Towards Autonomic Computational Science & Engineering (Autonomic Computing: Application Perspective)

Manish Parashar
The Applied Software Systems Laboratory
ECE/CAIP, Rutgers University
http://www.caip.rutgers.edu/TASSL
Ack: NSF (CAREER, KDI, ITR, NGS), DoE (ASCI-CIT)

THE CURRENT TEAM

The AutoMate team is composed of faculty and graduate and undergraduate students at The Applied Software Systems Laboratory, Department of Electrical and Computer Engineering and Center of Advanced Information Processing (CAIP), Rutgers, The State University of New Jersey. The team is organized as the Autonomic Computing Research Group and the Autonomic Applications Research Group. This research builds on our collaborations with application scientists, engineers and computer and computational scientists at California Institute of Technology, University of Texas at Austin, University of Arizona, Ohio State University, and University of Maryland.

Figure 1

OVERVIEW OF THE TALK

This talk motivates and introduces autonomic computational science and engineering, and presents the AutoMate framework for enabling autonomic applications on Grid. It describes the AutoMate architecture and briefly presents each of its components. These include the ACCORD autonomic component framework, the RUDDER decentralized deductive engine, the SESAME context-sensitive dynamic access management framework, the Pawn peer-to-peer messaging substrate, and the SQUID decentralized discovery service. Finally, it describes two applications of autonomic computing to science and engineering – autonomic runtime management framework for adaptive applications (V-Grid) and autonomic interactions for oil reservoir optimization.

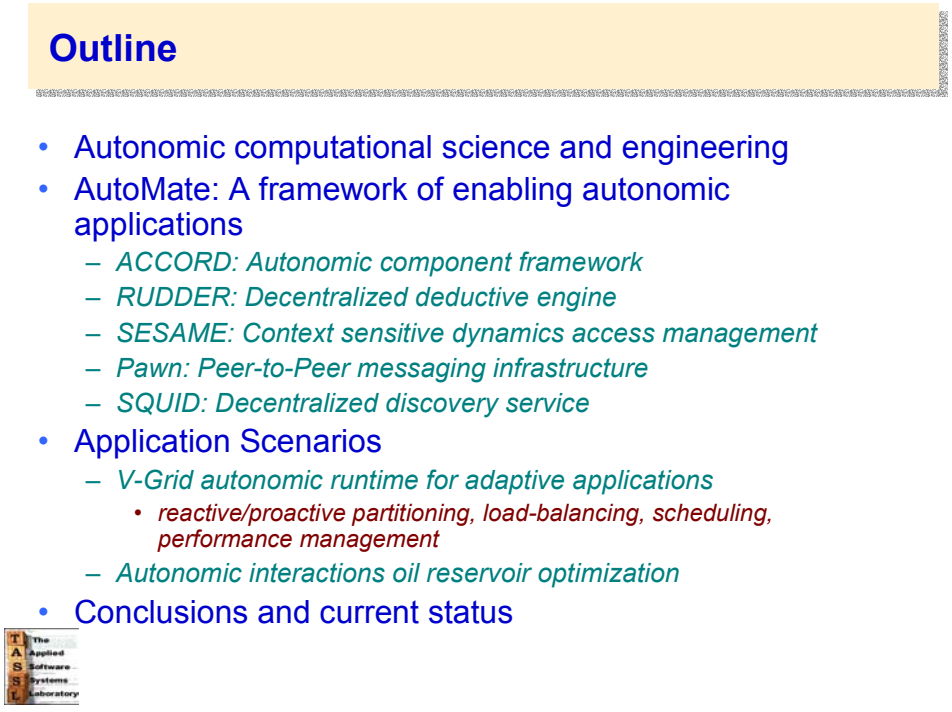


Figure 2

COMPUTATION MODELING OF PHYSICAL PHENOMENA

Realistic, physically accurate simulations of complex physical phenomena that symbiotically and opportunistically combine computations, experiments, observations, and real-time data have the potential for providing dramatic insights into complex systems such as interacting black holes and neutron stars, formations of galaxies, subsurface flows in oil reservoirs and aquifers, and dynamic response of materials to detonations. However, the phenomena being modeled by these applications are inherently large-scale, dynamic and heterogeneous (in time, space, and state). Furthermore, the applications are extremely large with unprecedented resource requirements, and are composed of a large numbers of software components with very dynamic compositions and interactions between these components.

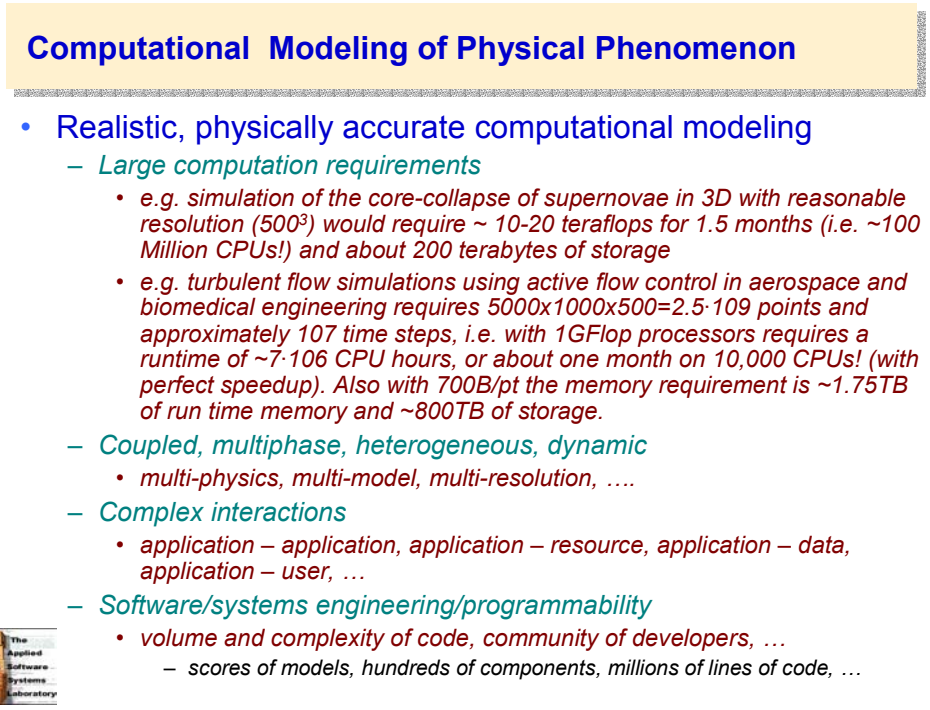


Figure 3

COMPUTATION MODELING AND THE GRID

The emergence of computational Grids and the potential for seamless aggregation, integration and interactions has made it possible to conceive the realistic, scientific and engineering simulations of complex physical phenomena described in the previous slide. However, the Grid infrastructure is also heterogeneous and dynamic, globally aggregating large numbers of independent computing and communication resources, data stores and sensor networks. The combination of the two (large, complex, heterogeneous and dynamic applications and Grids) results in application development, configuration and management complexities that break current paradigms based on passive components and static compositions. Clearly, there is a need for a fundamental change in how these applications are formulated, composed and managed so that their heterogeneity and dynamics can match and exploit the heterogeneous and dynamic nature of the Grid. In fact, we have reached a level of complexity, heterogeneity, and dynamism for which our programming environments and infrastructure are becoming unmanageable brittle and insecure. This has led researchers to consider alternative programming paradigms and management techniques that are based on strategies used by biological systems to deal with complexity, heterogeneity and uncertainty. The approach is referred to as autonomic computing. An autonomic computing system is one that has the capabilities of being self-defining, self-healing, self-configuring, self-optimizing, self-protecting, context aware, and open.

