Improving a Distributed Software System's Quality of Service via Architecture-Driven Dynamic Redeployment

Nenad Medvidovic

Center for Systems and Software Engineering
Computer Science Department
Viterbi School of Engineering
University of Southern California
Los Angeles, USA

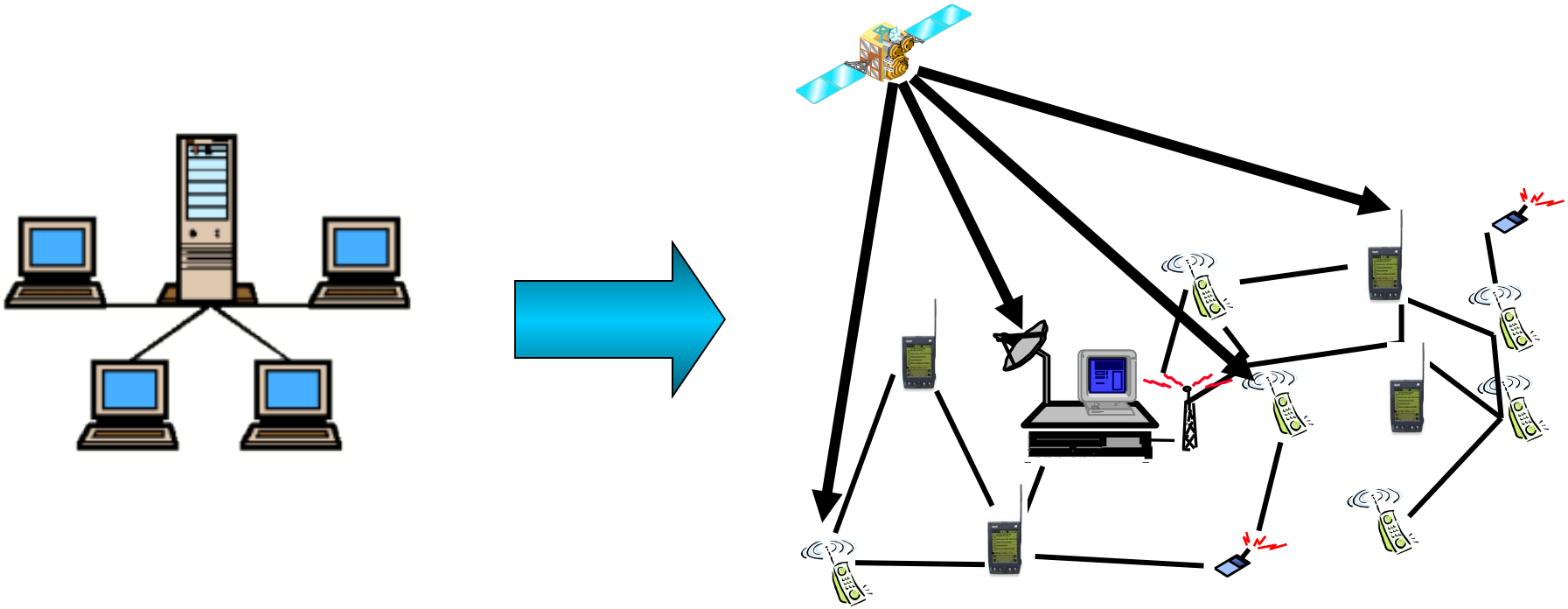
neno@usc.edu
http://sunset.usc.edu/~neno/

Outline

- Problem
  - Motivation
  - Approach
  - Contribution
- Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
- Runtime support
  - Prism-MW
  - Integration
- Collaborations and future work

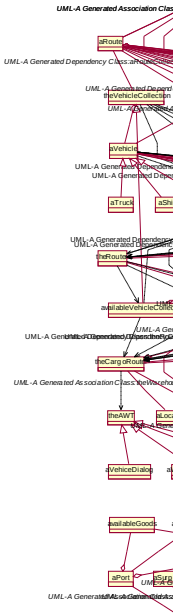
Software Engineering Trends

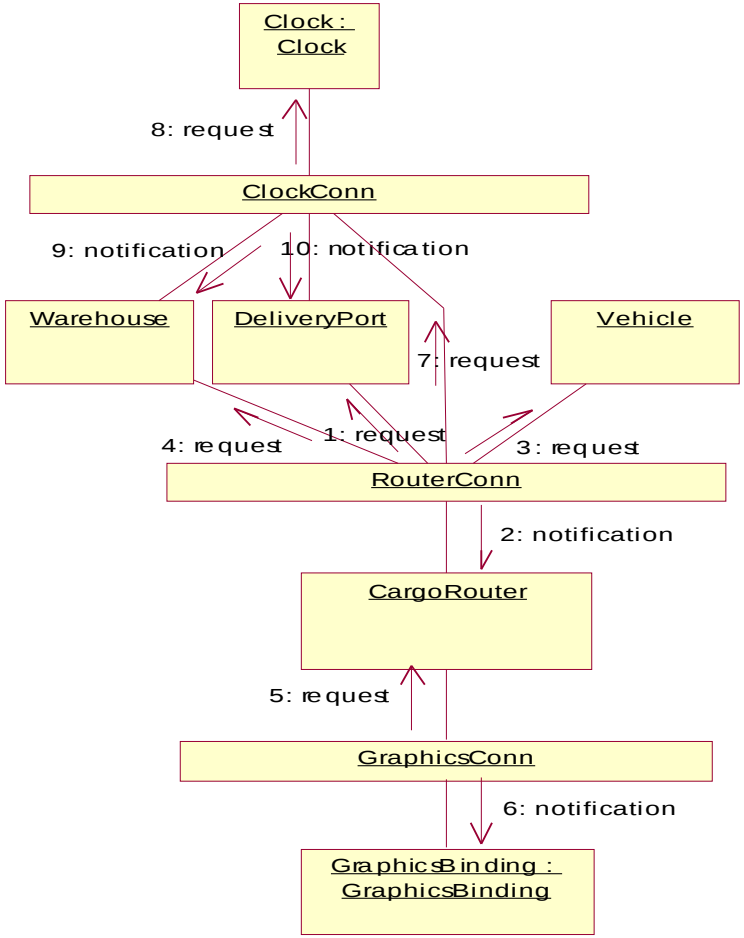
- Software systems are becoming more complex
- A wider spectrum of distribution and heterogeneity
- Hardware and software mobility is becoming the norm



Software Architecture

- Software Architecture
  - A high-level model of a system
  - Represents system organization
    - Components
    - Connectors
    - Events
    - Configurations

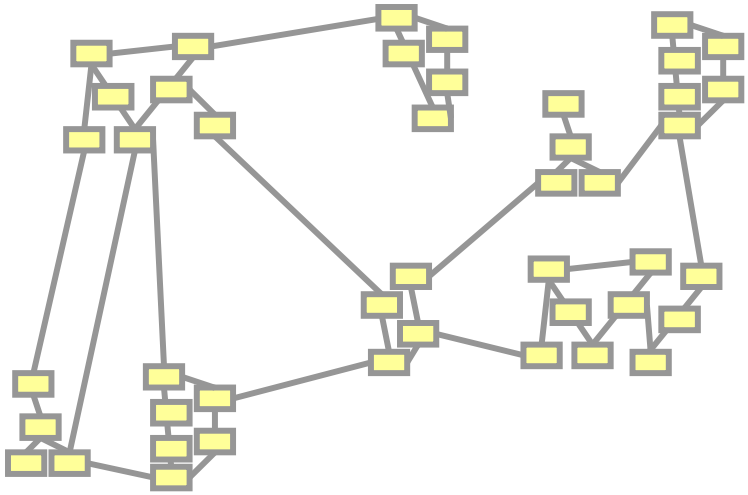




Architectural Decisions

- Architectural decisions impact non-functional properties of the system
- Non-functional property
  - Quality level at which an expected functionality is delivered
  - a.k.a. Quality of Service (QoS)
- Making optimal architectural decisions has remained an art form
  - Lack of quantification and measurement techniques
  - Reliance on domain expert knowledge

Deployment Architecture Impacts QoS

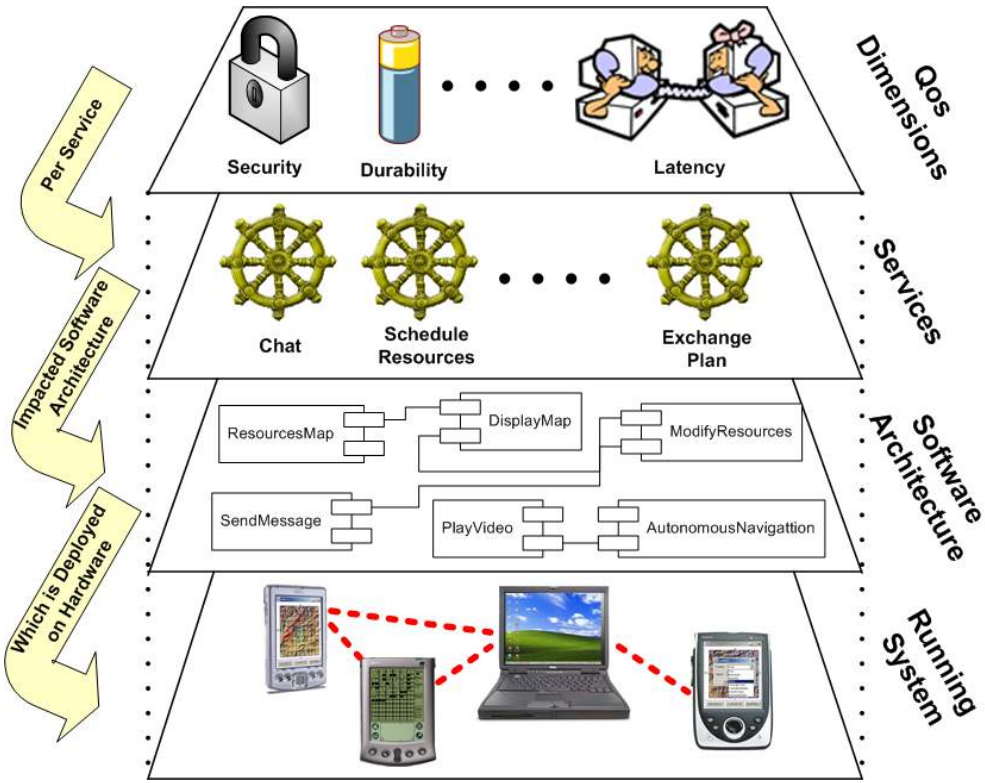


- *Deployment Architecture*: allocation of software components to hardware hosts
- h^c deployment architectures are possible for a given system
  - Many provide the same functionality
  - but different qualities of service (QoS)

