Organic Computing

Dr. rer. nat. Christophe Bobda
Prof. Dr. Rolf Wanka
Department of Computer Science 12
Hardware-Software-Co-Design

The Landscape

Existing Systems

- Academic Efforts
  - University Research
  - Work contributing to the Autonomic computing systems beyond
    - IBM's laboratories.
  - Few Research Projects include
    - OceanStore - UC Berkeley Computer Science Division
    - Kinesthetics eXtreme (KX) - Columbia University
    - Anthill - Department of Computer Science University of Bologna, Italy
    - Software Rejuvenation . Duke University
    - The Horus Project - Cornell University
    - eBiquity- University of Maryland Baltimore County
    - Recovery Oriented Computing - UC Berkeley / Stanford
    - Autonomia - University of Arizona

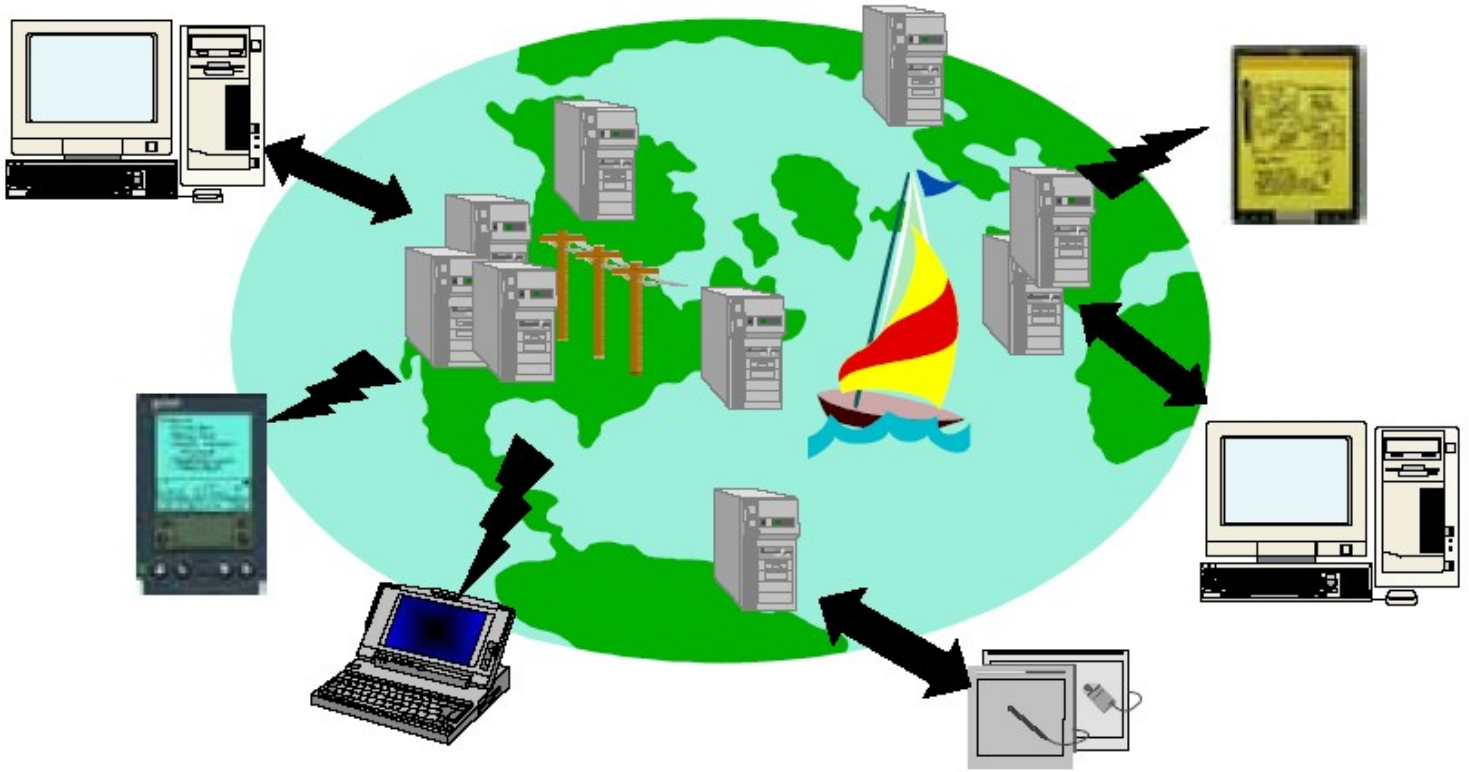
Existing Systems

- Industry Efforts
  - IBM committed focus to working within its own global labs and researchers.
    - Gryphon: Pub/Sub
    - Smart-Self Managing and Resource Tuning DB2
    - Sabio
    - Storage Tank
    - Océano
    - Smart Grid
  - Microsoft Research
    - AutoAdmin

OceanStore (UC Berkeley)

- Definition

  A utility infrastructure designed to span the globe and provide continuous access to persistent information.



OceanStore: Goal

- Computing everywhere:
  - Desktop, Laptop, Palmtop
  - Cars, Cellphones
  - Shoes? Clothing? Walls?
- Connectivity everywhere:
  - Rapid growth of bandwidth in the interior of the net
  - Broadband to the home and office
  - Wireless technologies such as CMDA, Satellite, laser