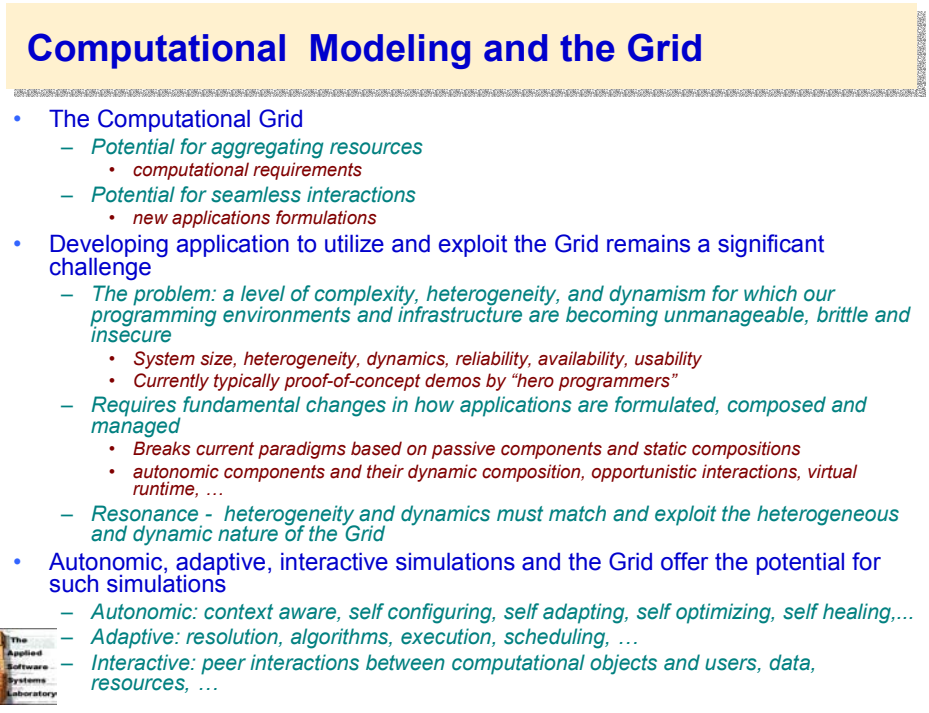


Figure 4

AUTOMATE

The overall objective of the AutoMate project is to investigate key technologies 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. Specifically, it will investigate the definition of autonomic components, the development of autonomic applications as dynamic composition of autonomic components, and the design of key enhancements to existing Grid middleware and runtime services to support these applications.

Definition of Autonomic Components: The definition of programming abstractions and supporting infrastructure that will enable the definition of autonomic components. In addition to the interfaces exported by traditional components, autonomic components provide enhanced profiles or contracts that encapsulate their functional, operational, and control aspects. These aspects export information and policies about their behavior, resource requirements, performance, interactivity and adaptability to system and application dynamics. Furthermore, they encapsulate sensors, actuators, access policies and a policy-engine. Together, aspects, policies, and policy engine allow autonomic components to consistently configure, manage, adapt and optimize their execution.

Dynamic Composition of Autonomic Applications: The development of mechanisms and supporting infrastructure to enable autonomic applications to be dynamically and opportunistically composed from autonomic components. The composition 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: The design, development, and deployment of key services on top of the Grid middleware infrastructure to support autonomic applications. One of the key requirements for autonomic behavior and dynamic compositions is the ability of the components, applications and resources (systems, services, storage, data) to interact as peers. Furthermore the components should be able to sense their environment. In this project, we extend the Grid middleware with (1) a peer-to-peer messaging substrate, (2) context aware services, and (3) peer-to-peer deductive engines for composition, configuration and management of autonomic applications. An active peer-to-peer control network will combine sensors, actuators and rules to configure and tune components and their execution environment at runtime and to satisfy requirements and performance and quality of service constraints.

AutoMate: Enabling Autonomic Applications

- Objective:
  - *Investigate key technologies 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 requirements for autonomic behavior and dynamic compositions is the ability of the components, applications and resources (systems, services, storage, data) to interact as peers*

Figure 5

AUTOMATE ARCHITECTURE

AutoMate builds on the emerging Grid infrastructure and extends the Open Grid Service Architecture (OGSA). AutoMate is composed of the following components:

AutoMate System Layer: The AutoMate system layer builds on the Grid middleware and OGSA and extends core Grid services (security, information and resource management, data management) to support autonomic behavior. Furthermore, this layer provides specialized services such as peer-to-peer semantic messaging, events and notification.

AutoMate Component Layer: The AutoMate component layer addresses the definition, execution and runtime management of autonomic components. It consists of AutoMate components that are capable of self configuration, adaptation and optimization, and supporting services such as discovery, factory, lifecycle, context, etc. (which builds on core OGSA services).

AutoMate Application Layer: The AutoMate application layer builds on the component and system layers to support the autonomic composition and dynamic (opportunistic) interactions between components.

AutoMate Engines: AutoMate engines are decentralized (peer-to-peer) networks of agents in the system. The context-awareness engine is composed of context agents and services and provides context information at different levels to trigger autonomic behaviors. The deductive engine is 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. Finally, the trust and access control engine is composed of access control agents and provides dynamic context-aware control to all interactions in the system.

In addition to these layers, 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. AutoMate leverages the experiences and technologies developed as part of the Discover/DIOS computational collaboratory project (http://www.discoverportal.org). The different components are described in the following sections.

AutoMate: Architecture

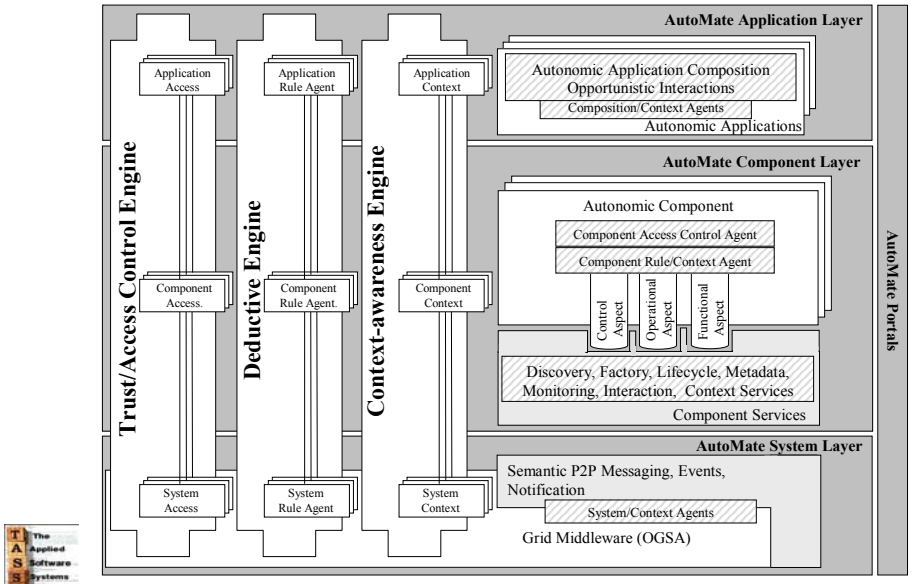


Figure 6

AUTOMATE ARCHITECTURE

Key components of AutoMate include:

- ACCORD (Autonomic Components, Compositions and Coordination) component framework that enables the definition of autonomic components, their autonomic compositions and opportunistic interactions.
- RUDDER (Rule Definition Deployment and Execution Service) decentralized deductive engine.
- SESAME (Scalable Environment Sensitive Access Management Engine) dynamic access control engine.
- Pawn decentralized (P2P) messaging substrate.
- SQUID flexible information discovery service.

These components are introduced in the following slides.

AutoMate: Components

- **ACCORD: Autonomic application framework**
- **RUDDER: Decentralized deductive engine**
- **SESAME: Dynamic access control engine**
- **Pawn: P2P messaging substrate**
- **SQUID: P2P discovery service**

Figure 7

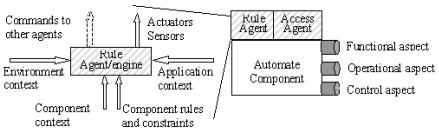
ACCORD: AUTONOMIC COMPONENTS

Autonomic components in AutoMate export information and policies about their behavior, resource requirements, performance, interactivity, and adaptability to system and application dynamics. In addition to the functional interfaces exported by traditional components, AutoMate components provide semantically enhanced profiles or contracts that encapsulate their functional, operational, and control aspects. A conceptual overview of an AutoMate component is presented in the figure. The functional aspect specification abstracts component functionality, such as order of interpolation (linear, quadratic, etc.). This functional profile is then used by the compositional engine to select appropriate components based on application requirements. The operational aspect specification abstracts a component's operational behavior, including computational complexity, resource requirements, and performance (scalability). This profile is then used by the configuration and runtime engines to optimize component selection, mapping and adaptation. Finally, the control aspect describes the adaptability of the component and defines sensors/actuators and policies for management, interaction and control.

AutoMate components also encapsulate access policies, rules, a rule agent, and an access agent that allow the components to consistently and securely configure, manage, adapt and optimize their execution based on rules and access policies. The access agent is a part of the AutoMate access control engine and the underling dynamic access control model, and manages access to the component based on its current context and state. The rule agent is part of RUDDER, the AutoMate deductive engine and manages local rule definition, evaluation and execution at the component level. Rules can be dynamically defined (and changed) in terms of the component's interfaces (based on access policies) and system and environmental parameters. Execution of rules can change the state, context and behavior of a component, and can generate events to trigger other rule agents.