Problem in a Nutshell

- **Motivating Question**
  - **How could we find and effect a deployment architecture that improves (maximizes) multiple QoS dimensions?**
    - Where other possible solutions such as caching, hoarding, replication, etc. are not appropriate or ideal

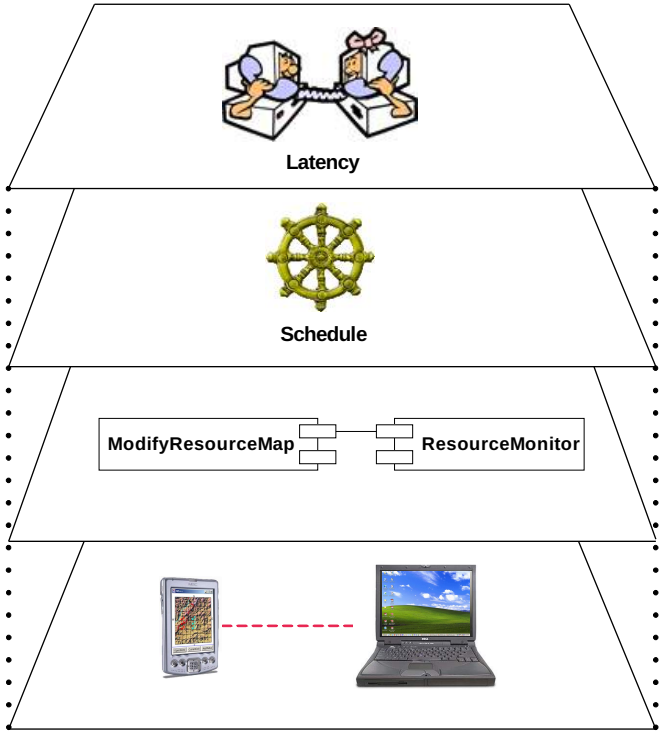
- **Objective**
  - **Devise a solution that is applicable to many classes of application scenarios**
    - No particular definition of QoS dimensions, class of systems, etc.

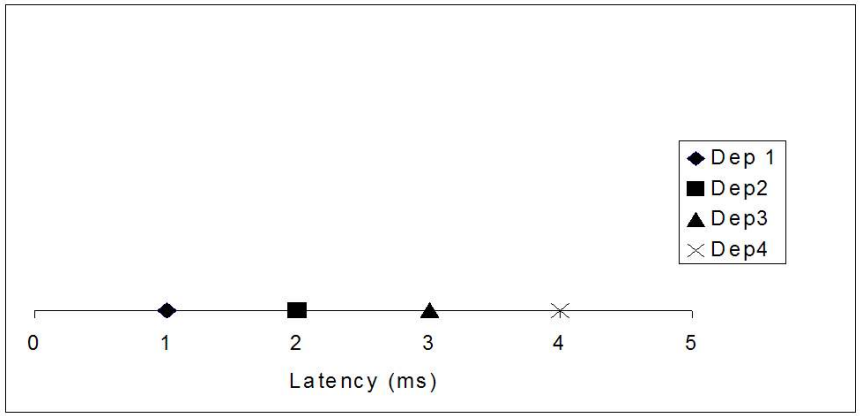


Outline

- Problem
  - Motivation
  - Approach
  - Contribution
- Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
- Runtime support
  - Prism-MW
  - Integration
- Collaborations and future work

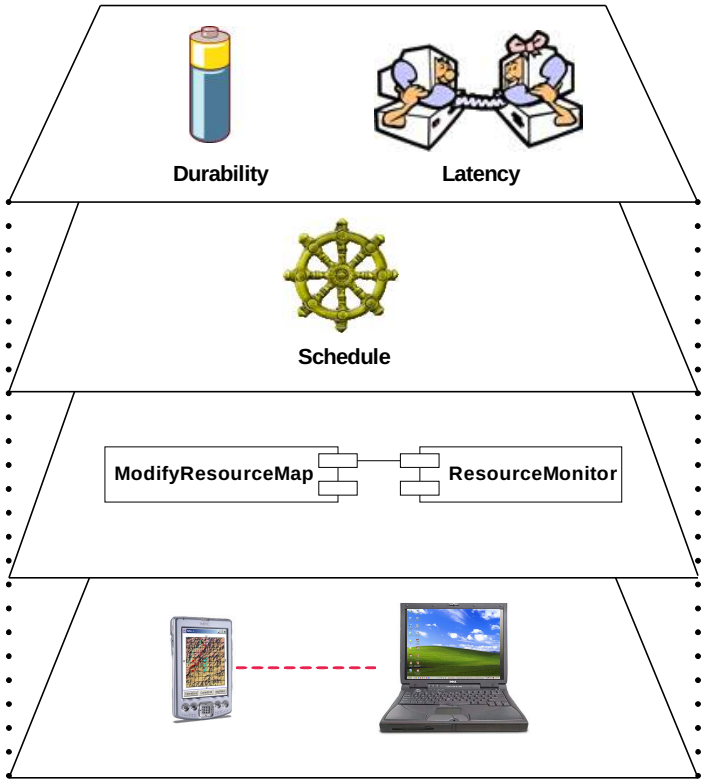
Scenario with a Single QoS Dimension

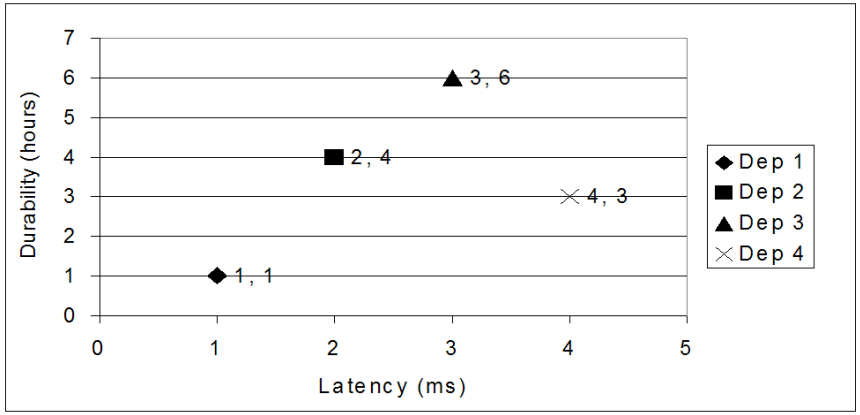




- Objective is to minimize latency
- The *optimal* deployment architecture is deployment 1
- Most all related approaches stop here, but clearly this is not good enough

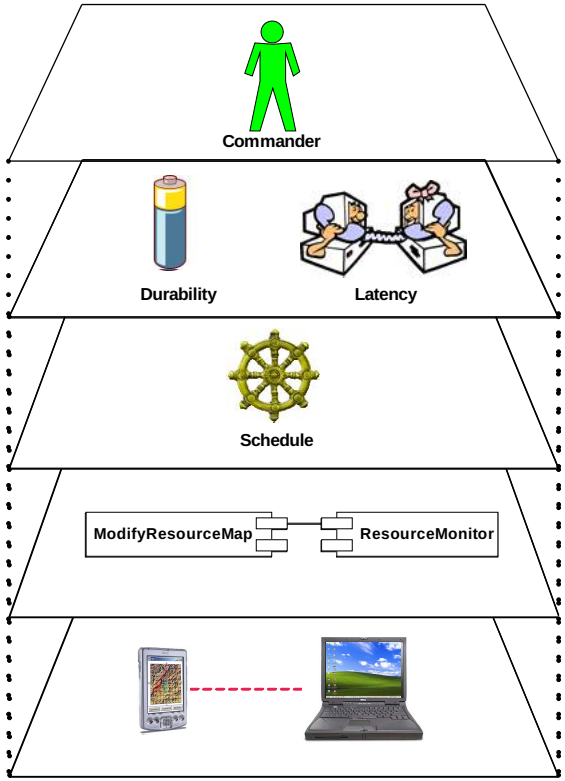
Conflicting QoS Dimensions

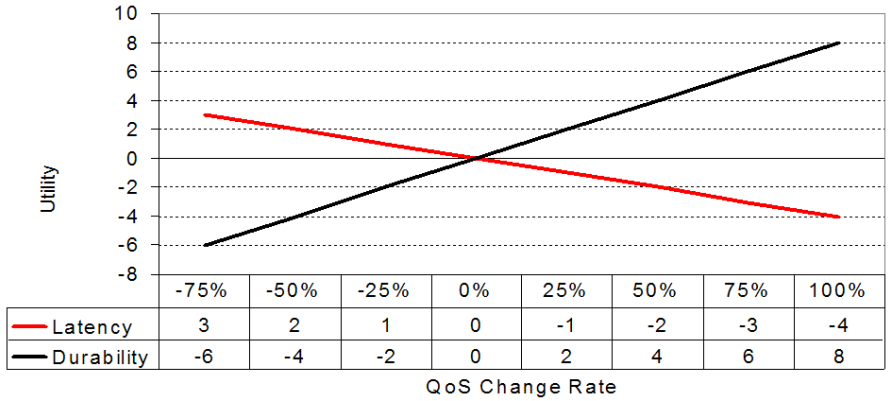




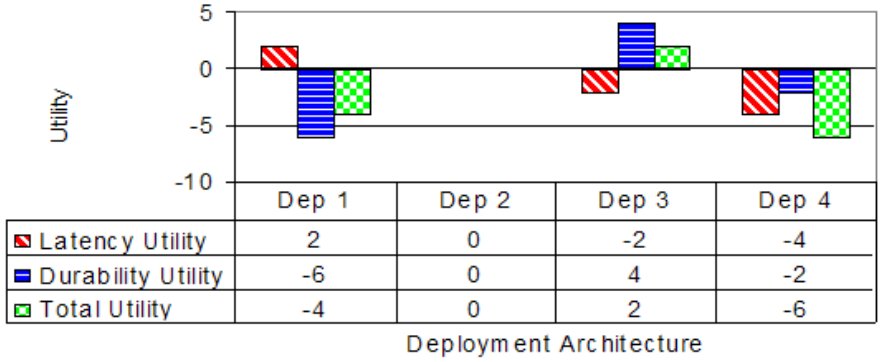
- Objective is to minimize latency and maximize durability
- There is no optimal deployment architecture!
- Phenomenon known as *Pareto Optimal* in multidimensional optimization

