as the Grid
    • 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.
  – 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.

• 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
    - load-balancing at join and runtime
  - existing systems, this is done using consistent hashing to uniformly map data element identifiers to indices
    - data elements are randomly distributed across peers without any notion of locality
SQUID: Operation
Pawn: A P2P Messaging Substrate

- A peer-to-peer messaging substrate that extends existing solutions to enable high-level interactions for scientific applications.
  - semantic “associative” interactions

- 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

- Builds on Project JXTA
Pawn: A P2P Messaging Substrate

- A peer-to-peer messaging substrate that extends existing solutions to enable high-level interactions for scientific applications.
  - semantic “associative” interactions

- 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

- Builds on Project JXTA
Autonomic Grid Computing: Seamless Interaction for Global Scientific Investigation

- **Secure/Seamless Interactions**
  - Client – Client -> Collaboration,
  - Client – Application -> Monitoring/Interaction/Steering,
  - Application – Application -> Application components, services, …
  - Application – Data -> Dynamic Data Injection, Sensor networks, I/O, Checkpoint/Restart, …
  - Client – Data -> Visualization, mining, …
  - Application – Resource -> Staging, dynamic resource allocation, execution migration

- **Autonomic Infrastructure**
  - Autonomons components and dynamic composition, configuration, optimizations
    - Self defining, self self-defining, self-configuring, self-optimizing, self-protecting, self-healing, context aware and anticipatory
    - “Deductive” computational middleware
      - Dynamic, rule-based, component configuration, optimization, and composition, constraint-based dynamic interactions
  - P2P Grid infrastructure
    - Security, dynamic access control, discovery, messaging, resource management, QoS, collaboration, interaction, …
Detect and track changes in data during production
Invert data for reservoir properties
Assimilate data & reservoir properties into the evolving reservoir model
Use simulation and optimization to guide future production

- Production simulation via reservoir modeling
- Monitor production by acquiring time lapse observation of seismic data
- Revise knowledge of reservoir model via imaging and inversion of seismic data
- Modify production strategy using an optimization criteria
Autonomic Oil Well Placement (UT-CSM, UT-IG)

- Optimization algorithm: use VFSA (Very Fast Simulated Annealing)
  - requires function evaluation only, no gradients
- IPARS delivers
  - fast-forward model (guess->objective function value)
  - post-processing
- Formulate a parameter space
  - well position and pressure \((y,z,P)\)
- Formulate an objective function:
  - maximize economic value \(\text{Eval}(y,z,P)(T)\)
- Normalize the objective function \(\text{NEval}(y,z,P)\) so that:

\[
\min \text{NEval}(y,z,P) \Leftrightarrow \max \text{Eval}(y,z,P)
\]
Components of the AORO Application

- **IPARS**: Integrated Parallel Accurate Reservoir Simulator
  - Parallel reservoir simulation framework
- **IPARS Factory**
  - Configures instances of IPARS simulations
  - Deploys them on resources on the Grid
  - Manages their execution
- **VFSA**: Very Fast Simulated Annealing
  - Optimizes the placement of wells and the inputs (pressure, temperature) to IPARS simulations.
- **Economic Modeling Service**
  - Uses IPARS simulations outputs and current market parameters (oil prices, costs, etc.) to compute estimated revenues for a particular reservoir configuration.
- **DISCOVER Computational Collaboratory**
  - Interaction & Collaboration
  - Distributed Interactive Object Substrate (DIOS)
  - Collaborative Portals

2. IPARS Factory discovers and initializes VFSA Optimization Service.

3. Client can configure IPARS params.

4. IPARS connects to VFSA Optimization Services and presents revenue.

5. Scientists/Engineers collaboratively interact with IPARS.

6. VFSA Optimization Service generates new well placement.

7. One optimal well placement is determined, IPARS Factory launches IPARS run.

8. IPARS Factory gets initial guess from VFSA Optimization Service launches IPARS instance on resource of choice.

Current oil price, market state, etc.
Autonomic Oil Well Placement

Permeability

Pressure contours
3 wells, 2D profile

Contours of $\text{NEval}(y,z,500)(10)$

Requires $NY\times NZ$ (450) evaluations. Minimum appears here.

VFSA solution: “walk”: found after 20 (81) evaluations
Sample Results

Guess Y 264.57623  Guess Z 170.17809

Y/Z positions for Y&Z [0,0,15]

First guess on injection well position

(Final) iterative guesses optimizing revenue

Fixed Injection well

Production wells

Minimizing cost

Accepted Guesses
Computational Modeling of Physical Phenomenon

- Realistic, physically accurate computational modeling
  - Large computation requirements
    - e.g. simulation of the core-collapse of supernovae in 3D with reasonable resolution ($500^3$) would require $\sim 10-20$ teraflops for 1.5 months (i.e. $\sim$100 Million CPUs!) and about 200 terabytes of storage
    - e.g. turbulent flow simulations using active flow control in aerospace and biomedical engineering requires $5000 \times 1000 \times 500 = 2.5 \times 10^9$ points and approximately $10^7$ time steps, i.e. with 1GFlop processors requires a runtime of $\sim 7 \times 10^6$ CPU hours, or about one month on 10,000 CPUs! (with perfect speedup). Also with 700B/pt the memory requirement is $\sim 1.75$TB of run time memory and $\sim 800$TB of storage.
  - Dynamically adaptive behaviors
  - Complex couplings
    - multi-physics, multi-model, multi-resolution, …
  - Complex interactions
    - application – application, application – resource, application – data, application – user, …
  - Software/systems engineering/programmability
    - volume and complexity of code, community of developers, …
      - scores of models, hundreds of components, millions of lines of code, …
A Selection of SAMR Applications