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.*



- AutoMate 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/polices 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*

Figure 8

ACCORD: AUTONOMIC COMPOSITIONS

Applications are typically composed with well defined objectives. In case of autonomic applications, however, these objectives can dynamically change based on the state of the application and/or the system. As a result, we need to dynamically select components and compose them at runtime based on current objectives. Together, the profiles, policies, and rules allow autonomous components to consistently and securely manage and optimize their executions. Furthermore, they enable applications to be dynamically composed, configured and adapted. Dynamic application work-flows can be defined to select the most appropriate components based on user/application constraints (highest-performance, lowest cost, reservation, execution time upper bound, best accuracy), on the current applications requirements, to dynamically configure the component's algorithms and behavior based on available resources or system and/or applications state, and to adapt this behavior if necessary.

The AutoMate dynamic composition model may be viewed as transforming a given composition or workflow into a new one by adding or modifying interactions and participating entities. Its primary goal is to enable dynamic (and opportunistic) choreography and interactions of components and services to react to the heterogeneity and dynamics of the application and underlying execution environment to produce the desired user objectives.

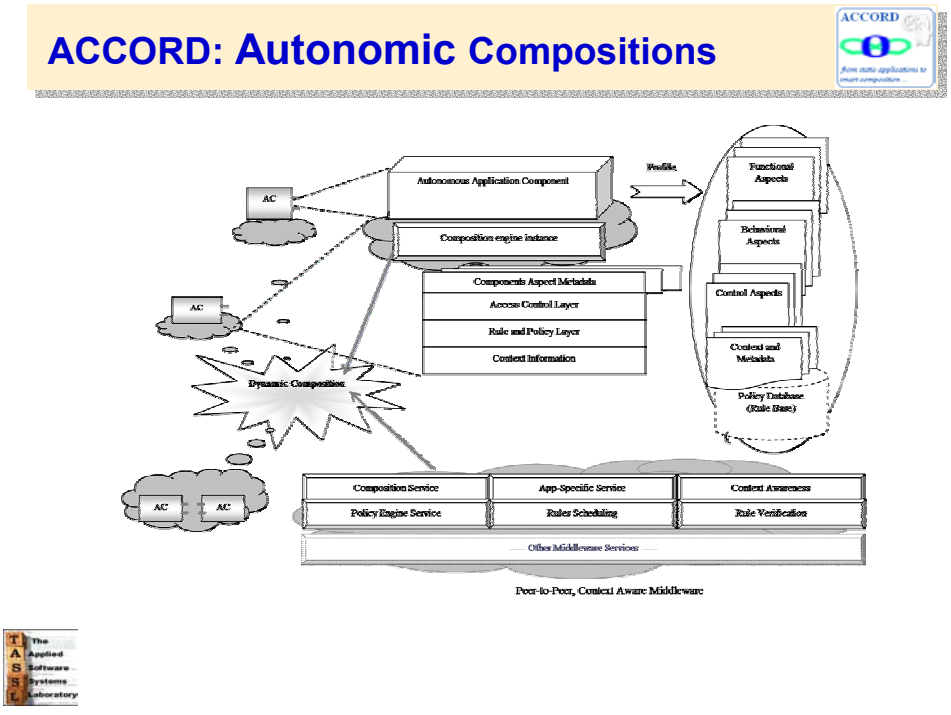


Figure 9

ACCORD: OPPORTUNISTIC INTERACTIONS

Opportunistic interactions are decentralized and based on the satisfaction of locally defined goals and constraints. These interactions are inherently dynamic and ad-hoc and use semantic publisher/subscriber messaging based on proximity, privileges, capabilities, context, interests, and offerings. The goals/constraints are typically long-term and may or may not be satisfied. The interactions do not involve explicit synchronization – the semantics are achieved through feedback and consensus building mechanisms.

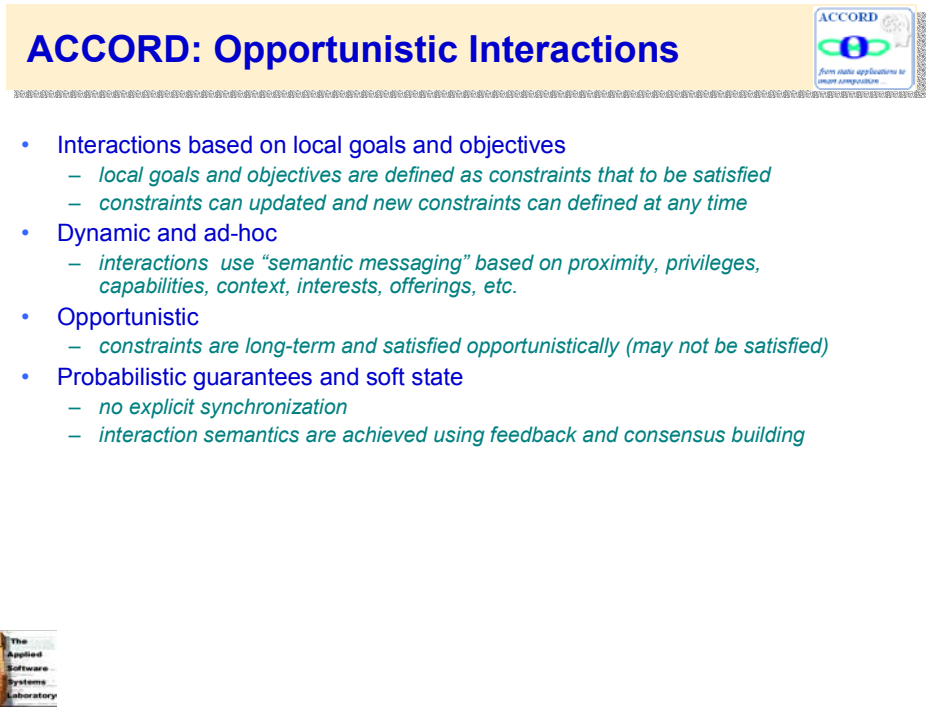


Figure 10

RUDDER: DEDUCTIVE ENGINE

RUDDER provides the core capabilities for supporting autonomic compositions, adaptations, and optimizations. It is a decentralized deductive engine composed of distributed specialized agents (component rule agents, composition agents, context agents and system agents) that exist at different levels of the system, and represents their collective behavior. It provides mechanisms for dynamically defining, configuring, modifying and deleting rules. Furthermore it defines an XML schema for composing rules and provides mechanisms for deploying and routing rules, decomposing and distributing them to relevant agents, and for coordinating the execution of rules. It also manages conflict resolutions within a single entity and across entities.

The figure presents a schematic overview of RUDDER. It builds on AutoMate and Grid services and the underlying semantic messaging infrastructure. Rules can be dynamically injected into the system and are routed by the messaging substrate to the appropriate agents. Furthermore, the agents may hierarchically decompose a rule and distribute it to peer agents. For example, an application level rule may be decomposed into sub-rules that are assigned to its components. The components rules may be further decomposed into rules for the underlying systems entities.

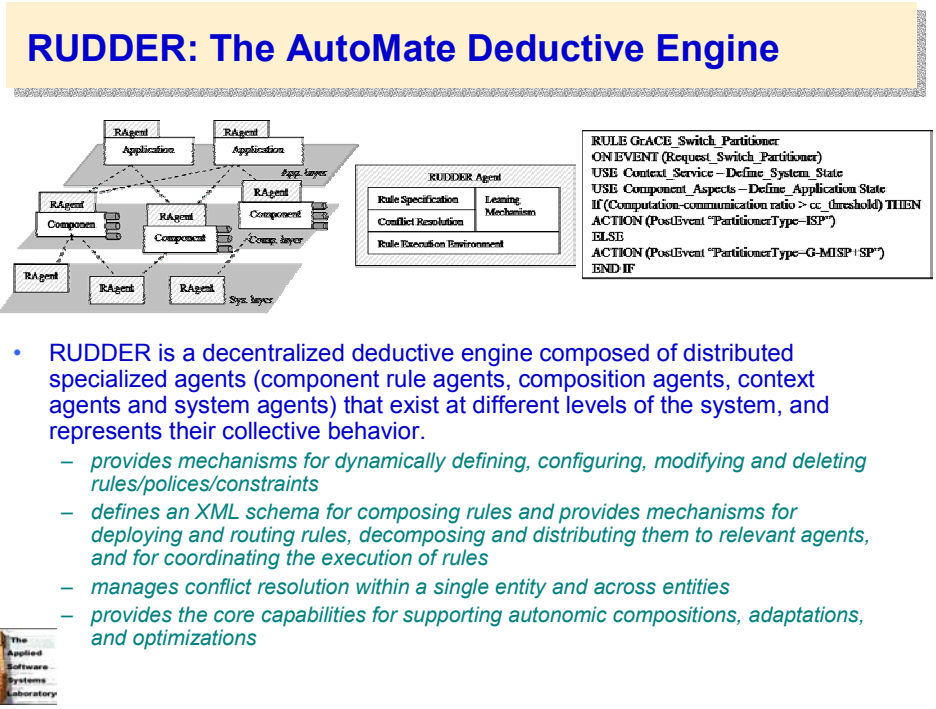


Figure 11

SESAME: CONTEXT SENSITIVE ACCESS MANAGEMENT

A key requirement of autonomic applications is the support for dynamic, seamless and secure interactions between the participating entities, i.e. components, services, application, data, instruments, resources and users. Ensuring interaction security requires a fine grained access control mechanism. Furthermore, in the highly dynamic and heterogeneous Grid environment, the access rights of an entity depend on the entity's privileges, capabilities, context and state. For example, 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. secure connection) and the state of the resource or component. The AutoMate Access Control Engine addresses these issues and provides dynamic access control to users, applications, services, components and resources. The engine is composed of access control agents associated with various entities in the system. The underlying dynamic role based access control mechanism extends the RBAC (Role Based Access Control) model to make access control decision based on dynamic context information. The access control engine dynamically adjusts *Role Assignments* and *Permission Assignments*.