Resolving Trade-Offs between QoS Dimensions





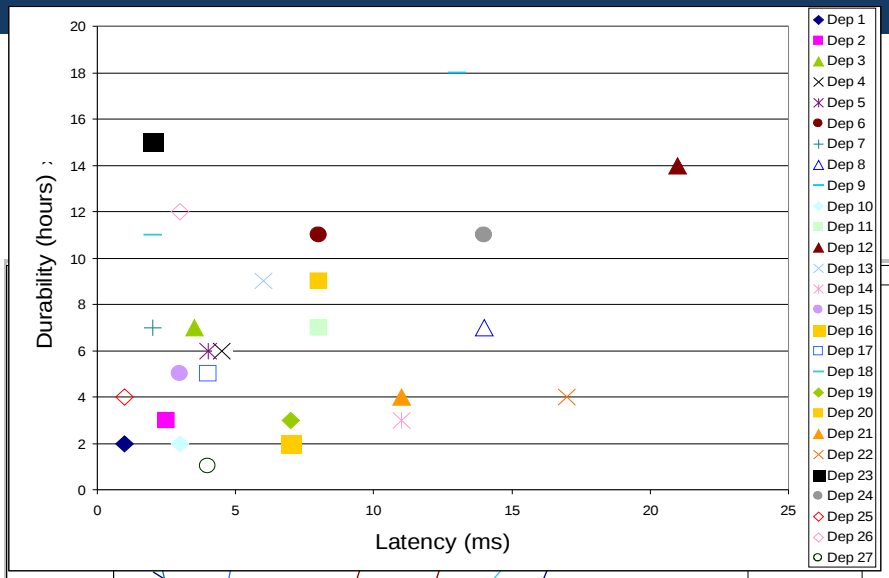
- A *utility function* denotes a user's preferences for a given rate of improvement in a QoS dimension

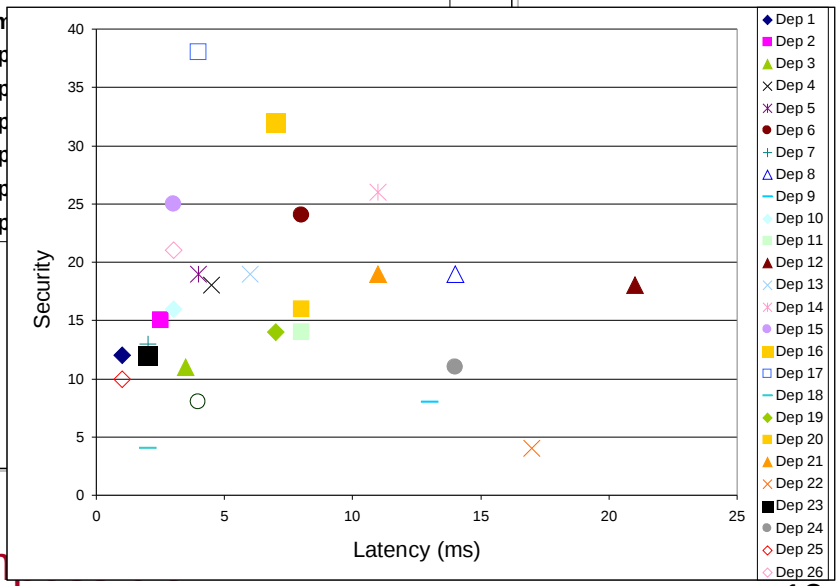


- Allows expression of multidimensional optimization in terms of a single scalar value

- Guiding Insight
  - System users have varying QoS preferences for the system services they access

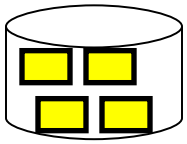
A Slightly Larger Scenario





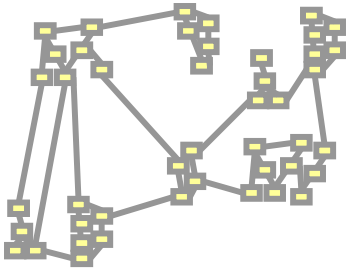
mes im

Overall Approach



Component Repository

Domain Expert



Deployment Model

Runtime Analysis

Repopulate

Software Architect

Redeploy

Monitoring Data

QoS Pref.

Monitor

Hardware System

Users

Outline

- Problem
  - Motivation
  - Approach
  - Outcomes
- Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
- Runtime support
  - Prism-MW
  - Integration
- Collaborations and future work

Outcomes

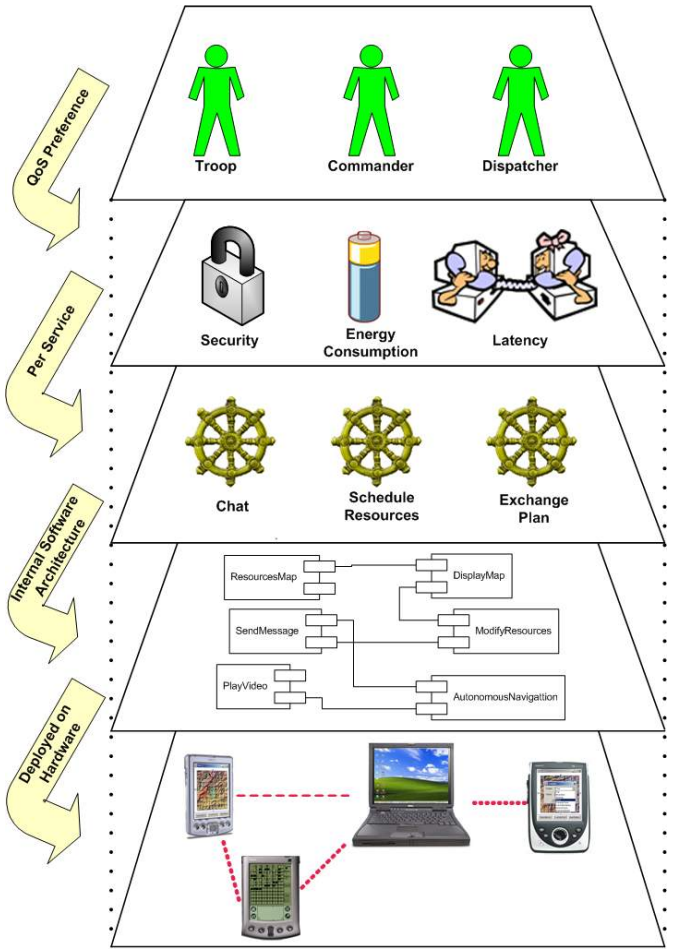
- Consideration of multiple QoS dimensions and constraints
- Consideration of multiple user-level services
- Generic formal modeling and analysis that can be tailored to each application scenario
- Suite of customizable tools
  - Extension points for configuring the tools
  - Promotes reuse and cross-evaluation of solutions to this problem

Outline

- Problem
  - Motivation
  - Approach
  - Outcomes
- Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
- Runtime support
  - Prism-MW
  - Integration
- Collaborations and future work

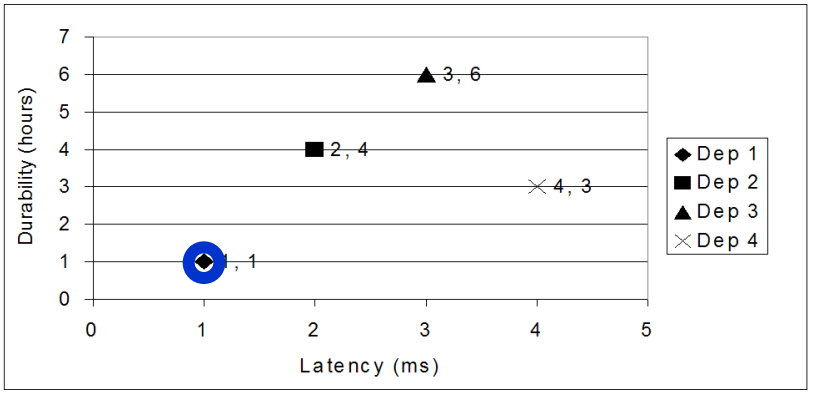
Formal Model of the System

- Define sets that specify the system elements and their properties
  - *Set C of software components*
    - *C = {ResourcesMap, SendMessage, DisplayMap, …}*
  - *Set CP of software component properties*
    - *CP = {size, reliability, …}*
  - *Other sets*
    - *H* of hardware nodes, N of network links, I of logical links, S of services, Q of QoS dimensions, U of users
    - *HP* of hardware parameters, NP of network link parameters, CP of software component parameters, IP of logical link parameters
- Define functions that quantify system properties
  - Function $cParam:C \times CP \rightarrow R$
    - cParam(ResourcesMap, size) = 150
  - Other functions
    - $hParam:H \times HP \rightarrow R$
    - $nParam:N \times NP \rightarrow R$
    - $lParam:I \times IP \rightarrow R$
    - $sParam:S \times \{H \cup C \cup N \cup I\} \times \{HP \cup CP \cup NP \cup IP\} \rightarrow R$

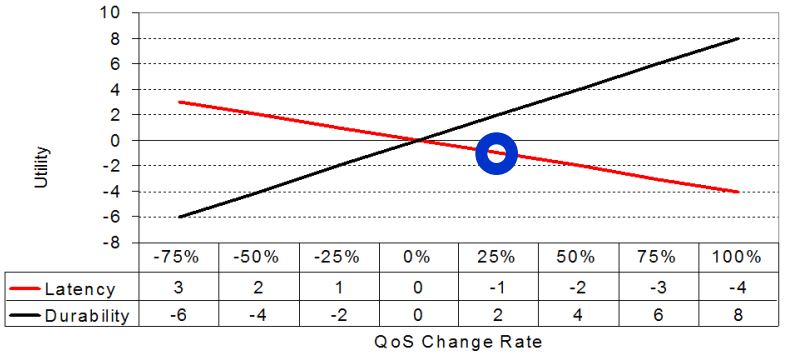


Formal Model of QoS & Users' Preferences

- Define QoS functions
  - *qValue:S×Q×DepSpace → R*
    - *quantifies the achieved level of QoS given a deployment*
    - *qValue(Schedule, Latency, Dep 1) = 1ms*



- Define users' preferences in terms of utility
  - *qUtil:U×S×Q×R → [MinUtil,MaxUtil]*
    - *represents the accrued utility for a given rate of change*
    - *qUtil(Commander, Schedule, Latency, 0.25) = -1*



	-75%	-50%	-25%	0%	25%	50%	75%	100%
Latency	3	2	1	0	-1	-2	-3	-4
Durability	-6	-4	-2	0	2	4	6	8