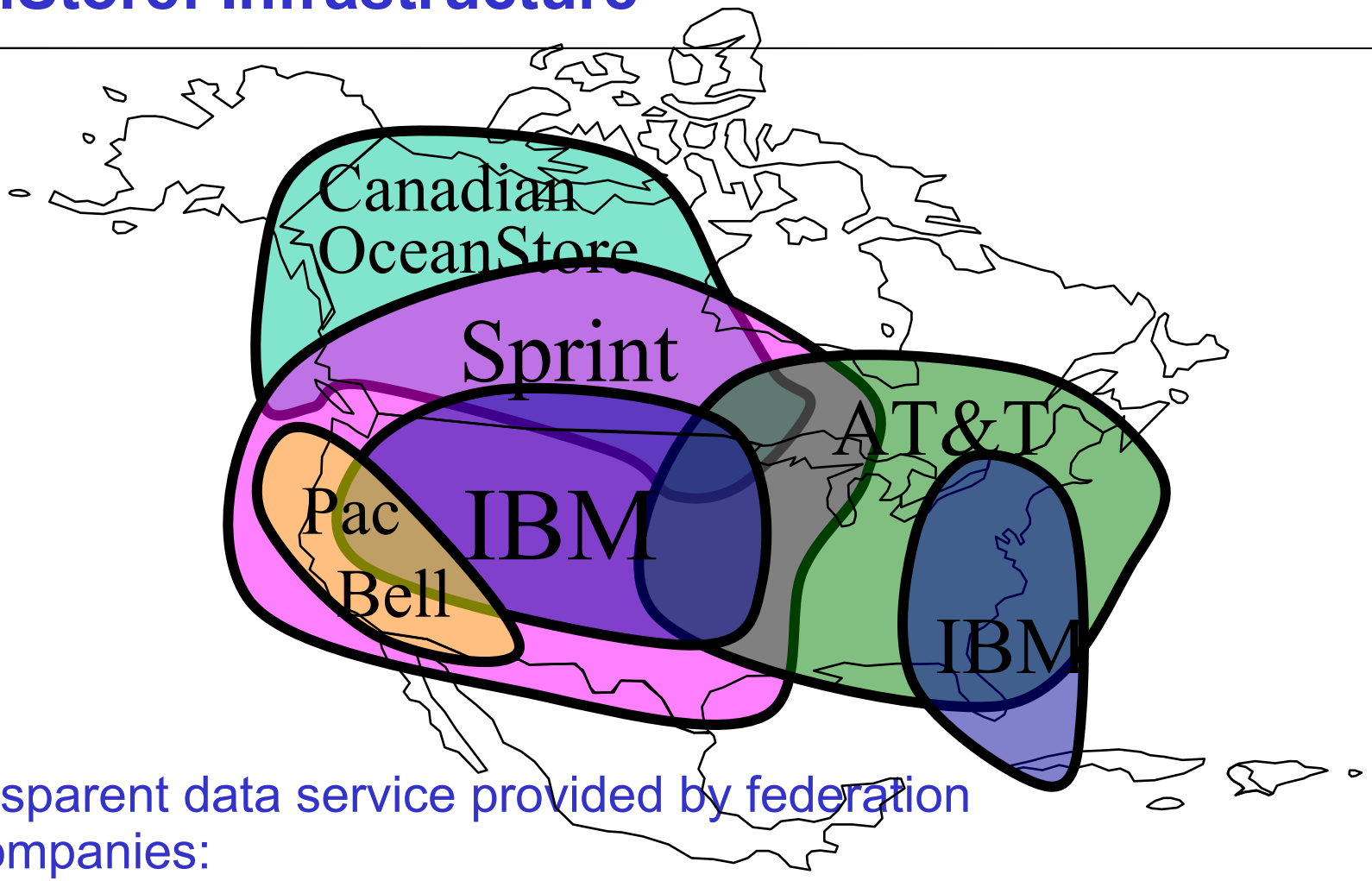
OceanStore: Goal

- Where is persistent information stored?
  - *Want: Geographic independence for availability, durability, and freedom to adapt to circumstances*
- How is it protected?
  - *Want: Encryption for privacy, signatures for authenticity, and Byzantine commitment for integrity*
- Can we make it indestructible?
  - *Want: Redundancy with continuous repair and redistribution for long-term durability*
- Is it hard to manage?
  - *Want: automatic optimization, diagnosis and repair*
- Who owns the aggregate resouces?
  - *Want: Utility Infrastructure!*

OceanStore: Goal

- Can't possibly manage billions of servers by hand!
- System should automatically
  - Adapt to failure
  - Repair itself
  - Incorporate new elements
- Can we guarantee data is available for 1000 years?
  - New servers added from time to time
  - Old servers removed from time to time
  - Everything just works
- Many components with geographic separation
  - System not disabled by natural disasters
  - Can adapt to changes in demand and regional outages
  - *Gain in stability through statistics*

OceanStore: Infrastructure



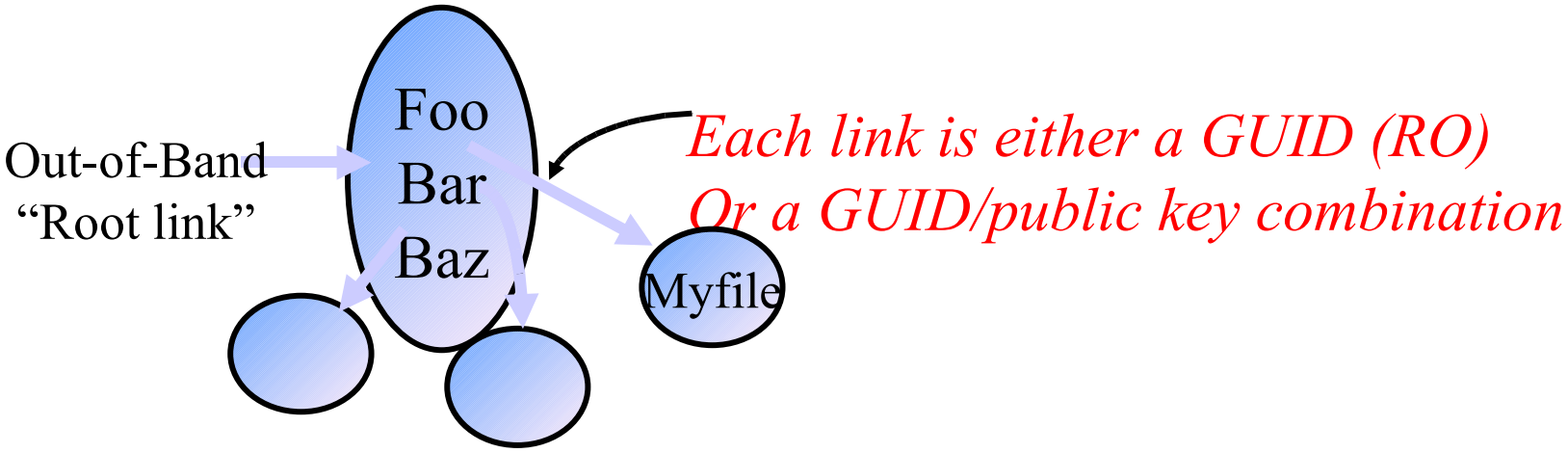
- Transparent data service provided by federation of companies:
  - Monthly fee paid to one service provider
  - Companies buy and sell capacity from each other