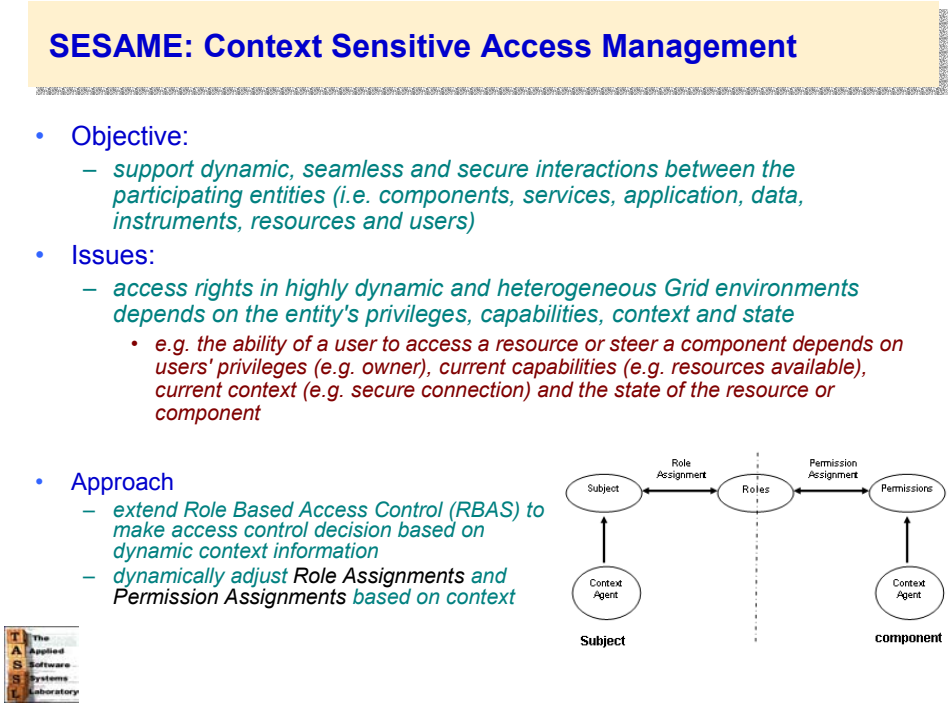


Figure 12

PAWN: P2P MESSAGING

Pawn is a peer-to-peer messaging substrate that builds on project JXTA to support peer-to-peer interactions on the Grid. Pawn provides a stateful and guaranteed messaging to enable key application-level interactions such as synchronous/asynchronous communication, dynamic data injection, and remote procedure calls. It exports these interaction modalities through services at every step of the scientific investigation process, from application deployment, to interactive monitoring and steering, and group collaboration.

A conceptual overview of the Pawn P2P substrate is presented in the figure. Pawn is composed of peers (computing, storage, or user peers), network and interaction services, and mechanisms. These components are layered to represent the requirements stack enabling interactions in a Grid environment. The figure can be read from bottom to top as: "Peers compose messages handled by services through specific interaction modalities".

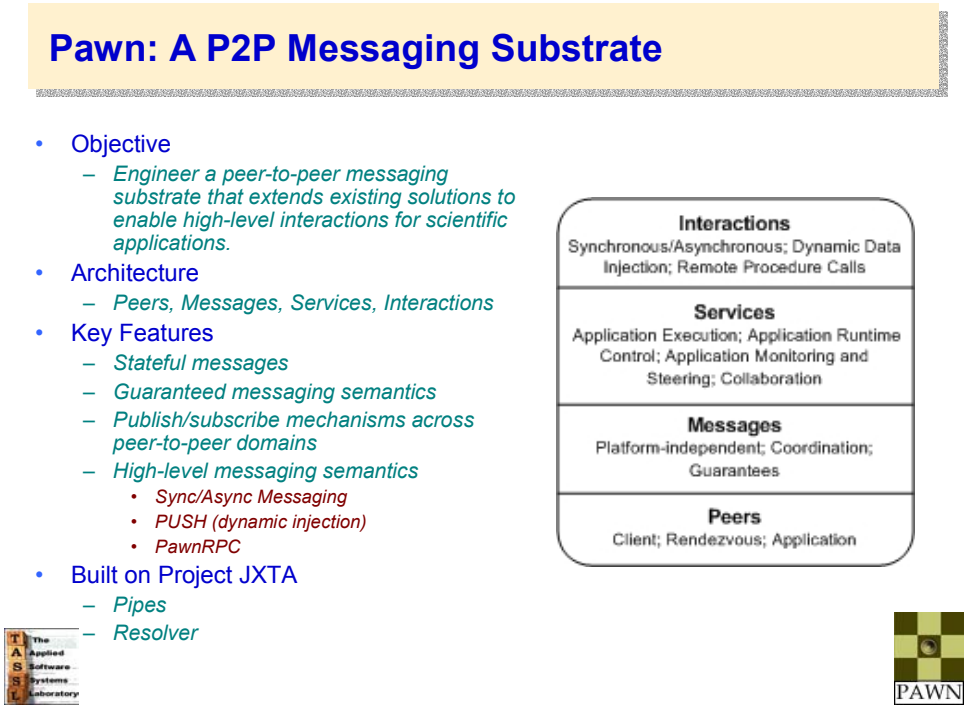


Figure 13

SQUID: DECENTRALIZED Discover

A fundamental problem in large, decentralized, distributed resource sharing environments such as the Grid is the efficient discovery of information, in the absence of global knowledge of naming conventions. For example 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. The 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. An ideal information discovery system has to be efficient, fault-tolerant, self-organizing, has to offer guarantees and support flexible searches (using keywords, wildcards, range queries). Decentralized peer-to-peer (P2P) systems, by their inherent properties (self-organization, fault-tolerance, scalability), provide an attractive solution.

SQUID supports decentralized information discovery in AutoMate. It is a P2P system that supports complex queries containing partial keywords, wildcards, and range queries, and 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 key innovation is a dimension reducing indexing scheme that effectively maps the multidimensional information space to physical peers.

Figure 14

SQUID OPERATION

The overall architecture of SQUID is a distributed hash table (DHT), similar to typical data lookup systems. The key difference is in the way we map data elements to the index space. In existing systems, this is done using consistent hashing to uniformly map data element identifiers to indices. As a result, data elements are randomly distributed across peers without any notion of locality. Our approach attempts to preserve locality while mapping the data elements to the index space. In our system, all data elements are 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). These 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. Thus, we map documents that are local in this multi-dimensional index space to indices that are local in the 1-dimensional index space, which are then mapped to the same node or to nodes that are close together in the overlay network. This mapping is derived from a locality-preserving mapping called Space Filling Curves (SFC).

In the current implementation, we use the Hilbert SFC for the mapping, and Chord for the overlay network topology. The overall operation of SQUID is presented in the figure. (a) shows a 2-dimensional keyword space. The data element "*Document*" is described by keywords "*Computer*" and "*Network"*. (b) shows the mapping of the 2-dimensional space to a curve. The query (011, *) defines clusters on the curve (segments). (c) shows the recursive refinement of query (011, *) viewed as a tree. Each node is a cluster, and the bold characters are the cluster's prefixes. (d) illustrates the query resolution process by embedding the leftmost tree path (solid arrows) and the rightmost path (dashed arrows) onto the overlay network topology.