Model of the Constraints

- A set PC of parameter constraints
  - *PC={memory, bandwidth,…}*
- A function *pcSatisfied:PC×DepSpace → [0,1]*
  - 1 if constraint is satisfied
  - 0 if constraint is not satisfied
- Functions that restrict locations of software components
  - *loc:C×H → [0,1]*
    - *loc(c,h)=1 if c can be deployed on h*
    - *loc(c,h)=0 if c cannot be deployed on h*
  - *colloc:C×C → [-1,1]*
    - *colloc(c1,c2)=1 if c1 has to be on the same host as c2*
    - *colloc(c1,c2)=-1 if c1 cannot be on the same host as c2*
    - *colloc(c1,c2)=0 if there are no restrictions*

Problem Definition

Given the current deployment *d*, find an improved deployment *d'* such that the users' overall utility defined as the function

$$overallUtil(d,d') = \sum_{u=1}^{|U|} \sum_{s=1}^{|S|} \sum_{q=1}^{|Q|} qUtil(u,s,q,\Delta q)$$

→ Total utility of changing d to d'

where $\Delta q = \dfrac{qValue(s,q,d') - qValue(s,q,d)}{qValue(s,q,d)}$

→ Impact of changing d to d' on a QoS dimension

is maximized and specific conditions are satisfied:

- $\forall c \in C,\ loc(c,H_c)=1$ → All location constraints are satisfied
- $\forall c1 \in C,\ \forall c2 \in C,\ if\ (colloc(c1,c2)=1) \rightarrow (H_{c1}=H_{c2}),$ $if\ (colloc(c1,c2)=-1) \rightarrow (H_{c1} \neq H_{c2})$ → All collocation constraints are satisfied
- $\forall constr \in PC$ $pcSatisfied(constr,d)=1$ → All system parameter constraints are satisfied

Model Instantiation

- The engineer needs to further refine the “loosely” defined elements of the model
  1. Define the pertinent properties of the application scenario
  2. Define QoS dimensions in terms of system properties

$$qValue(s, availability, d) = \sum_{c1=1}^{C_s} \sum_{c2=1}^{C_s} sParam(s, I_{c1,c2}, freq) * nParam(N_{H_{c1}, H_{c2}}, rel)$$

  3. Define system parameter constraints

- But how is this done practically?
  - Via appropriate tool support

Outline

- Problem
  - Motivation
  - Approach
  - Outcomes
- Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
- Runtime support
  - Prism-MW
  - Integration
- Collaborations and future work

Algorithms

- A survey of applicable approaches to solving various classes of application scenarios resulted in five types of solutions
  - Two solutions represent state-of-the-art off-the-shelf solutions to solving this problem
  - The remaining solutions are special-purpose algorithmic solutions to address the shortcomings of the off-the-shelf solutions

- Why different algorithms?
  - Each algorithm is suitable for a particular class of problems

Two Off-the-shelf Solvers

1. Mixed Integer Non-linear Programming (MINLP)
   - Represent the problem as a set of linear and non-linear constraint functions
   - Two drawbacks
     - Does not guarantee to find the optimal solution
     - In 20% of large problems, it cannot find any solution

2. Mixed Integer Linear Programming (MIP)
   - Transform the MINLP problem to an MIP problem
   - Developed heuristics to decrease the complexity from $O(2^{|H|^2|C|^2}) \rightarrow O(|H|^{|C|})$
   - Pros: finds the optimal solution
   - Cons: it is an exponentially complex approach ➔ infeasible for any realistic system

Three Optimization Algorithms

- Greedy — Polynomial $O(|S|^3(|C||U||Q|)^2)$
  - An iterative algorithm that leverages heuristics for
    - Ranking elements of the problem (e.g., services, hosts, components)
    - Assigning software components to hardware hosts

- Genetic — Linear per generation $O(|S||U||Q|)$
  - An individual represents a solution to the problem
  - Populations of individuals are evolved via cross-overs and mutations

Common Theme

Heuristically make local decisions that maximize the global objective

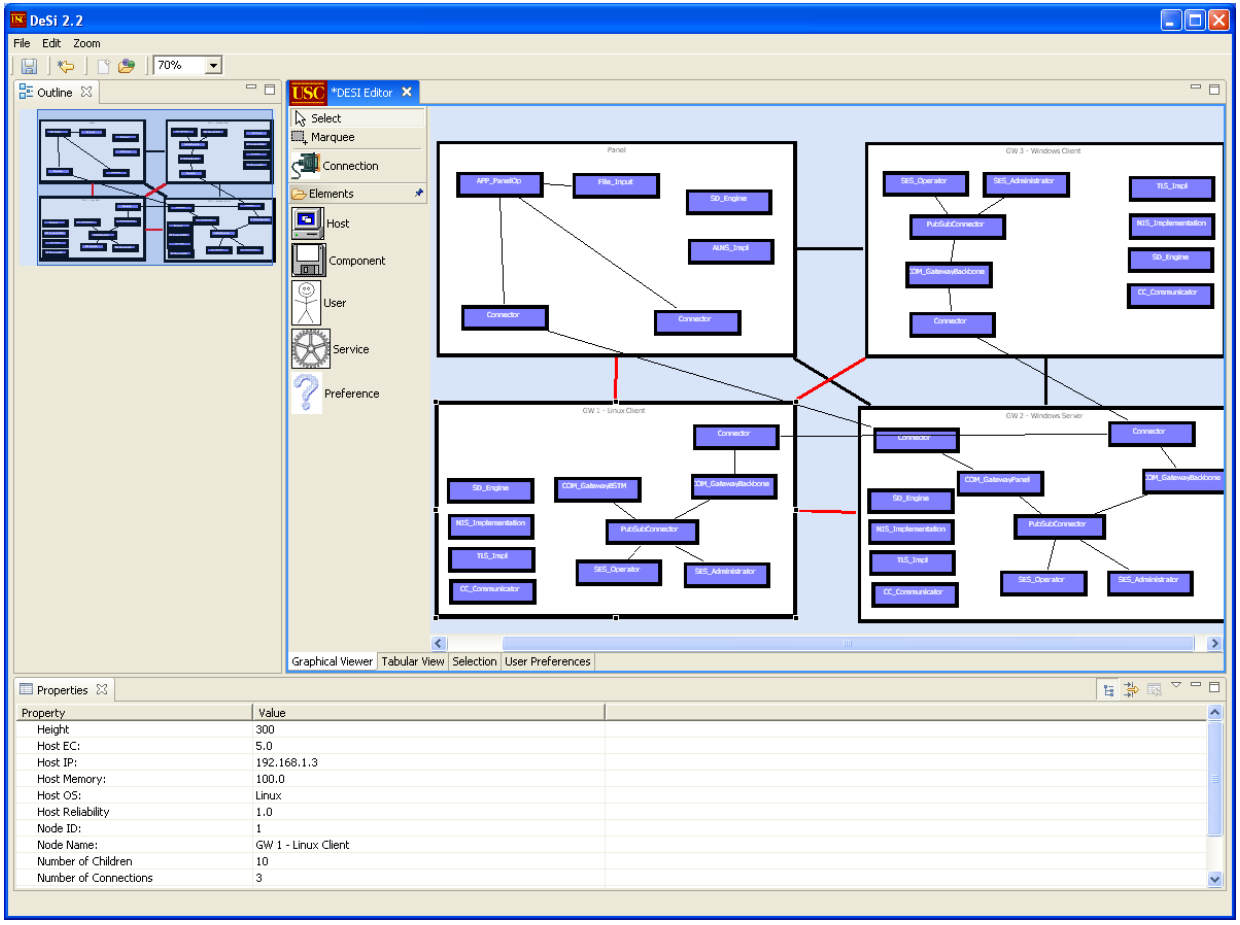
ch

- Market-based — Polynomial $O(|C|^2|S||U||Q|)$
  - Decentralized
  - Autonomous agents on each device auction their local components and bid on each others' components
  - A good mechanism design (the calculation of auctions and bids) improves the global objective

Outline

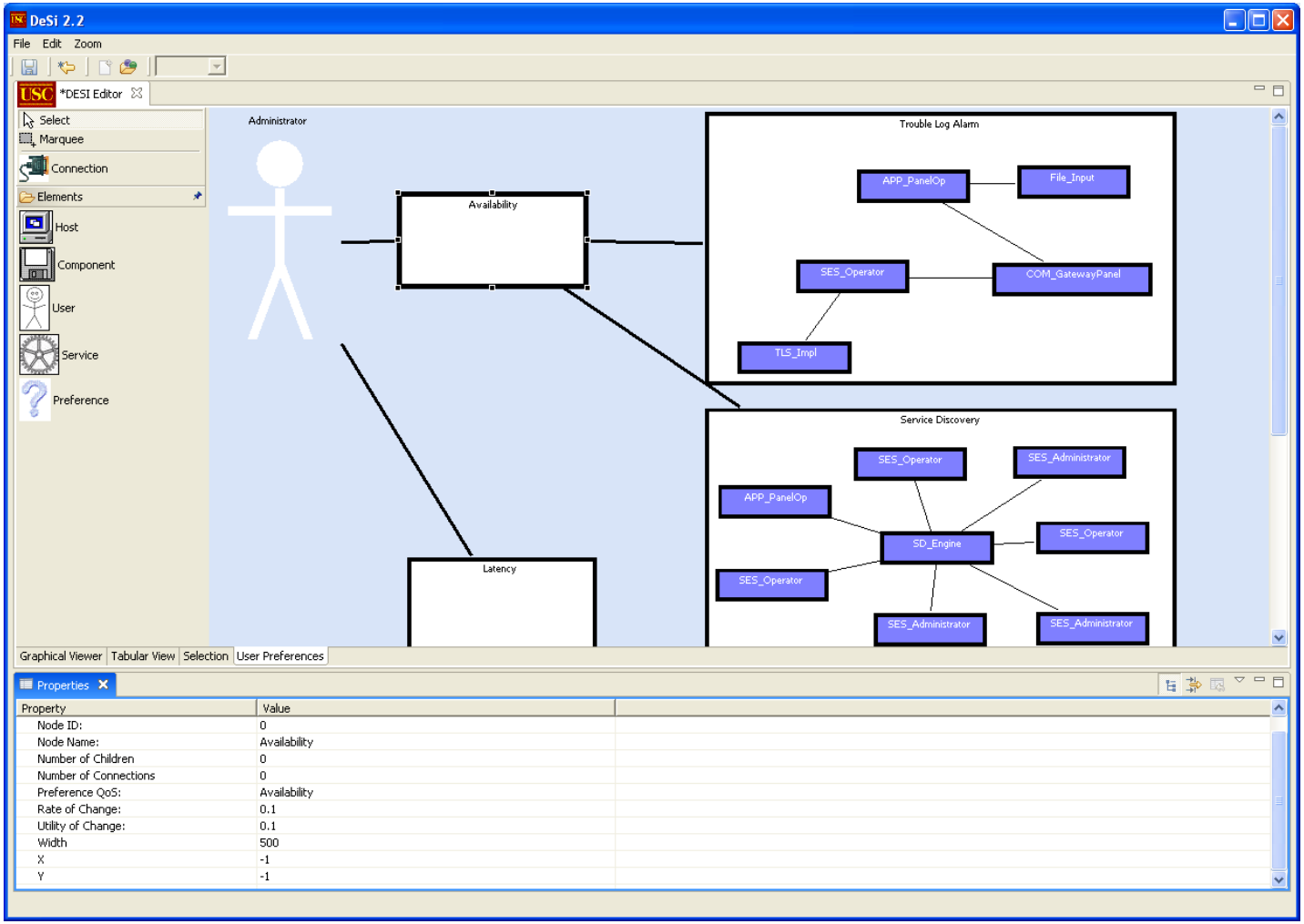
- Problem
  - Motivation
  - Approach
  - Outcomes
- Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
- Runtime support
  - Prism-MW
  - Integration
- Collaborations and future work