System architecture

- Naming and Access Control
  - GUID (Naming)
  - Reader and Writer Restriction
- Data Location and Routing
  - Fast Probabilistic Routing Algorithm (Self - Optimizing)
  - Slower, reliable hierarchical routing method (Plaxton Scheme)
    - Self healing, Self optimizing and Self Managing
- Update Model
  - Managed by a series of replicas
    - Master, Primary and Secondary tier of replicas.
- Durable Storage
  - Active Data in floating replicas
  - Archival Data in Erasure Coded fragments

OceanStore: Security

- Unique, location independent identifiers
  - Every *version* of every unique entity has a permanent, *Globally Unique ID (GUID)*
  - All OceanStore operations operate on GUIDs
- Naming hierarchy
  - Users map from names to GUIDs via hierarchy of OceanStore objects
  - Requires set of “root keys” to be acquired by user



OceanStore: Routing and data localization

- Requirements
  - Find data quickly, wherever it might reside
    - Locate nearby data without global communication
    - Permit rapid data migration
  - Insensitive to faults and denial of service attacks
    - Provide multiple routes to each piece of data
    - Route around bad servers and ignore bad data
  - Repairable infrastructure
    - Easy to reconstruct routing and location information
- Technique: Combined Routing and Data Location
  - Packets are addressed to GUIDs, not locations
  - Infrastructure gets the packets to their destinations and verifies that servers are behaving

OceanStore: Routing and data localization

- Fast, probabilistic search for “routing cache”
  - Built from *attenuated* bloom filters
  - Approximation to gradient search
  - *Not going to say more about this today*
- Redundant *Plaxton Mesh* used for underlying routing infrastructure
  - Randomized data structure with locality properties
  - Redundant, insensitive to faults, and repairable
  - Amenable to continuous adaptation to adjust for:
    - Changing network behavior
    - Faulty servers
    - Denial of service attacks

OceanStore: Automatic maintenance

- All Tapestry state is Soft State
  - State maintained during transformations of network
  - Periodic restoration of state
- Self-Tuning of link structure
- Dynamic insertion
  - New nodes contact small number of existing nodes
  - Integrate themselves automatically
  - Later, introspective optimization will move data to new servers
- Dynamic deletion
  - Node detected as unresponsive
  - Pointer state routed around faulty node (signed deletion requests authorized by servers holding data)

OceanStore: Introspective Optimization

- Monitoring and adaptation of routing substrate
  - Optimization of Plaxton Mesh
  - Adaptation of second-tier multicast tree
- Continuous monitoring of access patterns
  - Clustering algorithms to discover object relationships
    - Clustered prefetching: demand-fetching related objects
    - Proactive-prefetching: get data there *before* needed
  - Time series-analysis of user and data motion
- Continuous testing and repair of information
  - Slow sweep through all information to make sure there are sufficient erasure-coded fragments
  - Continuously reevaluate risk and redistribute data
  - Diagnosis and repair of routing and location infrastructure
  - *Provide for 1000-year durability of information?*

OceanStore: Data eradication

- *Eradication* is antithetical to *durability!*
  - If you can eradicate something, then so can someone else! (denial of service)
  - Must have “eradication certificate” or similar
- Some answers:
  - Bays: limit the scope of data flows
  - Ninja Monkeys: hunt and destroy with certificate
- Related: Revocation of keys
  - Need hunt and re-encrypt operation
- Related: Version pruning
  - Temporary files: don’t keep versions for long
  - Streaming, real-time broadcasts: Keep? Maybe
  - Locks: Keep? No, Yes, Maybe (auditing!)
  - Every key stroke made: Keep? For a short while?

Autonomic features

- Autonomic and Dynamic Optimization *Self-optimization*
- Monitoring and adaptation of routing substrate *Self-configuration*
  - Optimization of Plaxton Mesh
  - Adaptation of second-tier multicast tree
- Continuous monitoring of access patterns *Self-healing*
  - Enhance performance through pro-active movement of data
- Continuous testing and repair of information *Self-protection*
  - Automatic replication for disaster recovery
  - Diagnosis and repair of routing and location infrastructure

- For more Infos
  - OceanStore vision paper for ASPLOS 2000
    "OceanStore: An Architecture for Global-Scale Persistent Storage"
  - OceanStore web site:
    http://oceanstore.cs.berkeley.edu/

Kinesthetics eXtreme (KX) - Columbia University

- Modifications in structure and behavior that can be made to
  - individual components,
  - sets of components
  - the overall target system configuration,
- Modifications include such as adding, removing or substituting components, while the system is running and without bringing it down.
- Goals:
  - Supporting run-time software composition,
  - Enforcing adherence to requirements,
  - Ensuring uptime and quality of service of mission-critical systems,
  - Recovering from and preventing faults,
  - Seamless system upgrading, etc.
- Reference
  - http://www.psl.cs.columbia.edu/kx/

KX: Approach

- Dispatch mobile agents to components to perform dynamic adaptation tasks
- Coordinate concerted action of multiple agents on multiple components via decentralized workflow
- Dynamic adaptation workflow process incorporates knowledge about the specifications and architecture of the target software system
- Allows to address at run-time configuration management, deployment, validation and evolution concerns normally dealt with only at development time