SQUID: Operation

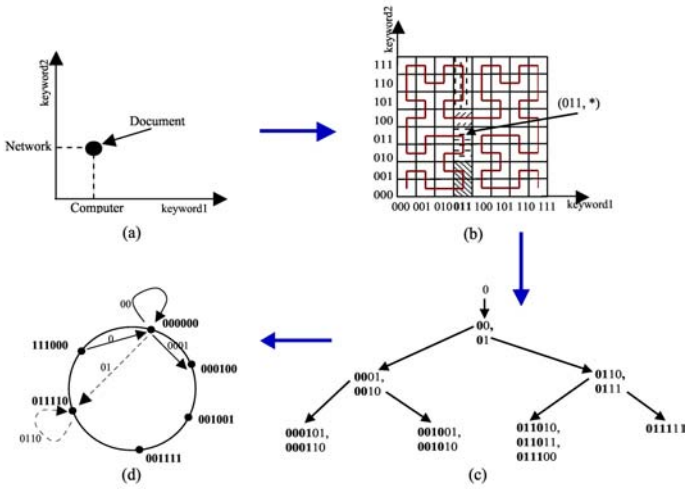


Figure 15

V-GRID AUTONOMIC APPLICATION MANAGEMENT

Truly realistic scientific and engineering simulations require enormous amounts of resources that can surpass even the aggregated capacity of the Grid. The V-Grid (virtual Grid) infrastructure is an application of autonomic computing to science and engineering that is based on the concept of virtualizing grid resources and application execution (analogous to virtual memory). The V-Grid autonomic runtime management framework allows the implementation of a simulation to be driven by the requirements of the science being modeled rather than the size and configuration of the machine that it will be run on.

The autonomic behavior in the V-Grid has three primary aspects: (1) V-Grid Monitoring, (2) V-Grid Deduction, and (3) V-Grid Execution.

The V-Grid monitoring engine is a decentralized entity composed of context agents that provides application and system context awareness. Application monitoring uses sensors exported by the autonomic components and services and provides information about the current state, dynamics and requirements of components and the application. System/resource monitoring builds on context information provided by OGSA and existing Grid middleware (e.g. NWS, Globus, Autopilot) and extends their capabilities to support dynamic monitoring requirements and information aggregation.

The V-Grid deduction engine uses application/components specifications, context and predicted behavior to deduce objective functions and execution and management strategies. This includes identifying and characterizing natural regions, defining Virtual Computational Units or VCUs that reflect the current state of the application, mapping them onto Virtual Resource Units or VRUs based on their specifications, and outlining scheduling policies and constraints. This mapping of VCUs onto VRUs exploits the spatial, temporal and functional heterogeneity of the application to reduce couplings and maximize performance.

The V-Grid execution engine implements polices and strategies defined by the deduction engine using OGSA and autonomic Grid services. The main activities of this engine are (1) dynamic reservation and allocation of VRUs, (2) adaptive mapping and scheduling of VCUs to VRUs, and VRUs to physical resources, and (3) autonomic management, control and adaptation of application execution.

V-Grid: Autonomic Application Management

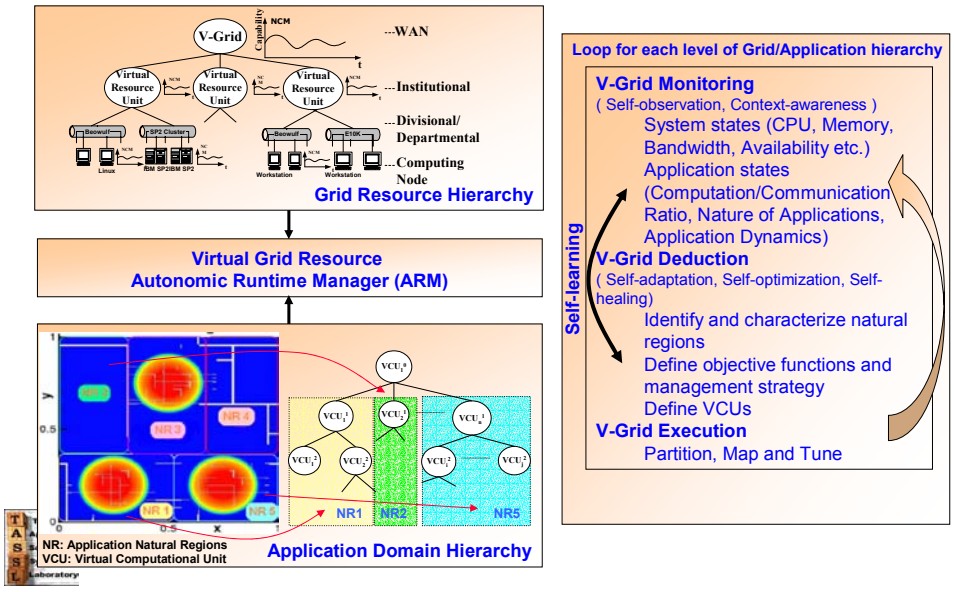


Figure 16

ADAPTIVE MESH REFINEMENT

Dynamically adaptive mesh refinement (AMR) methods for the numerical solution to partial differential equations (PDEs) employ locally optimal approximations, and can yield highly advantageous ratios for cost/accuracy when compared to methods based upon static uniform approximations. These techniques seek to improve the accuracy of the solution by dynamically refining the computational grid in regions with large local solution error.

Structured AMR (SAMR) techniques start with a coarse base grid with minimum acceptable resolution that covers the entire computational domain. As the solution progresses, regions in the domain with large solution error, requiring additional resolution, are identified and refined. Refinement proceeds recursively so that the refined regions requiring higher resolution are similarly tagged and even finer grids are overlaid on these regions. The resulting grid structure is a dynamic adaptive grid hierarchy (such as the SAMR formulation by Berger and Oliger, shown in the figure).

Methods based on SAMR can lead to computationally efficient implementations as they require uniform operations on regular arrays and exhibit structured communication patterns. Distributed implementations of these methods, however, lead to interesting challenges in dynamic resource allocation, data-distribution, load-balancing, and runtime management.

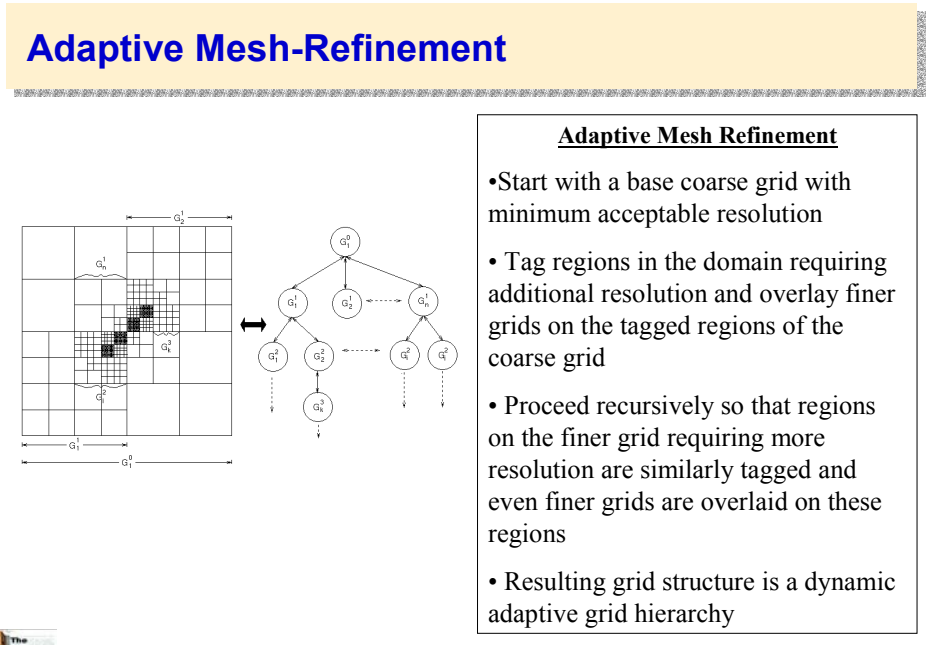


Figure 17

STRUCTURE ADAPTIVE MESH REFINEMENT APPLICATIONS

Structured adaptive mesh refinement (SAMR) methods are being effectively used for adaptive PDE solutions in many domains, including computational fluid dynamics, numerical relativity, astrophysics, and subsurface modeling and oil reservoir simulation.

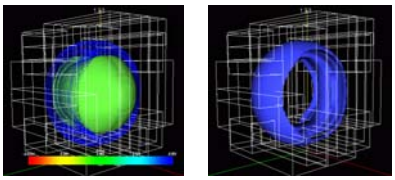
The top-left application belongs to the Zeus kernel coupled with GrACE (SAMR infrastructure) and Cactus (problem solving environment) packages, and shows a 3-D blast wave in the presence of a uniform magnetic field with 3 levels of refinement. Zeus-MP solves the equations of ideal (non-resistive), non-relativistic, hydrodynamics and magnetohydrodynamics, including externally applied gravitational fields and self-gravity.