DeSi – Deployment Modeling



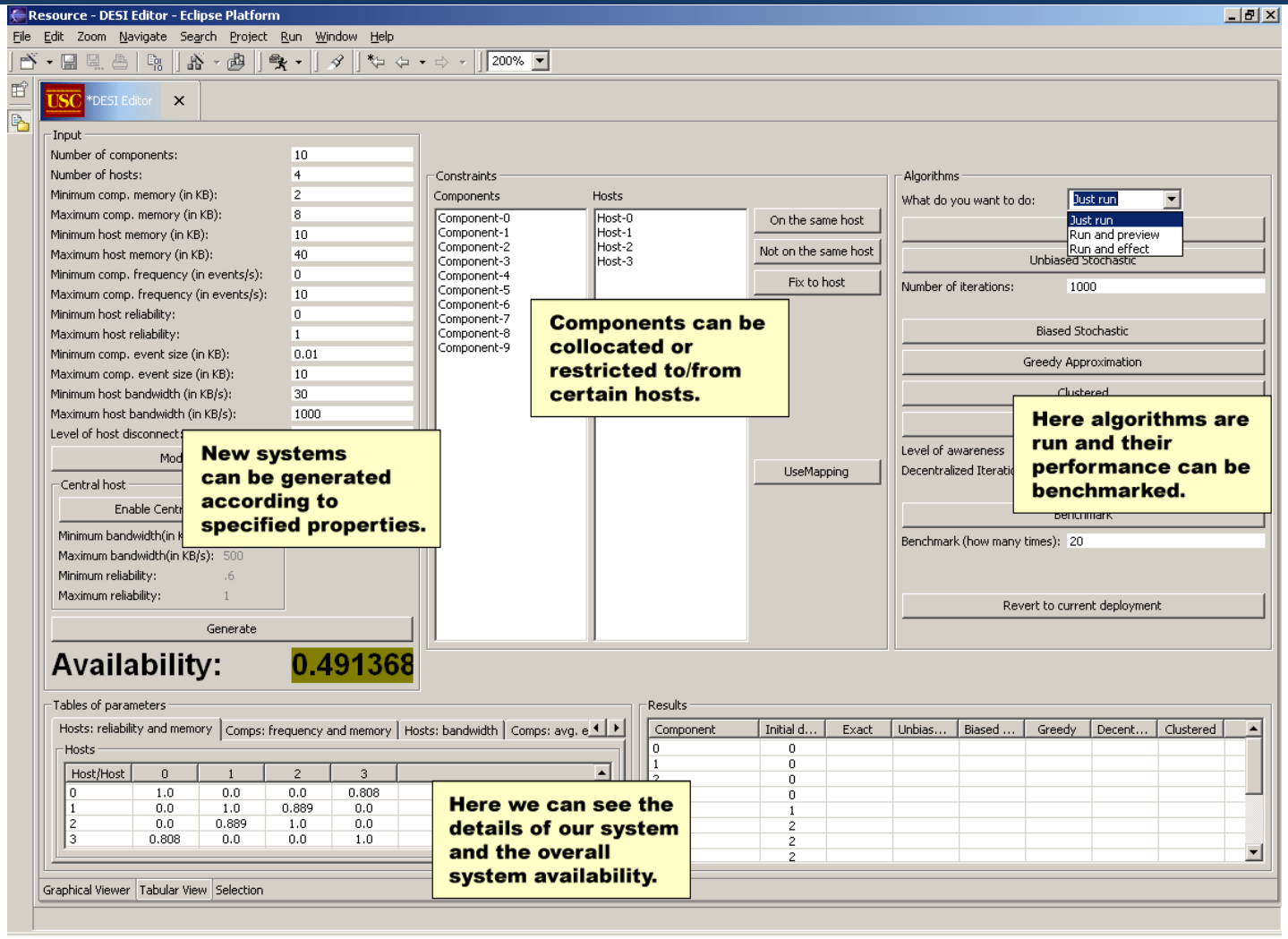
- Provides modeling constructs for this problem
- Arbitrary parameters can be associated with these constructs
- A QoS dimension can be defined as a function of the modeling constructs and their parameters

DeSi – User Preference Modeling



- DeSi's MVC architecture allows for the addition of separate but synchronized views of the underlying model

DeSi – Control Panel



- New algorithms can be plugged into DeSi to analyze and modify its underlying model

Outline

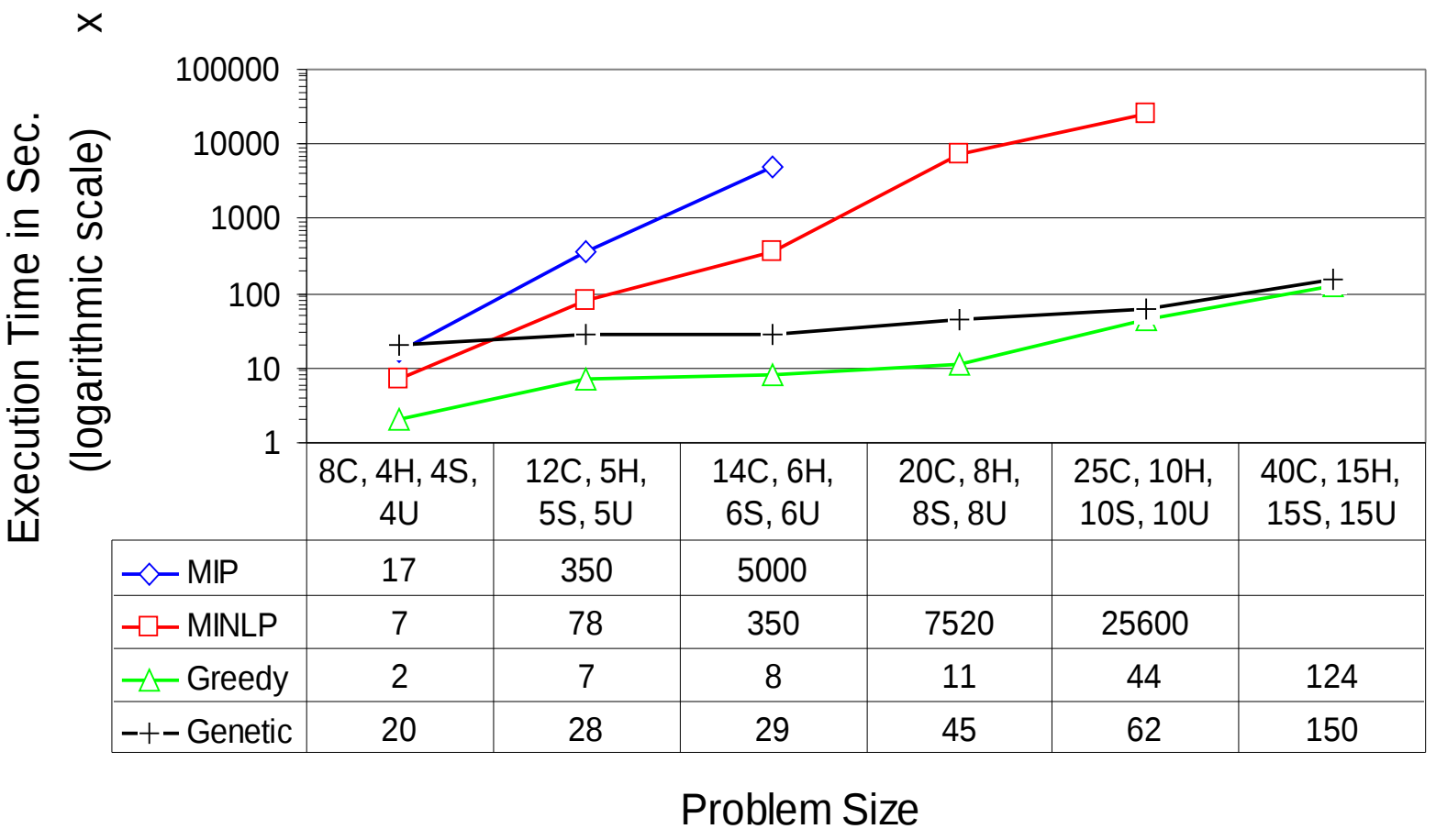
- Problem
  - Motivation
  - Approach
  - Outcomes
- Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
- Runtime support
  - Prism-MW
  - Integration
- Collaborations and future work

Algorithms in Practice

	MIP				MINLP				Greedy				Genetic			
QoS	Avail.	Latency	Comm. Security	Energy Cons.	Avail.	Latency	Comm. Security	Energy Cons.	Avail.	Latency	Comm. Security	Energy Cons.	Avail.	Latency	Comm. Security	Energy Cons.
service 1	56%	-8%	18%	-8%	33%	2%	-5%	14%	24%	-8%	4%	-4%	16%	-2%	18%	-8%
service 2	93%	94%	97%	24%	91%	41%	32%	24%	83%	91%	62%	15%	93%	84%	35%	18%
service 3	39%	30%	22%	49%	32%	38%	11%	69%	39%	30%	22%	49%	19%	30%	22%	49%
service 4	215%	97%	302%	7%	215%	97%	302%	7%	165%	50%	220%	12%	180%	91%	150%	10%
service 5	59%	7%	25%	26%	23%	5%	39%	21%	43%	7%	19%	18%	29%	5%	35%	33%
service 6	99%	55%	37%	44%	83%	35%	45%	32%	99%	55%	37%	44%	99%	55%	37%	44%
service 7	91%	57%	20%	47%	97%	29%	44%	25%	91%	37%	14%	23%	91%	43%	4%	49%
service 8	43%	22%	7%	56%	41%	11%	-5%	72%	32%	21%	-10%	58%	13%	51%	7%	72%
Average	86%	44%	66%	30%	76%	32%	57%	33%	72%	35%	46%	26%	67%	44%	38%	33%
overallUtil	64				57				55				52			