Blast wave in the presence of a uniform magnetic field – 3 levels of refinement. (Zeus + GrACE + Cactus, P. Li, NCSA, UCSD)

Mixture of H₂ and Air in stoichiometric proportions with a non-uniform temperature field (GrACE + CCA, Jaideep Ray, SNL, Livermore)

Multi-block grid structure and oil concentrations contours
IPARS, M. Peszynska, UT Austin)

Richtmyer-Meshkov - detonation in a deforming tube - 3 levels. Z=0 plane visualized on the right
(VTF + GrACE, R. Samtaney, CIT)
Autonomic Computational Science and Engineering
Application Runtime Management in V-Grid

Loop for each level of Grid/Application hierarchy

**V-Grid Monitoring**
( Self-observation, Context-awareness )
- System states (CPU, Memory, Bandwidth, Availability etc.)
- Application states
  - (Computation/Communication Ratio, Nature of Applications, Application Dynamics)

**V-Grid Deduction**
( Self-adaptation, Self-optimization, Self-healing )
- Identify and characterize natural regions
- Define objective functions and management strategy
- Define VCU's

**V-Grid Execution**
Partition, Map and Tune
ARMaDA: Application-sensitive Adaptations

- PAC tuple, 5-component metric
- Octant approach: app. runtime state
- GrACE (ISP), Vampire (pBD-ISP, GMISP+SP) partitioners
- ARMaDA framework
  - Computation/communication
  - Application dynamics
  - Nature of adaptation
- RM3D, 64 procs on “Blue Horizon”
  - 100 steps, base grid 128*32*32
  - 3 levels, RF = 2, regrid 4 steps

**ARMaDA evaluation for VectorWave-2D application on 32 processors on “Frea”**

<table>
<thead>
<tr>
<th>Partitioner</th>
<th>Execution time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFC</td>
<td>637.478</td>
</tr>
<tr>
<td>G-MISP+SP</td>
<td>611.749</td>
</tr>
<tr>
<td>pBD-ISP</td>
<td>592.05</td>
</tr>
<tr>
<td>ARMaDA with SFC start</td>
<td>470.531</td>
</tr>
</tbody>
</table>

\[ P_t = f(A_t, C_t) \]
ARMaDA: System-sensitive Adaptations

- System characteristics using NWS
- RM3D compressible turbulence application
  - 128x64x64 base (coarse) grid
  - 3 levels, factor 2 refinement
- System/Environment
  - University of Texas at Austin (32 nodes), Rutgers (16 nodes)

<table>
<thead>
<tr>
<th>Procs</th>
<th>Dynamic Sensing (s)</th>
<th>Static Sensing (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>423.7</td>
<td>805.5</td>
</tr>
<tr>
<td>4</td>
<td>292</td>
<td>450</td>
</tr>
<tr>
<td>6</td>
<td>272</td>
<td>424</td>
</tr>
<tr>
<td>8</td>
<td>225</td>
<td>430</td>
</tr>
</tbody>
</table>
Autonomic Forest Fire Simulation

Predicts fire spread (the speed, direction and intensity of forest fire front) as the fire propagates, based on both dynamic and static environmental and vegetation conditions.
Autonomic Design of Nanomaterials

**INPUT: COMPOSITION/PROCESSING**

**Atomistics:** Simulate non-equilibrium solidification process, crystallization, diffusion and growth

- Interface Kinetics
- Thermodynamics

**Composition**
- Atomistic Structure
- Force Fields

**Nanoscale:** Model the evolution and interaction of topologically complex nanosize metastable structures and their effective behavior

- Effective Particle Behavior
- Hardening Mechanisms

**Microscale:** Model the collective behavior of assemblies of nanostructured particles

- Strength and Ductility of Nanostuctured Particles

**Optimization:** Optimize metastable nanocomposition based on atomistic, nanoscale and macroscale properties.

**Computational Infrastructure:** To develop autonomic computational infrastructures and runtime management techniques for scalable parallel/distributing computing, automated interaction and data exchange between scales, real time sensing and computational response, collaborative monitoring and steering.

**OUTPUT: OPTIMIZED METASTABLE NANOCOMPOSITES**
Conclusion

• Autonomic applications are necessary to address scale/complexity/heterogeneity/dynamism/reliability challenges
• AutoMate addresses key issues to enable the development of autonomic Grid applications
  – ACCORD: Autonomic application framework
  – RUDDER: Decentralized deductive engine
  – SESAME: Dynamic access control engine
  – Pawn: P2P messaging substrate
  – SQUID: P2P discovery service
• Application scenarios
  – Autonomic optimization of oil reservoirs
  – Autonomic runtime management
• More Information, publications, software, conference
  – http://automate.rutgers.edu
  – automate@caip.rutgers.edu / parashar@caip.rutgers.edu
  – http://www.autonomic-conference.org
The Team

• TASSL Rutgers University
  – Autonomic Computing Research Group
    • Viraj Bhat
    • Manish Agarwal
    • Nanyan Jiang
    • Hua Liu (Maria)
    • Zhen Li (Jenny)
    • Manish Mahajan
    • Vincent Matossian
    • Cristina Schmidt
    • Guangsen Zhang
  – Autonomic Applications Research Group
    • Sumir Chandra
    • Xiaolin Li
    • Dave Roberts
    • Taher Saif
    • 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
    • Hector Klie, Mary Wheeler
  – IG, University of Texas at Austin
    • Mrinal Sen, Paul Stoffa
  – ASCI/CACR, Caltech
    • Michael Aivazis, Julian Cummings, Dan Meiron
  – CRL, Sandia National Laboratory, Livermore
    • Jaideep Ray, Johan Steensland