The top-right figure is taken from the IPARS oil reservoir simulator and shows the multi-block grid structure and oil concentration contours. The MACE (Multi-block Adaptive Computational Engine) infrastructure support multi-block grids where multiple distributed and adaptive grid blocks with heterogeneous discretization are coupled together with lower dimensional mortar grids.

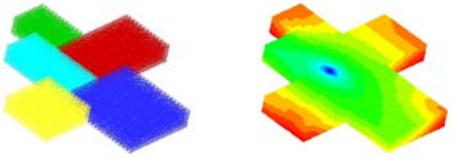
The CCA (Common Component Architecture) and GrACE application at bottom-left investigates the direct numerical simulation of flames with detailed chemistry solving the Navier-Stokes and species evolution equations without approximations. The figure shows this simulation for a mixture of H_2 and Air in stoichiometric proportions, with 3 hot spots at 1000K causing H_2-Air mixture to ignite and create many different radicals. The scientific problems being studied are the flame stabilization mechanisms of unsteady laminar and turbulent flames, with emphasis on the flame structure at the flame base.

The bottom-right application simulates the dynamic response of materials, with the goal to develop a Virtual shock physics Test Facility (VTF) for a wide range of compressive, tensional, and shear loadings, including those produced by detonation of energetic materials. GrACE is the computational engine underlying the VTF. The figure shows the compressible turbulence simulation solving the Richtmyer-Meshkov instability in 3D (RM3D) using adaptive refinements. The Richtmyer-Meshkov instability is a fingering instability that occurs at a material interface accelerated by a shock wave.

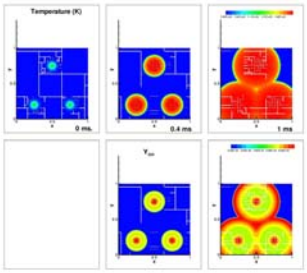
A Selection of SAMR Application Enabled



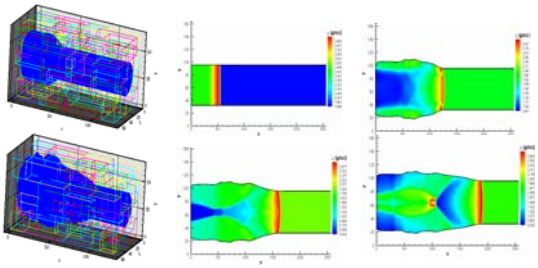
Blast wave in the presence of a uniform magnetic field) – 3 levels of refinement. (Zeus + GrACE + Cactus, P. Li, NCSA, UCSD)



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



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



Richtmyer-Meshkov - detonation in a deforming tube - 3 levels. Z=0 plane visualized on the right
(VTF + GrACE, R. Samtaney, CIT)

Figure 18

ARMADA: AUTONOMIC RUNTIME MANAGEMENT OF DYNAMIC APPLICATIONS

ARMaDA is a framework for the autonomic run-time management and optimization for dynamic SAMR applications. Autonomic behavior is achieved by adapting SAMR application execution to optimize partitioning, load-balancing, and scheduling. Adaptation parameters include the partitioning scheme based on current runtime state (GrACE, Vampire, etc.), granularity/patch size affecting load balance and overhead, dynamic allocation of processors (from beginning or "on-demand"). Other optimizations include hierarchical decomposition using dynamic processor groups, communication optimization, latency tolerance, multithreading, etc.

Autonomic application management involves system-sensitive and application-sensitive adaptation. System-sensitive application management uses current and predicted system state characterization to make application adaptation decisions. For example, the information about the current load and available memory may determine the granularity of the mapping of the application components to the processing nodes, while the availability and "health" of the computing elements on the grid may determine the nature (refined grid size, aspect ratios, etc.) of refinements to be allowed.

Application sensitive adaptations use the current state of the application to drive the run-time adaptations. The abstraction and characterization of the application state is used to drive the resource allocations, partitioning and mapping of application components onto the grid, selection of partitioning and load-balancing algorithms and their configurations, communication mechanisms, etc.

ARMaDA: Autonomic Run-time Management and Optimization for Dynamic (SAMR) Applications

- Partitioning, load-balancing and scheduling of SAMR applications.
  - *Partitioning Scheme*
    - *"Best" partitioning based on application/system configuration and current application/system state*
      - *G-MISP+SP, pBD-ISP, SFC (Vampire, GrACE, Zoltan, ParMetis, …)*
  - *Granularity*
    - *patch size, AMR efficiency, comm./comp. ratio, overhead, node-performance, load-balance, …*
  - *Number of processors/Load per processor*
    - *Dynamic allocations/configuration/management*
      - *1000+ processor from the beginning or "on-demand"*
  - *Hierarchical decomposition using dynamics processor groups*
  - *Communication optimizations/latency tolerance/multithreading*
  - *Availability, capabilities, and state of system resources*
    - *SNMP, NWS*

Figure 19

ARMADA: AUTONOMIC RUNTIME MANAGEMENT

Starting in the upper-left of the figure, the SAMR application is monitored by the V-Grid Monitoring Engine to enable the V-Grid Planning and Analysis Engines to identify natural regions and characterize application state. Simultaneously, the V-Grid Monitoring Engine also monitors and characterizes the system. The synthesized system capability combines monitored information with history and predictive models. Both of these characterizations flow into the V-Grid Analysis and Execution Engines. The V-Grid Analysis Engine deduces objective functions, strategies, and normalized work and resource metrics, using policies and constraints to navigate the decision space. The V-Grid Execution engine uses this information to autonomically partition or repartition the application into VCUs that are mapped and scheduled onto VRUs. Global-Grid Scheduling (GGS) is first used across VRUs and then Local-Grid Scheduling (LGS) within a VRU. The V-Grid Execution Engine then allocates and configures Grid resources and schedules execution of VRUs. This execution is, in turn, is monitored by the monitoring engine. This flow of events occurs within a distributed framework.

A dynamic topology of V-Grid framework agents will locally monitor the application and resources. Changes in the local natural regions will be monitored along with changes in the local resource performance. The V-Grid Analysis Engine may be able to make many local decisions, but may also be able to make improved decisions by "comparing notes" with neighboring framework agents. The autonomic partitioning and scheduling may move work among agents or may acquire new resources and add new agents to the framework.

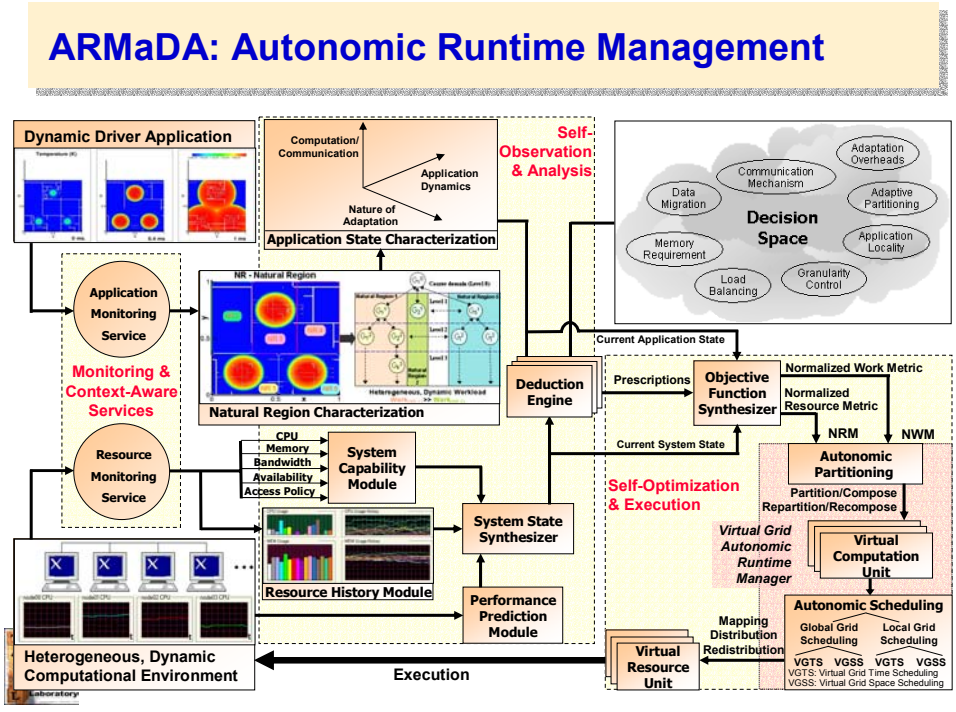


Figure 20

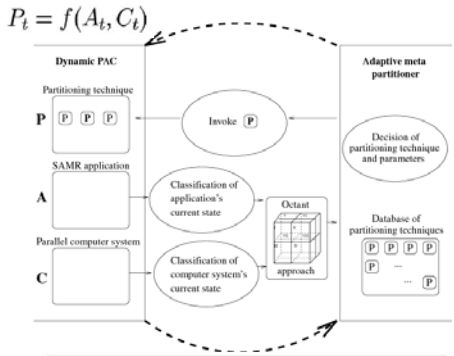
ARMADA: APPLICATION-SENSITIVE ADAPTATIONS

The ARMaDA framework performs adaptive application-sensitive partitioning based on the input parameters and the application's current runtime state. Partitioning behavior is characterized based on the {Partitioner, Application, Computer System} (PAC) tuple. Each PAC tuple is evaluated using a 5-component metric that includes load imbalance, communication requirement, amount of data migration, partitioning induced overhead, and the partitioning time. The PAC relationship is dynamic and the partitioner P is a function of the state of the application A and the computer system C at that time. The octant approach is used to classify application runtime state with respect to the adaptation pattern, computations/communications, and activity dynamics.