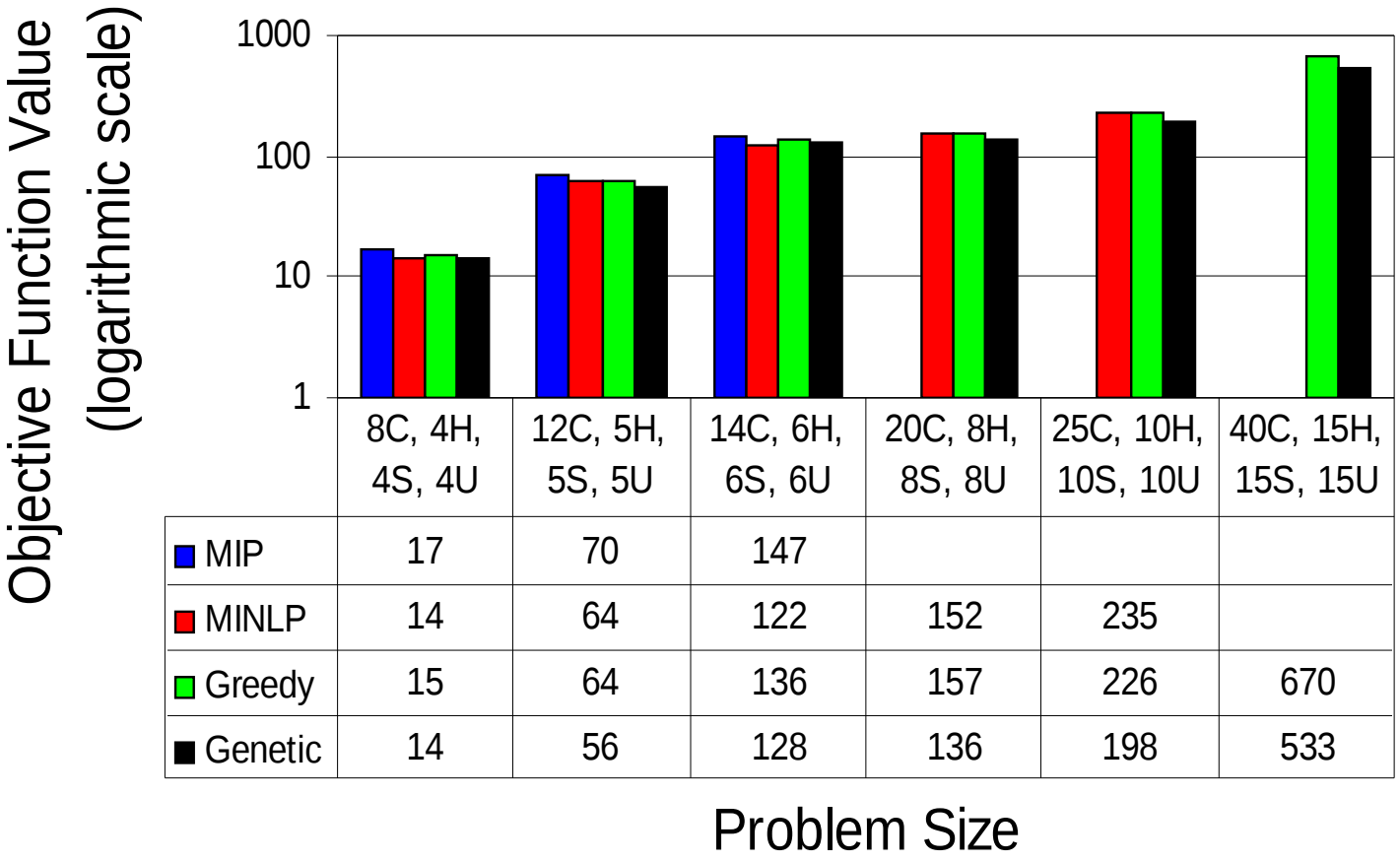
- Results of running the algorithms on an example scenario of 12 Comps, 5 Hosts, 8 Services, and 8 Users
- Significant improvements for all the four QoS dimensions by all the algorithms
- The more important QoS dimensions of services have improved significantly more than others

Algorithms' Performance



	8C, 4H, 4S, 4U	12C, 5H, 5S, 5U	14C, 6H, 6S, 6U	20C, 8H, 8S, 8U	25C, 10H, 10S, 10U	40C, 15H, 15S, 15U
MIP	17	350	5000			
MINLP	7	78	350	7520	25600	
Greedy	2	7	8	11	44	124
Genetic	20	28	29	45	62	150

Algorithms' Accuracy



	8C, 4H, 4S, 4U	12C, 5H, 5S, 5U	14C, 6H, 6S, 6U	20C, 8H, 8S, 8U	25C, 10H, 10S, 10U	40C, 15H, 15S, 15U
MIP	17	70	147			
MINLP	14	64	122	152	235	
Greedy	15	64	136	157	226	670
Genetic	14	56	128	136	198	533

Algorithmic Trade-Offs

- Architectural style
  - MIP for constrained styles (e.g., Client-Server), optimization algorithms for flexible styles (e.g., Peer-to-Peer)
- Number of QoS dimensions
  - Large number of QoS → Genetic outperforms others
- Number of system parameter constraints
  - Large number of constraints → Genetic has poor accuracy
- Stable vs. unstable systems
  - MIP for stable systems, optimization algorithms for unstable systems
- Available resources
  - Resource constrained system → Execute the genetic algorithm in parallel on multiple devices
- Centralized vs. decentralized systems
  - Decentralized system → Market-based algorithm

Outline

- Problem
  - Motivation
  - Approach
  - Outcomes
- Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
- Runtime support
  - Prism-MW
  - Integration
- Collaborations and future work

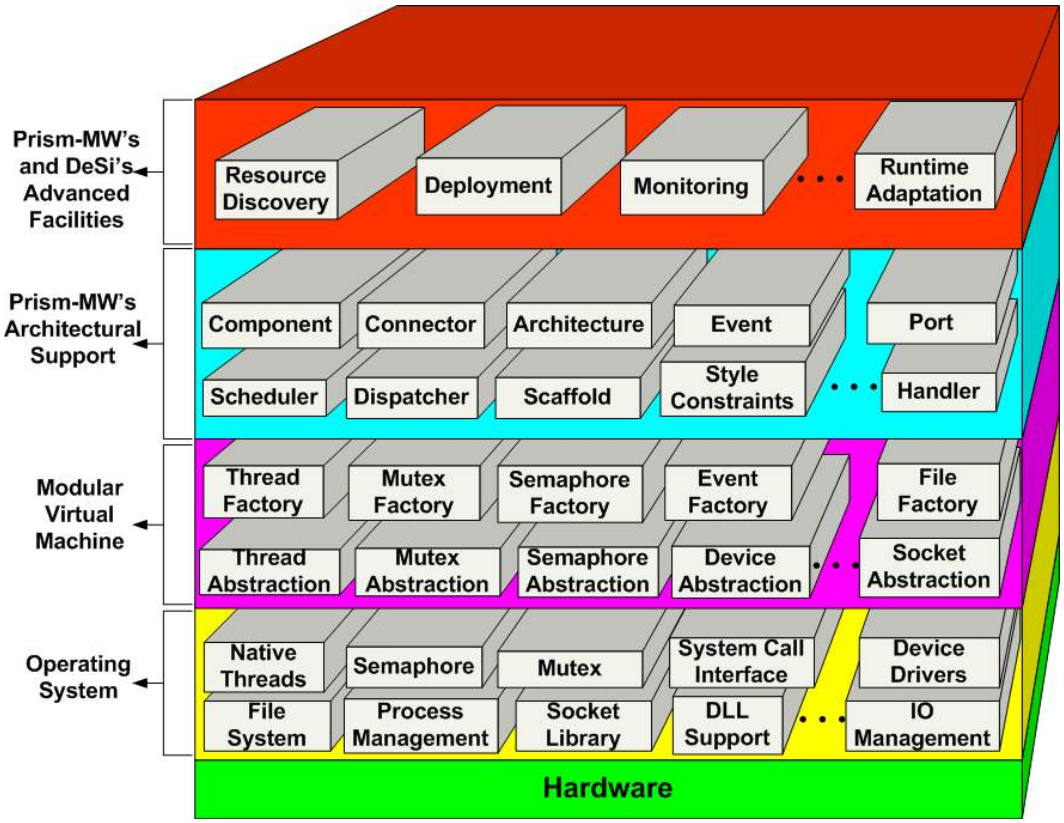
From Deployment Models to Running System

- Architectural models provide *high-level* concepts
  - Components, hosts, links, configurations , etc.
- Software systems are implemented using *low-level* PL constructs
  - Variables, pointers, procedures, objects, etc.
- Deployment is ad-hoc
  - Shell scripts, make files, system commands, etc.
- Bridging the models to runtime (re)deployment is challenging
  - Middleware can help
- Existing middleware technologies
  - Do not support some architectural concepts (e.g., explicit connectors, configuration)
  - Lack support for remote (re)deployment and monitoring facilities