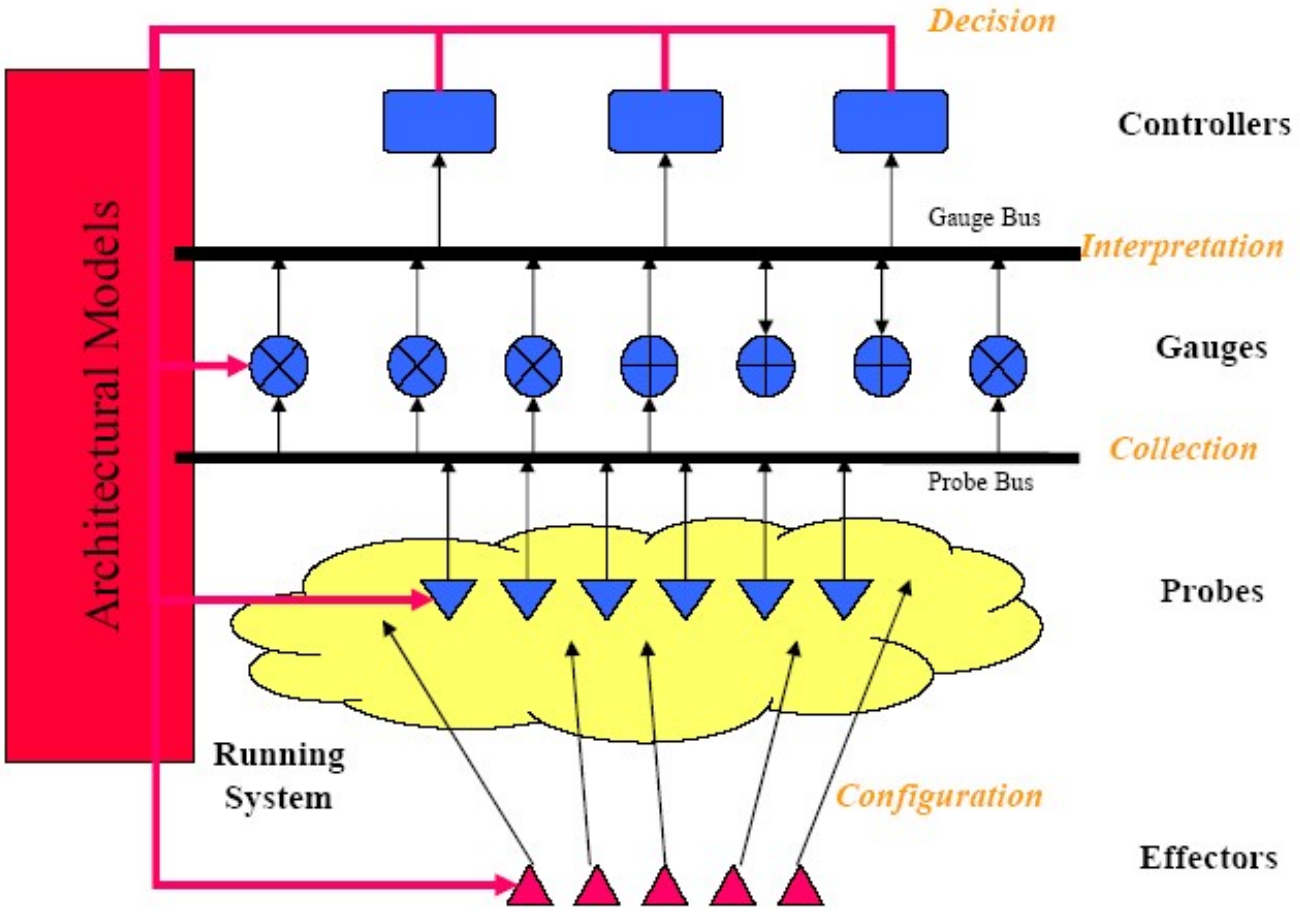
KX: Prerequisite Infrastructure

- Superimpose a minimally intrusive monitoring meta-architecture on top of the target system
- Introduce an adaptation feedback and feedforward control loop onto the target system, detecting and responding to the occurrence of certain conditions at and among components and connectors

KX: Prerequisite Infrastructure

- Probes register
  and report low-level events indicating the behaviors/activities (or lack thereof) of the target system (via active interfaces, probelets, instrumented connectors, others)
- Distributed asynchronous event bus
  receives probed events and propagates them through packaging, filtering and notification facilities
- *Gauges*
  describe/measure the progress of the target system, rendered as visual or textual portal panels or as data feeds into decision support

KX: Prerequisite Infrastructure



The Anthill project

- Motivation
  - Incresaing popularity of P2P systems (Napster, Freenet, Gnutella)
    - Sharing of digital documents (audio, video, etc...)
    - Distributed computing (Seti@Home, Distributed.net)
    - Messaging and collaborative tools (Groove)
  - Poor popularity of P2P systems
    - Copyrighted documents
  - However high potential: The “dark matter” of Internet
    - Huge amount of decentralized resource available
- Goal
  - Provide a framework to design P2P systems
- Reference
  - http://www.cs.unibo.it/projects/anthill/

The Anthill project

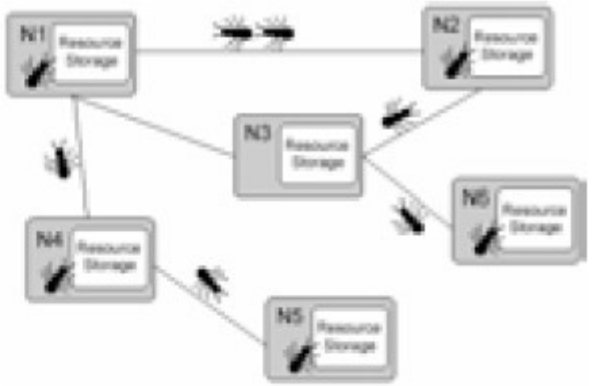
- The Anthill project builds on the similarities between P2P systems and social colonies of ants.
- Anthill construct P2P services that exhibit adaptation and self-organization properties.
- Ant Colony Algorithms
  - Agent Based
    - Artificial Ants of limited individual capabilities move across network of nodes trying to solve a particular problem.
    - While moving they build partial solutions and modify the problem representation by adding collected information.
  - Complex Adaptive
    - Individual ants are unintelligent and have no problem solving capability.
    - Nevertheless ant colonies manage to perform several complicated tasks
- Reference
  - http://www.cs.unibo.it/projects/anthill/

The Anthill project

- In the Anthill framework, a P2P system is composed of a network of connected “nest”
  - A nest is a peer entity running on the machine of some user and capable of
    - Performing some computation
    - Storing documents
  - Nest generates requests or react to requests
  - Request create ants (autonomous agents) which travell across the network and perform some actions like
    - Performing some computation
    - Querying the nest for documents and inserting new documents
    - Release information (“pheromone”) about documents
- The framework anthill cares about the low level details such as security, communication, ants scheduling, etc...
- An evaluation framework is also provided to analyse the behavior of the ants
  - Evaluation is base on ants algorithm simulated in artificial nests

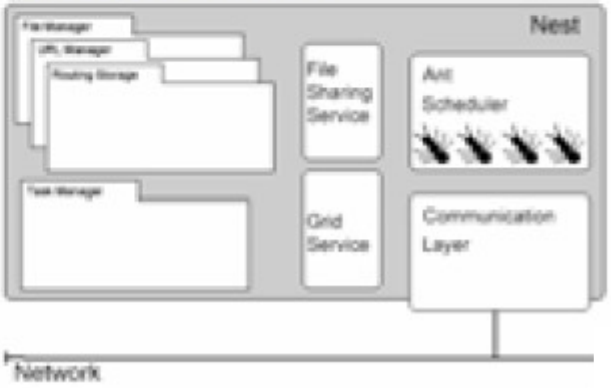
The Anthill project

Ant Colonies	*Anthill Systems*
Real ants need food to survive	Anthill ants need resources to survive (i.e. documents, cpu etc.)
Environment composed of nests, food sources and land.	Anthill networks composed of nests (which are both nest and food sources) and links between them.
Real ants move in the environment in the search of food.	Anthill ants move across the network looking for resources.
Pheromone trails linking nests and food sources.	"Information" trial linking nests and pointing to resources.
Real ants move food from source to their originating nest	Anthill ants copy a resource from nest to nest



Anthill infrastructure composed of self organizing overlay network of nests (N1 to N6).

