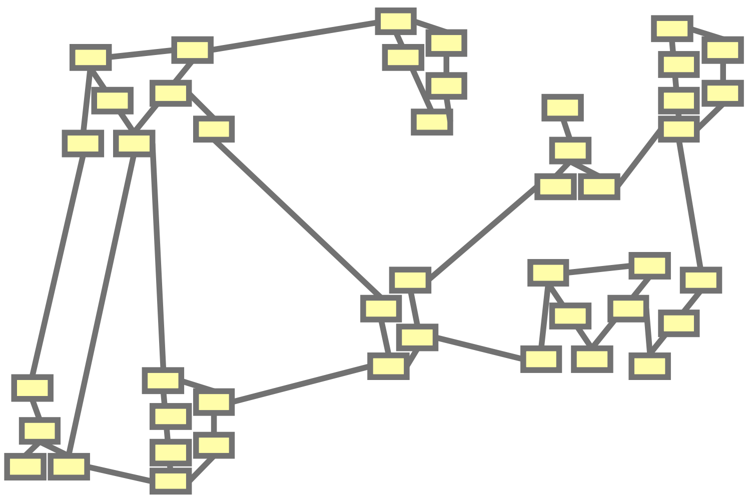
Improving a Distributed Software System's Quality of Service via Redeployment

Nenad Medvidovic
(joint work with Sam Malek, Chiyoung Seo, and Marija Mikic-Rakic)
Computer Science Department
University of Southern California
Los Angeles, CA 90089
neno@usc.edu
http://sunset.usc.edu/~neno/

Deployment Architecture and QoS



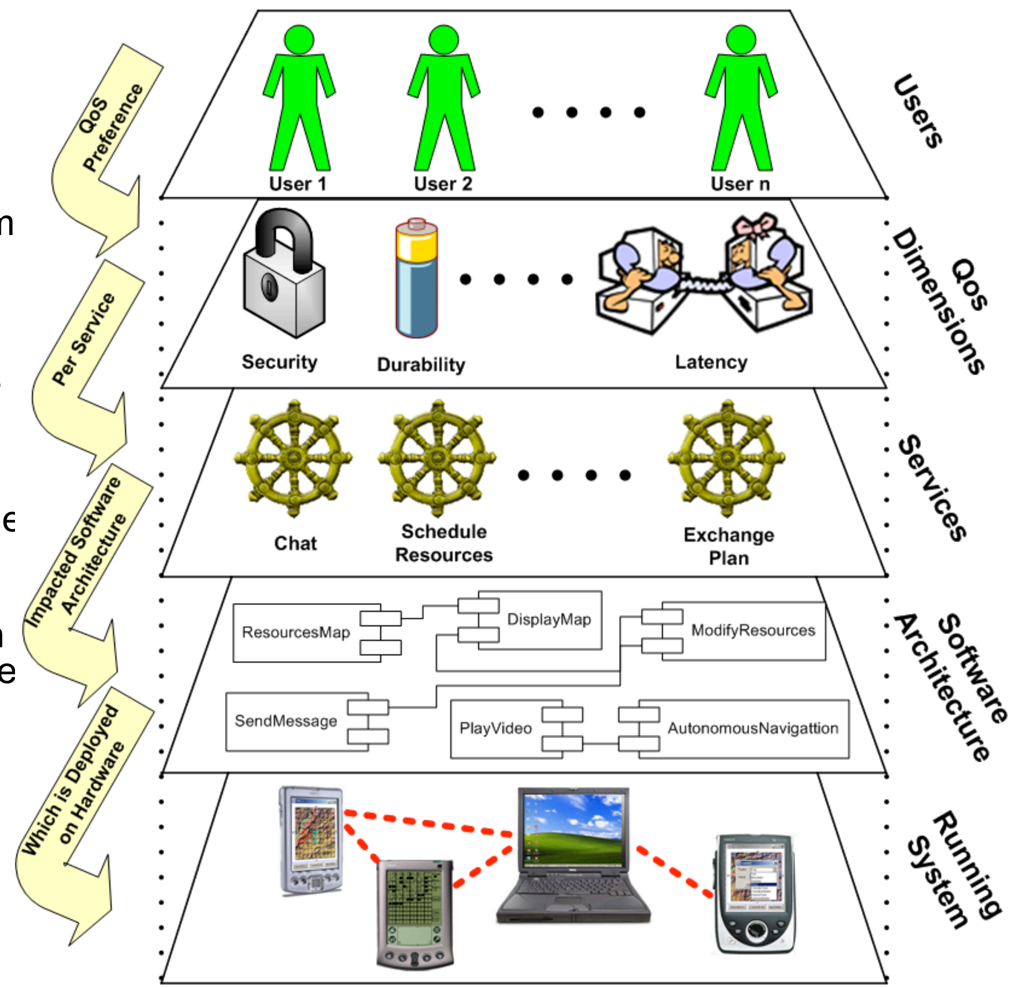
- *Deployment Architecture*: allocation of s/w components to h/w hosts
- h^c deployment architectures are possible for a given system
  - same functionality
  - different qualities of service (QoS)