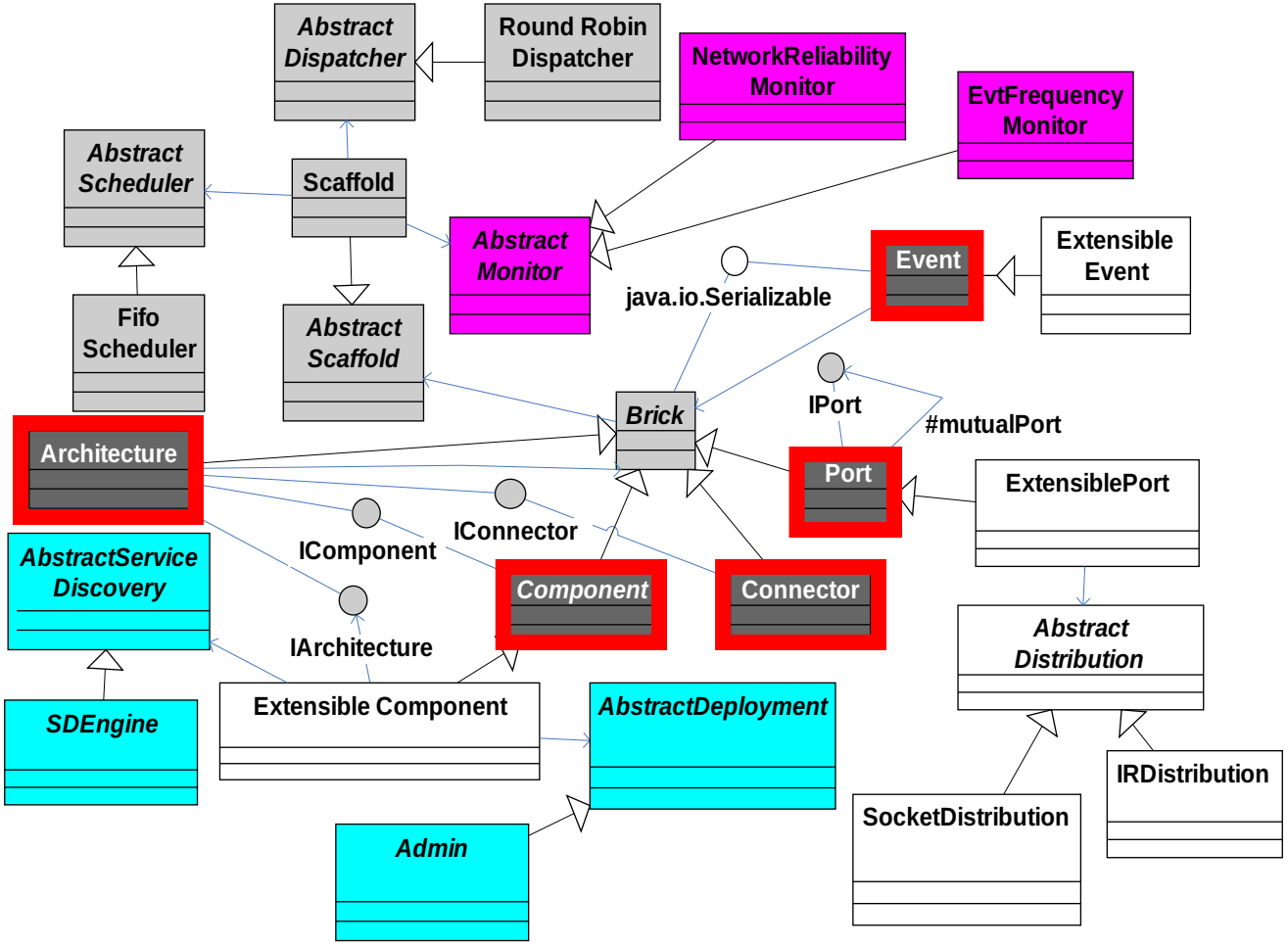
What is needed is “architectural middleware”

Prism-MW

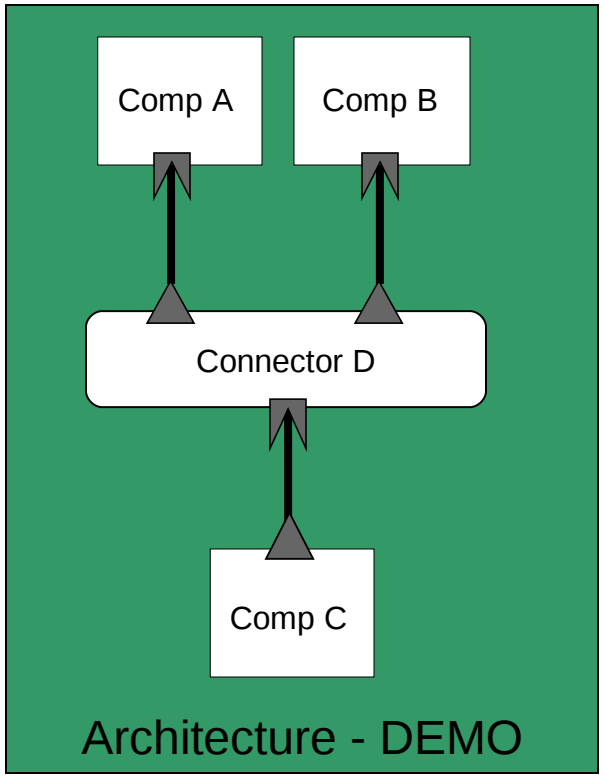
- Prism-MW is an extensible architectural middleware
  - PL-level constructs for architectural concepts
- Extensible design allows for the addition of new facilities
- Developed facilities for
  - (Re)Deployment and (Re)Configuration in terms of architectural constructs
  - Monitoring both at system and architecture level



Prism-MW Design

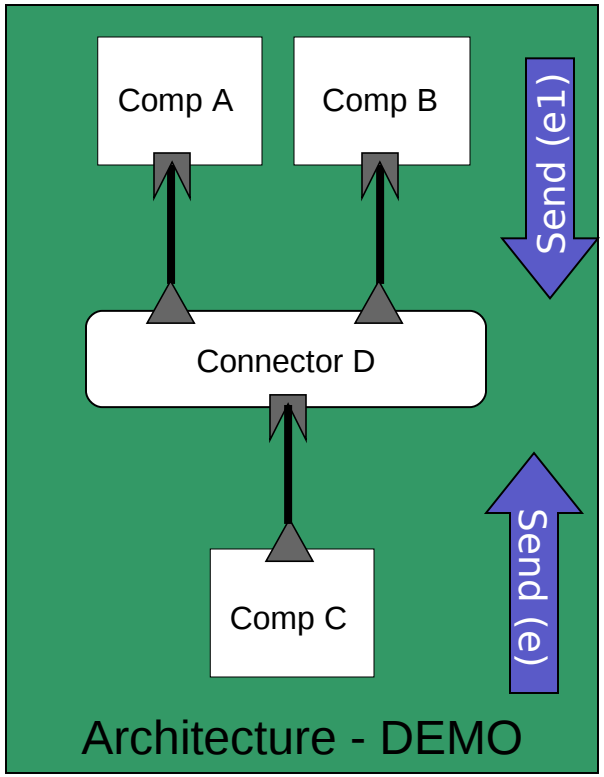


Using Prism-MW




```
class DemoArch {
    static public void main(String argv[]) {
            Architecture arch = new Architecture ("DEMO");
            // create components
            Component a = new CompA ("A");
            Component b = new CompB ("B");
            Component c = new CompC (“C");
            // create connectors
            Connector d = new Connector(“D");
            // add components and connectors
            arch.addComponent(a);
            arch.addComponent(b);
            arch.addComponent(c);
            arch.addConnector(d);
            // establish the interconnections
            arch.attach(a, d);
            arch.attach(b, d);
            arch.attach(d, c);
            }
}
```

Using Prism-MW



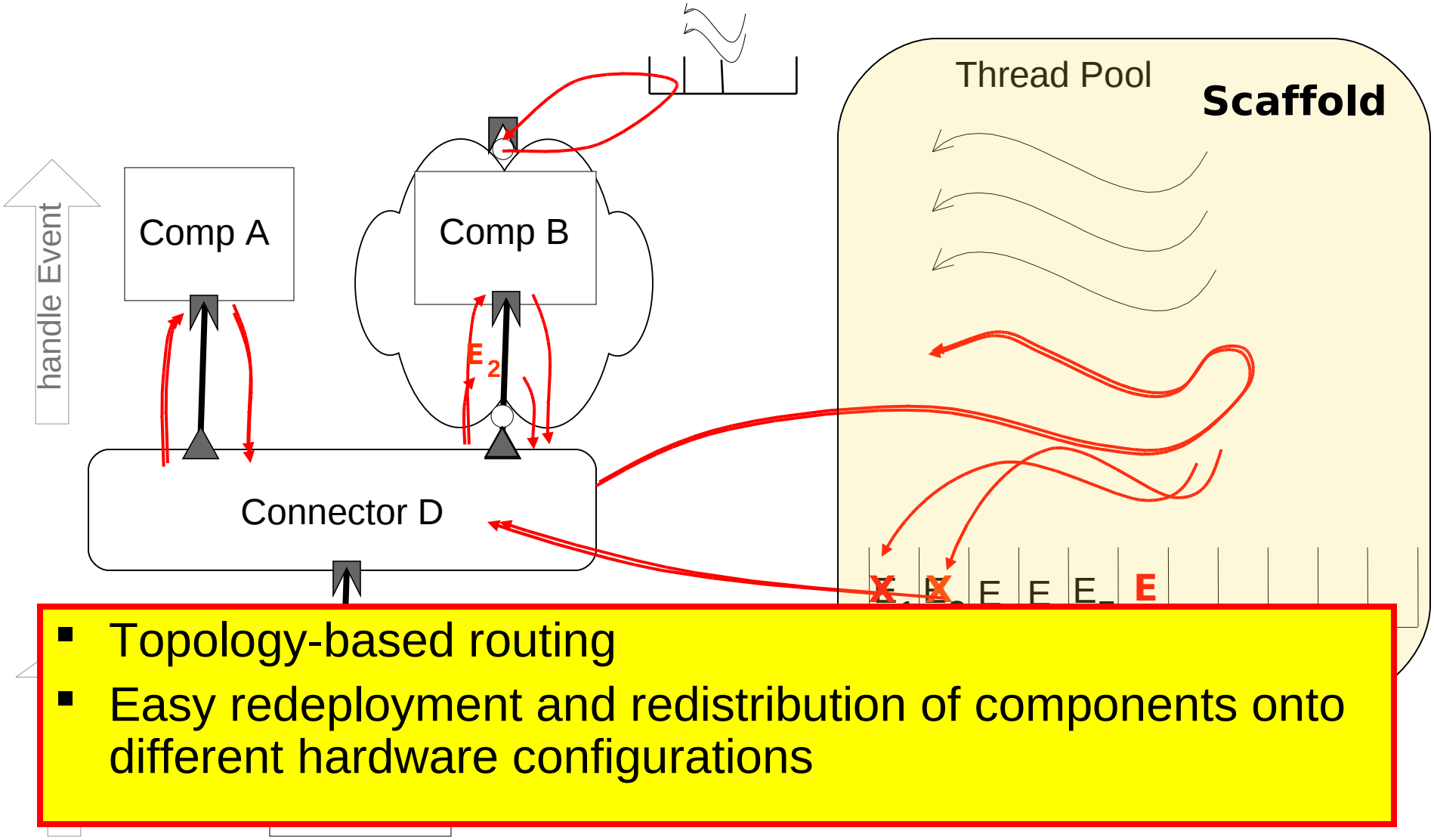
Component C sends an event

```
Event e = new Event ("Event_C“, REQUEST);
e.addParameter("param_1", p1);
send (e);
```

Component B handles the event and sends a response

```
public void handle(Event e)
{
        if (e.equals("Event_C")) {
        ...
        Event e1= new Event("RSP_to_C“, REPLY);
        e1.addParameter("response", resp);
        send(e1);
        }...
}
```