The ARMaDA framework has three components: application state monitoring and characterization, partitioner repository and policy engine, and an adaptation component. The state characterization component implements mechanisms that abstract the current application state in terms of the computation/communication requirements, application dynamics, and the nature of the adaptation. The policy engine provides an association for mapping octants to partitioners and the partitioning repository includes a selection from popular software tools such as GrACE (ISP) and Vampire (pBD-ISP, GMISP+SP). Subsequently, the meta-partitioner or adaptation component dynamically selects the appropriate partitioner at runtime and configures it with associated parameters such as granularity. As shown in the slide, experimental results demonstrate the improvement in SAMR application execution using application-sensitive partitioning – 26.19% for VectorWave-2D application on 32 processors on Linux Beowulf cluster "Frea" and 38.28% for RM3D application on 64 processors on IBM SP2 "Blue Horizon".

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"

Partitioner	Execution time (sec)
SFC	637.478
G-MISP+SP	611.749
pBD-ISP	592.05
ARMaDA with SFC start	470.531

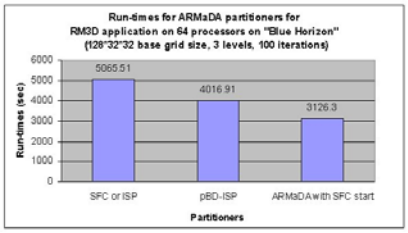


Figure 21

ARMADA: SYSTEM-SENSITIVE ADAPTATIONS

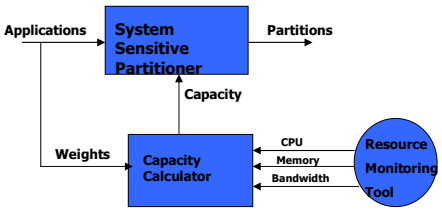
The ARMaDA framework reacts to system capabilities and current system state to select and tune distribution parameters by dynamically partitioning and load balancing the SAMR application grid hierarchy. Current system state is obtained at runtime using the Network Weather Service (NWS) resource monitoring tool. NWS measurements include CPU availability, end-to-end network bandwidth, free memory, and the amount of space unused on a disk. System state information along with system capabilities are then used to compute the relative capacity of each computational node as a weighted sum of the normalized system metric. The weights are application dependent and reflect its computational, memory, and communication requirements. These relative capacities are used by the "system-sensitive" partitioner for dynamic distribution and load-balancing.

The system-sensitive partitioner is evaluated using the RM3D CFD kernel on a 32-node Linux-based workstation cluster. The kernel used 3 levels of factor 2 space-time refinements on a base mesh of size 128*32*32. System-sensitive partitioning reduced execution time by about 18% in the case of 32 nodes. The table in the slide illustrates the effect of sensing frequency on overall application performance. Dynamic runtime sensing improves application performance by as much as 45% compared to sensing only once at the beginning of the simulation. In this experimental setup, the best application performance was achieved for a sensing frequency of 20 iterations.

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)*

Procs	Dynamic Sensing (s)	Static Sensing (s)
2	423.7	805.5
4	292	450
6	272	424
8	225	430



$$C_k = w_p P_k + w_m M_k + w_b B_k$$

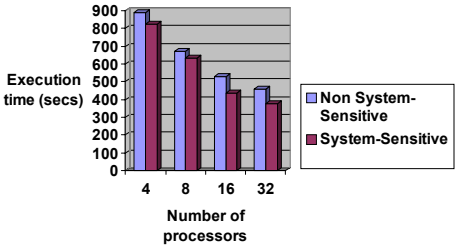


Figure 22

ARMADA: PROACTIVE MANAGEMENT

The ARMaDA framework uses performance prediction functions to estimate execution time and application performance. Performance Functions (PF) describe the behavior of a system component, subsystem or compound system in terms of changes in one or more of its attributes. The PFs of each resource used by an application can be composed to generate an overall end-to-end PF that quantifies application performance.

Performance functions model the application execution time for SAMR-based RM3D and describe overall behavior with respect to the computational load metric on the machine of choice (such as IBM SP "Seaborg" and Linux Beowulf "Discover"). The evaluation on IBM SP yields 2 PFs for small loads (≤30,000 work units) and large loads (>30,000 units) respectively, whereas the Linux Beowulf produces a single PF. The error in modeling the execution time is low – 0-8% for IBM SP and 0-6% for Linux Beowulf.

The PF modeling approach is used by the ARMaDA framework to determine when the benefits of dynamic load redistribution exceed the costs of repartitioning and data movement (if workload imbalance exceeds a certain threshold). A threshold of 0 indicates regular periodic load redistribution while a high threshold represents the ability of the application hierarchy to tolerate workload imbalance. The RM3D evaluation on 8 processors on Linux Beowulf cluster analyzes the effect of dynamic load redistribution on application recompose time for redistribution thresholds of 0 and 1. The application uses 3 refinement levels on a base mesh of size 64*16*16 with regriding every 4 steps. Threshold of 1 considers the costs of redistributing load and results in recompose time being reduced by half (improvement of almost 100%) as compared to a threshold of 0.

ARMaDA: Proactive Management

- Performance Function (PF) – behavior in terms of attribute changes
- "Computational load" metric to model RM3D execution time
- IBM SP "Seaborg" (NERSC)
  - *PF_s – small loads (≤ 30000 units), PF_h – large loads (> 30000)*
  - *Error in modeling execution time is low (0 – 8%)*
- Linux Beowulf "Discover" (Rutgers)
  - *Single PF*
  - *Error in modeling execution time is low (0 – 6%)*

$$PF = \sum_{i=0}^{10} b_i * x^i$$

- Dynamic load redistribution for RM3D & effect on "recompose" time
  - *8 processors, base mesh 64*16*16, 3 levels of factor 2 refinements*
  - *Redistribution thresholds of 0 and 1*
  - *Thresh=1 improves recompose time by 100% compared to thresh=0*

Figure 23

AUTONOMIC OIL WELL PLACEMENT

The goal of this application is to dynamically optimize the placement and configuration of oil wells to maximize revenue. The peer components involved include:

1. Integrated Parallel Accurate Reservoir Simulator (IPARS) providing sophisticated simulation components that encapsulate complex mathematical models of the physical interaction in the subsurface, and execute on distributed computing systems on the Grid.
2. IPARS Factory responsible for configuring IPARS simulations, executing them on resources on the Grid and managing their execution.
3. Very Fast Simulated Annealing (VFSA) optimization service based on statistical physics and the analogy between the model parameters of an optimization problem and particles in an idealized physical system.
4. Economic Modeling Service that uses IPARS simulation outputs and current market parameters (oil prices, costs, etc.) to compute estimated revenues for a particular reservoir configuration.
5. Discover Middleware that integrates Globus Grid services (GSI, MDS, GRAM, and GASS), via the CORBACog, and Discover remote monitoring, interactive steering, and collaboration services, and enables resource discovery, resource allocation, job scheduling, job interaction and user collaboration on the Grid.
6. Discover Collaborative Portals providing experts (scientists, engineers) with collaborative access to other peer components. Using these portals, experts can discover and allocate resources, configure and launch peers, and monitor, interact with, and steer peer execution. The portals provide a shared workspace and encapsulate collaboration tools such as Chat and Whiteboard.