Applications perform requests and listens for replies through its local nest.



Nest Architecture composed of 3 logical modules

Ant Scheduler, Communication Layer and Resource Manager

Anthill autonomic properties

- Ants are autonomous agents capable of
  - Moving across a nest network.
  - Interacting with the nest they visit to pursue their goals.
  - Characterized by their algorithm (“species”).
  - Behavior of an ant may be
    - non-deterministic (probabilistic).
    - Depends on its algorithm and its current state.
- Evolutionary Framework (Autonomic Nature)
  - Anthill exploits “nature” metaphor using evolutionary techniques
    - Genetic Algorithms in tuning ant algorithms.
    - Minimization of the total path length traversed by ants.
  - Investigates: can genetic algorithms be applied at runtime ??
    - Nests could “steal” algorithms and parameters of visiting ants.
    - Crossover and mutation techniques for generating new ants.

Software Rejuvenation

- Fault classifications
  - Bohrbugs
    - Permanent design fault
    - Deterministic in nature
    - Can be identified easily and weeded out during testing and debugging phase
  - Heisenbugs
    - Permanent
    - Cause transient failures
    - Condition of activation rarely reproducible
      - Typical occur at boundaries between software components
  - Aging-related
    - Like heisenbugs, activated under certain conditions
    - May not be reproducible
- Reference: http://www.software-rejuvenation.com/

Software Rejuvenation

- Software aging
  - Error conditions accumulating over time
    - Leading to performance degradation/crash
  - Not related to application program becoming obsolete due to changing requirements/maintenance
    - Frees up OS resources Removes error accumulation
  - What constitute aging
    - Memory leaking
    - data corruption
    - Storage space fragmentation
    - Round off error accumulation
  - How does it manifest itself
    - Performance degradation, transient failure
  - Common causes
    - Memory leaks, data corruption, fragmentation
- Reference:
  - http://www.software-rejuvenation.com/

Software Rejuvenation

- Software rejuvenation is a proactive fault management technique aimed at cleaning up the system internal state to prevent the occurrence of more severe crash failures in the future.
  - Involves occasionally stopping
    - the running software,
    - Cleaning its internal state and restarting it
  - Counteracts the aging phenomenon
    - Frees up OS resources
    - Removes error accumulation
  - Common techniques for cleaning
    - Garbage collection
    - Defragmentation
    - Flushing kernel and file server tables

Approaches to Rejuvenation

- Open loop periodic
  - No feedback from the system is required
  - Rejuvenation is done periodically (elapsed time)
  - Rejuvenation is done based on the load on the system
- Closed loop (feedback control)
  - Feedback from the system (monitoring)
    - Collects information on the system resources and activities
  - Rejuvenation is performed based on the system health
    - Data analysis to estimate time of exhaustion of a component or an entire system degradation/crash
    - Estimation can be based both on time and system load
    - Estimation can also be based on system failure data analysis
  - Data analysis can be
    - Off-line: best suit for system with deterministic behavior
    - On-line: best suited for system with unpredictable behavior. Future system behavior is computed based on data

Software rejuvenation Examples

- Applications
  - AT&T billing applications
  - On-board preventive maintenance for long-life deep space missions (NASA.s X2000 Advanced Flight Systems Program)
  - Patriot missile system software - switch off and on every 8 hours
  - IBM Director Software Rejuvenation
  - Process Recycling in IIS 5.0

Recovery oriented computing (ROC)

- Philosophy: Even the most robust systems still occasionally encounter failures due to
  - Human operator
  - Permanent hardware failure
  - Software anomalies resulting from heisenbugs or software aging
- Failures are facts, not problems
  - Recovery/repair is how we cope with them
- Improving recovery/repair improves availability

- Five “ROC Solid” Principles
  - Given errors occur, design to recover rapidly.
  - Given humans make errors, build tools to help operator find and repair problems.
    - e.g., undo; hot swap; graceful, gradual SW upgrade.

Recovery oriented computing

- Extensive sanity checks during operation.
  - To discover failures quickly (and to help debug)
- Any error message in HW or SW can be routinely invoked, scripted for regression test.
  - To test emergency routines during development.
- Recovery benchmarks to measure progress.
  - Recreate performance benchmark competition.

➢ Three R's for recovery

- Rewind: roll all system state backwards in time.
- Repair: change system to prevent failure.
  - e.g., fix latent error, retry unsuccessful operation, install preventative patch.
- Replay: roll system state forward, replaying end-user interactions lost during rewind.

➢ Reference: http://roc.cs.berkeley.edu/

Recovery oriented computing – issues

- Isolation and redundancy
  - Isolation is crucial in fault containment and safe on-line recovery
    - Ability to isolate portion of the system
  - Isolation demands redundancy
    - Continue service delivery while portion of the system are isolated
  - Isolation and redundancy must be failure-proof under a broad failure model, including all software and human-induced failures
- System wide support for undo
  - Human error is the largest single cause of failure (data analysis)
  - Undo facilities to allows the human to recover from their errors
    - For complex performed operation, “repair the past”
    - Three steps useful
      - Rewinding time
      - Repair problems
      - Replaying the system back to the current time

Recovery oriented computing – issues

- Integrated diagnostic support
  - Rapidly detect the presence of failures
  - Identify root failure for quick containment and repair
  - Self-testing and verification of the behavior of all modules a system depends on
  - Automate root cause analysis
- Dependability/Availability benchmarking
  - Develop benchmark that provides impartial measure of system dependability
  - Berkeley benchmark uses
    - Injection of system-level faults and perturbation to evaluate the impact of realistic failure on delivered quality of service
    - Based on collection of faults and failure from real internet service environments
    - Definition of metrics for dependability