Event Dispatching

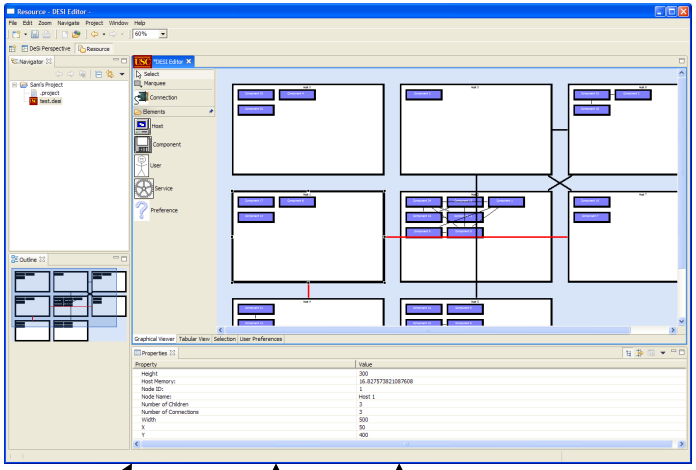


- Topology-based routing
- Easy redeployment and redistribution of components onto different hardware configurations

Outline

- Problem
  - Motivation
  - Approach
  - Outcomes
- Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
- Runtime support
  - Prism-MW
  - Integration
- Collaborations and future work

Deployment and Monitoring Support

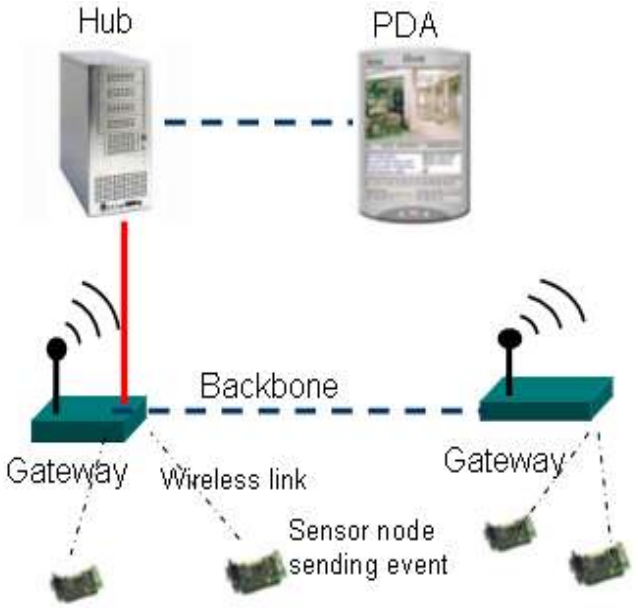


Outline

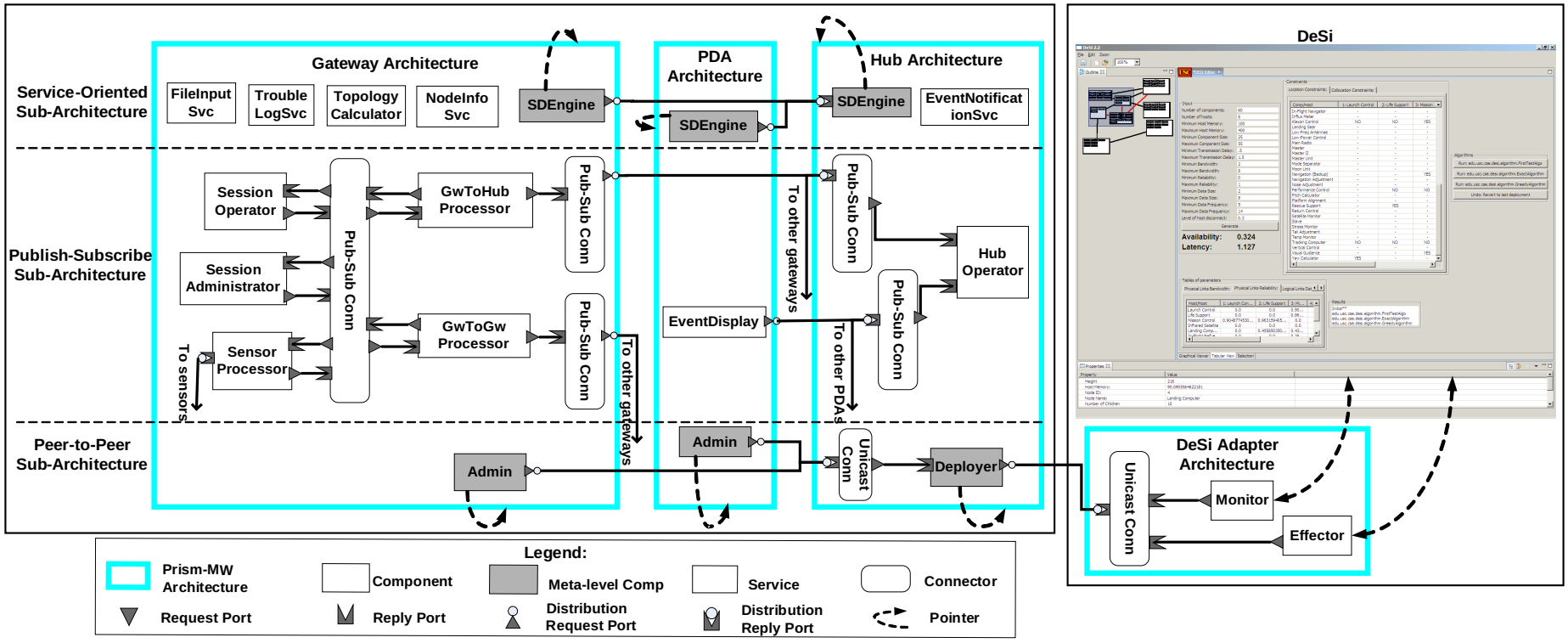
- Problem
  - Motivation
  - Approach
  - Outcomes
- Deployment analysis
  - Formulation
  - Algorithms
  - Tool support
  - Evaluation
- Runtime support
  - Prism-MW
  - Integration
- Collaborations and future work

Collaborations with Industrial Partners

- US Army
  - Emergency Deployment Simulation
  - Large scale search and rescue teams
- Boeing
  - Future Combat Systems
  - Determine allocation of software components on virtual hardware platforms
- NASA's Jet Propulsion Lab
  - Dynamic analysis and adaptation of space mission software
- Bosch Rsrch. & Tech. Center
  - MIDAS
  - Sensor-network product family

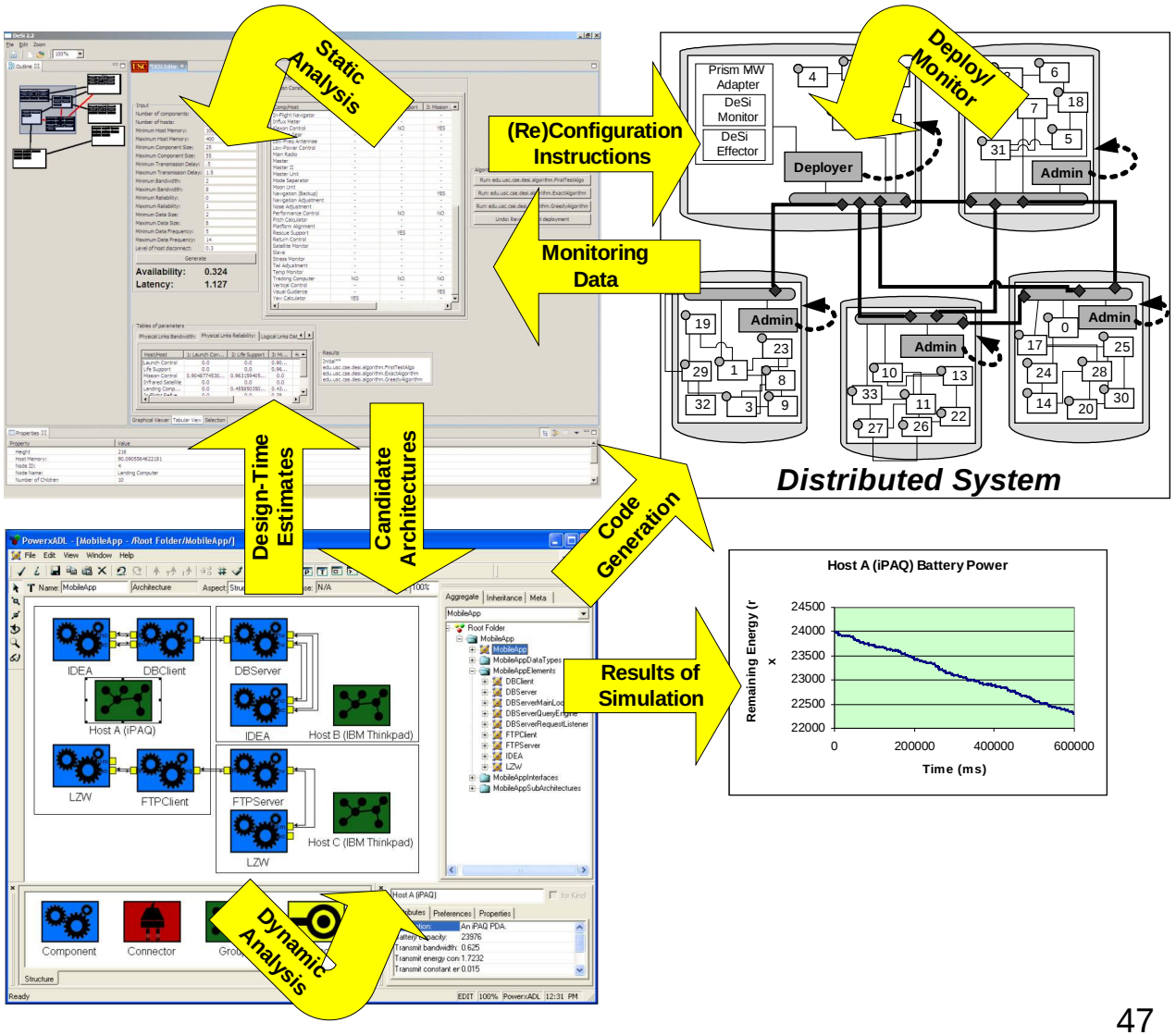


Partial View of MIDAS Architecture



On-Going Work

- Complement static analysis with dynamic analysis
- XTEAM
  - Supports scenario-driven dynamic analysis
  - Provides temporal view of variations in QoS



Remaining Challenges

- A comprehensive software architectural trade-off analysis and implementation framework
- What is the impact of other architectural decisions? What is their relationship?
- How should they be modeled, analyzed, and implemented?