(This slide is courtesy M. Peszynska)

Autonomic Oil Well Placement

- 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 Eval(y,z,P)(T)*
- Normalize the objective function NEval(y,z,P) so that:

$$\min Neval(y,z,P) \Leftrightarrow \max Eval(y,z,P)$$

Figure 24

These peer entities involved in the optimization process need to dynamically discover and interact with one another as peers to achieve the overall application objectives. The experts use the portals to interact with the Discover middleware and the Globus Grid services to discover and allocate appropriate resource, and to deploy the IPARS Factory, VFSA and Economic model peers ((1)). The IPARS Factory discovers and interacts with the VFSA service peer to configure and initialize it ((2)). The expert interacts with the IPARS Factory and VFSA to define application configuration parameters ((3)). The IPARS Factory then interacts with the Discover middleware to discover and allocate resources and to configure and execute IPARS simulations ((4)).

The IPARS simulation now interacts with the Economic model to determine current revenues, and discovers and interacts with the VFSA service when it needs optimization ((5)). VFSA provides IPARS Factory with optimized well information ((6)), which then launches new IPARS simulations ((7)). Experts at anytime can discover and collaboratively monitor and interactively steer IPARS simulations, configure the other services and drive the scientific discovery process ((8)). Once the optimal well parameters are determined, the IPARS Factory configures and deploys a production IPARS run.

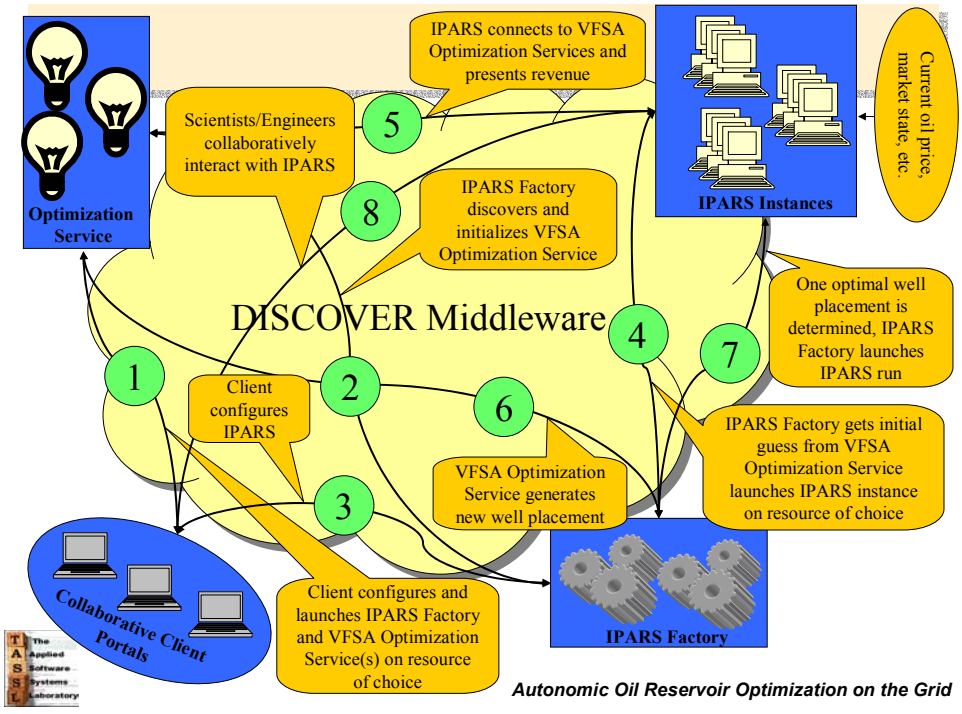


Figure 25

AUTONOMIC OIL WELL PLACEMENT

The figure below show results from the autonomic oil well placement applications. It shows that the process converges to the optimal placement in 20 iterations.
(This slide is courtesy M. Peszynska)

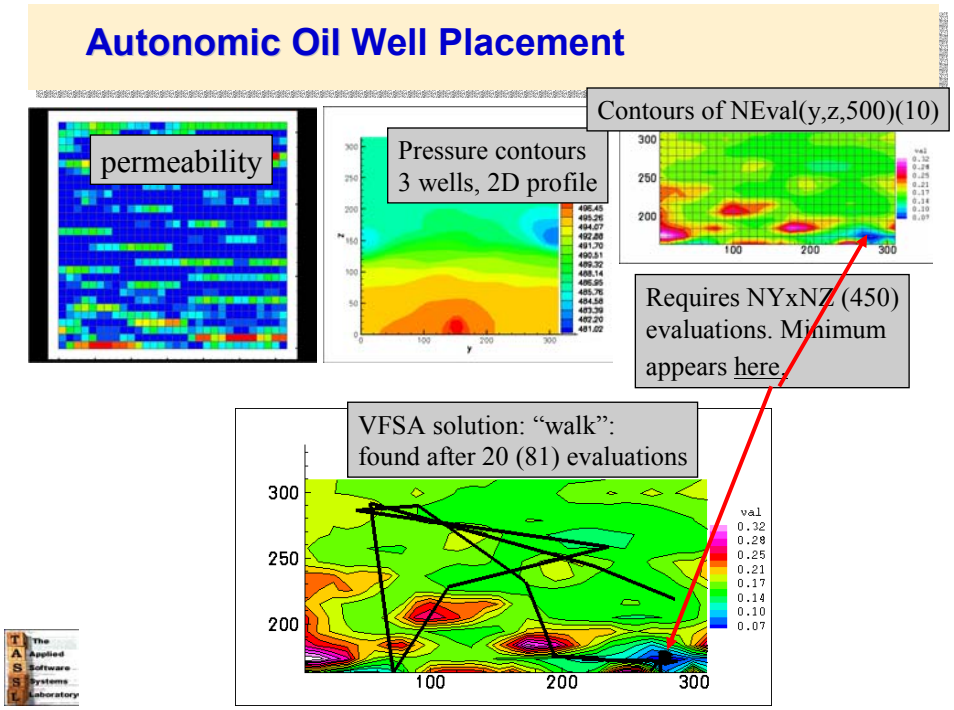


Figure 26

CONCLUSION

The computational solutions addressed by the AutoMate project are based on fundamental innovations in the development, optimization and deployment of component-based Grid applications, thereby allowing the heterogeneity and dynamics of the applications to match that of the Grid and fully exploit its potential. These innovations will enable scientists to choreograph high performance, integrated end-to-end simulations that were never possible or attempted before. The key IT contributions are the methodology and associated technologies that enable the development of applications that can manage and exploit the dynamism and heterogeneity of the Grid, and that address the extremely serious problem of software complexity that is threatening both academia and industry.

We currently have working prototypes of each of the components presented in this paper, and are in the process of integrating them to support autonomic structured adaptive mesh refinement applications (SAMR) in science and engineering. Further information about AutoMate and its components can be obtained from http://automate.rutgers.edu.

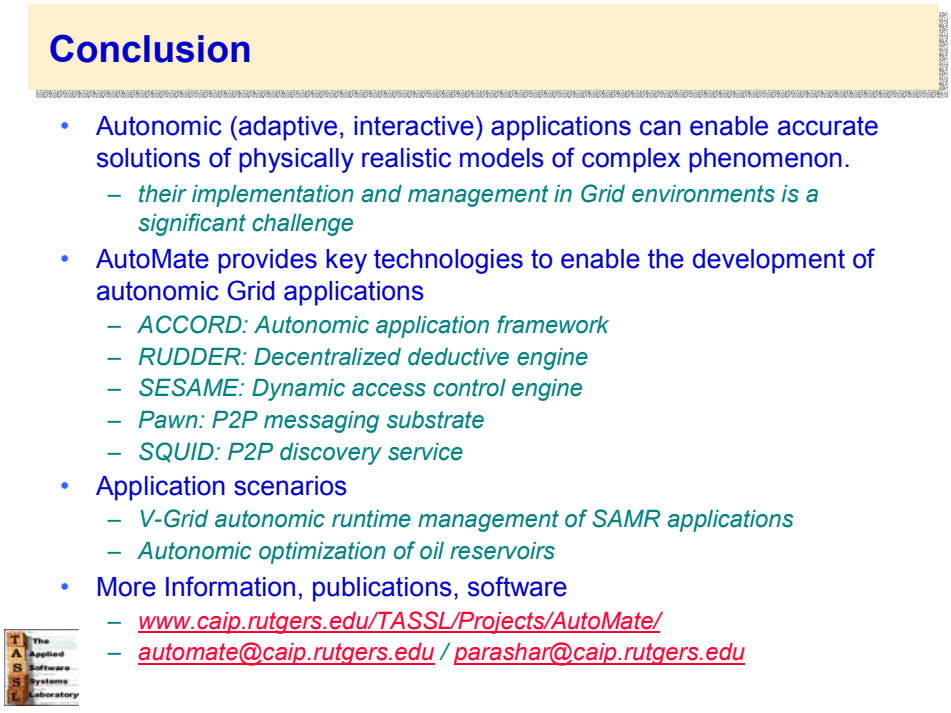


Figure 27