Pinpoint

- Motivation
  - Systems are large and getting larger
  - Systems are dynamic
  - Difficult to diagnose failures
- Pinpopint V1
  - Fault diagnosis via data clustering
    - Off-line analysis;
  - Sun's J2EE reference implementation
    - Single-node; log EJB, JSP, and JSP tags
  - Results: trade-off accuracy against false-positives
    - Accuracy: 70-90%. False-positives: 20-40%
    - Other techniques: Either have poor accuracy (40%) or many false-positives (90%)

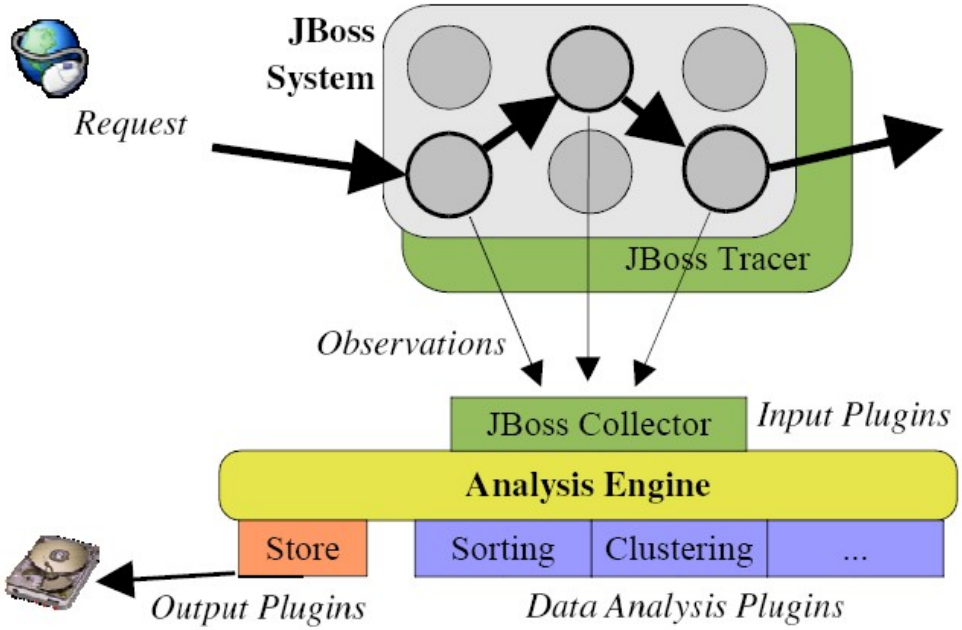
Pinpoint

- Pinpoint V2
  - On-line analysis
    - Instrument more robust, clustered J2EE platform (JBoss)
  - Tools to attack wider range of problems
    - Deducing application structure
    - Detecting application-level faults
  - Improve fault diagnosis for multiple component faults
  - Integration with repair processes

- Home assignment:
  - Investigate the clustering policy for fault root identification and isolation

Pinpoint

- Implementation
  - Built on top of J2EE platform
  - Version 2 of Pinpoint
    - instruments JBoss middleware.
    - Observe calls to and returns from components, exceptions
    - Record component details, SQL queries, timestamp
    - Record path context: request id, sequence number
- Reference
  - http://pinpoint.stanford.edu/



Sabio (IBM)

- Sabio takes large collections of documents within an enterprise and breaks them down automatically into a taxonomy.
  - unaided by human categorizers.
  - Automated Taxonomy Generator as it is called in Raven.
- employs Bayesian statistics
- decompose each document into a collection of "tokens.
- assembles a collection of relevant words and phrases in all the documents.
  - treats this collection mathematically as points in a huge multidimensional space.
    - each dimension corresponds to a single word or phrase,
  - number of times the word or phrase appears determines how far out along the dimension the point lies.
- 2 documents which share many of the same words and phrases relatively close together in this multidimensional space.
- Combined with the Lotus Product

SMART DB2

- The DB2 SMART project aims to create technology for reducing human
  - intervention and cost in DB2 operation.
- It builds on and extends existing self-managing technologies in DB2.
  - Adjust every configuration parameter dynamically while the system is in use
  - Expand and shrink memory usage, based on workload
  - Automatically profile workloads and recommend and create indexes,
  - partitioning, clustering, summary tables, and so on to improve performance
  - Automatically detect the need for, estimate the duration of, and schedule maintenance operations such as reorganization, statistics collection, backup, copy
  - Observe actual performance and exploit that information to improve operations
  - Recommend action when the performance isn't meeting the DBA's expectations
  - Predict problems such as low memory or constrained disk space and notify someone by pager or email in advance.

SMART DB2 – Autonomic features

- Query Optimizer
  - Automatically determines the best way to execute a declarative SQL query.
- Automatic selection of degree of parallelism
  - Setting and adjusting degree of parallelism for queries and utilities.
- Detection of partial disk writes
  - Protects data integrity by automatically detecting any corrupted data from incomplete I/O's.
- Application Control and Tuning
  - Query Patroller
    - "Predictive Governer" uses the "Query Optimizer" estimate of relative resources for each query to limit surges of arriving or long running queries.
    - "Reactive Governer" monitors the actual resources consumed to prevent runaway queries.
- Performance Expert
  - Performs passive monitoring and collects trace and monitor data in a performance data warehouse.
  - Buffer Pool Analyzer
    - Collects buffer pool activity and models changes to the objects in the buffer pools.

Autoadmin - Microsoft

- Self Tuning and Self Administering Databases.
- Enabling databases to track the usage of their systems and to gracefully adapt to application requirements
- Bottom up approach
  - Choose appropriate physical objects and their organization
    - Indexes
    - Materialized Views
    - Statistics
  - Goal: Optimize performance

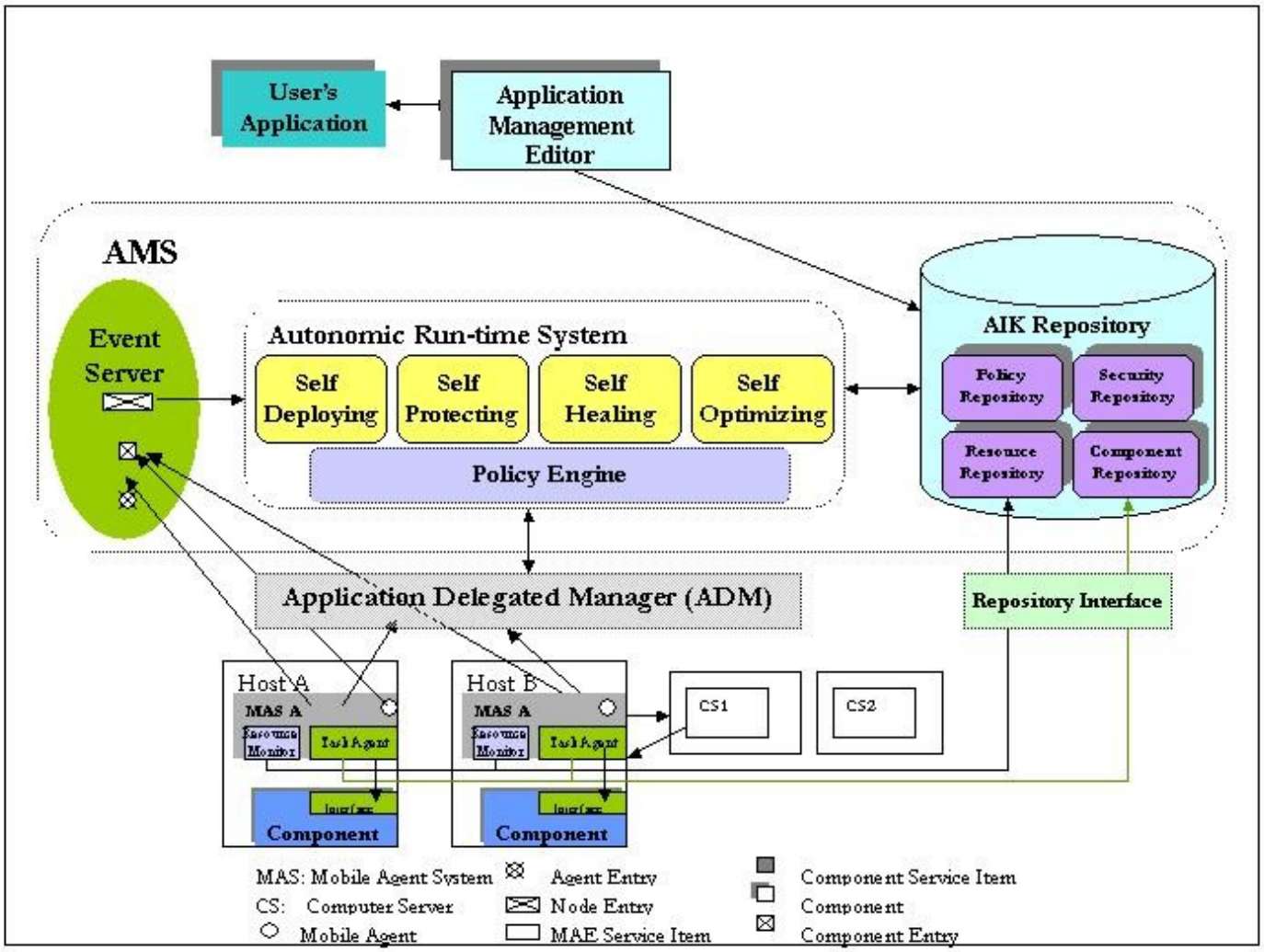
Autonomia - University of Arizona

- AUTONOMIA environment provides the application developers
  - Tools required to specify the appropriate control and management schemes
    - to maintain any quality of service requirement
    - application attribute/functionality (e.g., performance, fault, security, etc.)
  - Core autonomic middleware services
- Self-Configuring Engine
  - Responsible for configuring/reconfiguring the applications on the air.
  - Chooses the appropriate policy specified by the self-configuring profile to configure the application.
- Self Optimizing
  - Optimizes application as well as system performance at runtime
  - Handler selects appropriate mechanism to optimize application performance

Autonomia

- Self-Protecting Handler
  - Uses the idea of intention list to make decisions on the fly about access control to various tasks.
  - Autonomia Security Manager constantly monitors the agent intention list and tasks.
  - Agent only allowed to execute tasks published in the intention list.
  - Any deviation causes loss of further access for the agent.
- Reference
  - http://www.ece.arizona.edu/~hpdc/projects/AUTONOMIA/
- Home assignment:
  - Investigate the self healing, self configuring, self protecting algorithms in AUTONOMIA

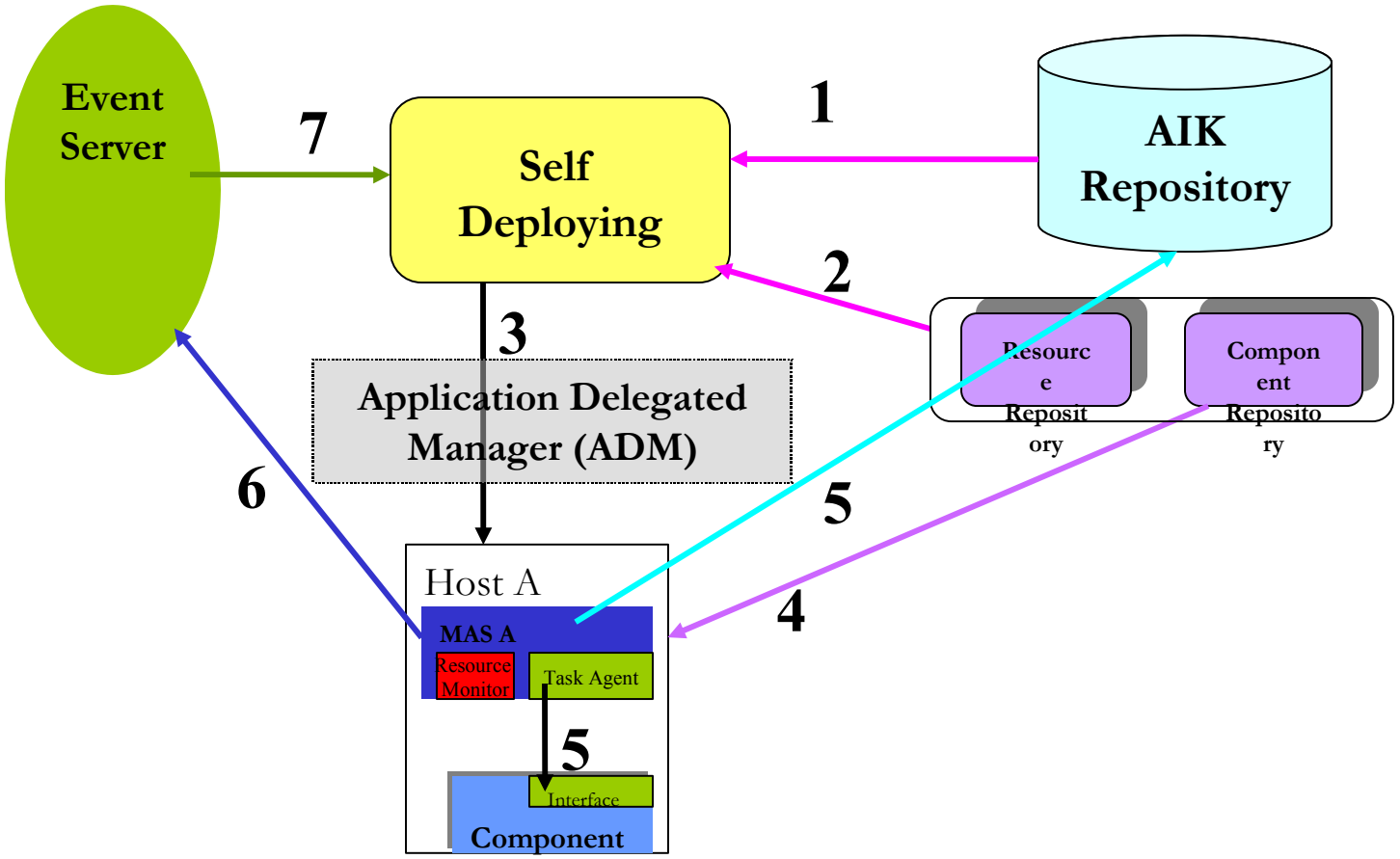
Autonomia - Architecture



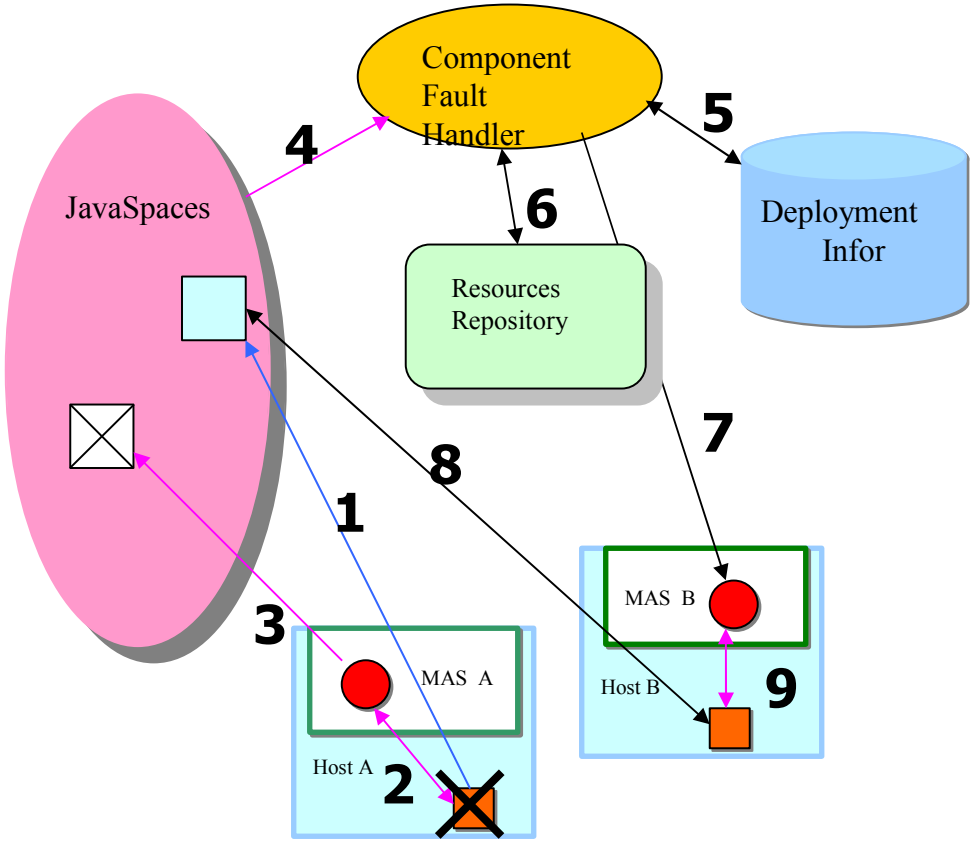
Autonomia – main components

- Application Management Editor (AME)
  - Enable users to develop their application with the ability to specify the control and management policy
- Autonomic Middleware Servicem
  - Provides common middleware service and tools.
  - The main modules are:
    - Component repository
    - Resource repository
    - Application Information and Knowledge repository
    - Event server
    - Policy engine and autonomic services
- Application Delegate Manager
  - Sets up the application execution environment und maintaining its requirements at run-time
- Monitoring Services
  - Allow the components and system resources to be monitored

Autonomia – self deployement/configuration



Autonomia – Fault Recovery Mechanism



Step:

2. **Checkpointing regularly**
3. **Detect a component failure**
4. **Report a component fault entry**
5. **Notify the fault handler**
6. **Read the component type**
7. **Get another available MAS**
8. **Dispatch an agent to the host**
9. **Read the checkpoint**
10. **Resume execution**

Smart Grid - IBM

- OptimalGrid is a project in the distributed systems department at the IBM Almaden Research Center designed to solve the next generation of large scale parallel problems on a large number of network-attached, heterogeneous compute nodes (i.e. "The Grid“)

- OptimalGrid automates aspects of solving a large scale connected problem on a computing Grid
- Research prototype of grid-enabled middleware designed to hide complexities of
  - partitioning,
  - distributing, and
  - load balancing.
- Reference - download
  - http://www.alphaworks.ibm.com/tech/optimalgrid/download

Smart Grid - Prototype

- Major components:
  - Autonomic Program Manager (APM)
  - Variable Problem Partitions (VPP)
  - Computing Agents (CA)
  - Autonomic Rule Engine (ARE)
  - Micropayment Broker (MPB)
  - UDDI Server (Universal Description Discovery Integration)
  - OSGI (Open Services Gateway Initiative)
- Component Roles:
  - APM employs the ARE and manages CAs
  - CAs run VPPs, communicating with other VPPs (CAs) through some mechanism.
  - CAs log performance data that is used by the ARE to adjust the VPP sizes(allocations) for each of the CAs