Problem in a Nutshell

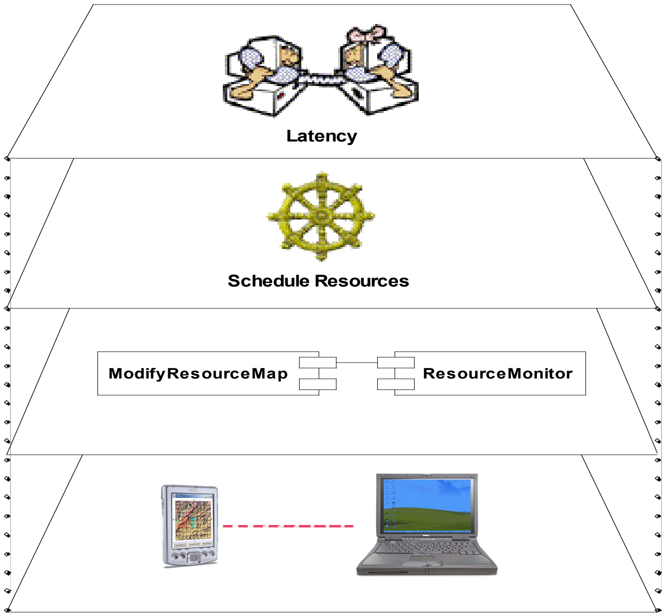
- **Guiding Insight**
  - **System users have varying QoS preferences for the system services they access**
    - Impacts their satisfaction with the system

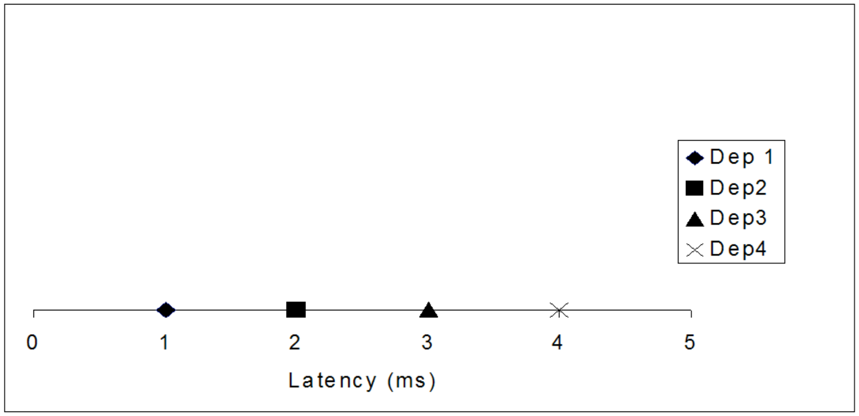
- **Research Question**
  - **How could we improve system's deployment architecture to maximize users' satisfaction?**
    - Where users' satisfaction depends on the system's ability to meet their QoS preferences
    - And where other possible solutions such as caching, hoarding, replication, etc. are not appropriate or ideal

- **Research Objective**
  - **Devise a solution that is applicable to many classes of application scenarios**



Scenario with a Single QoS Dimension

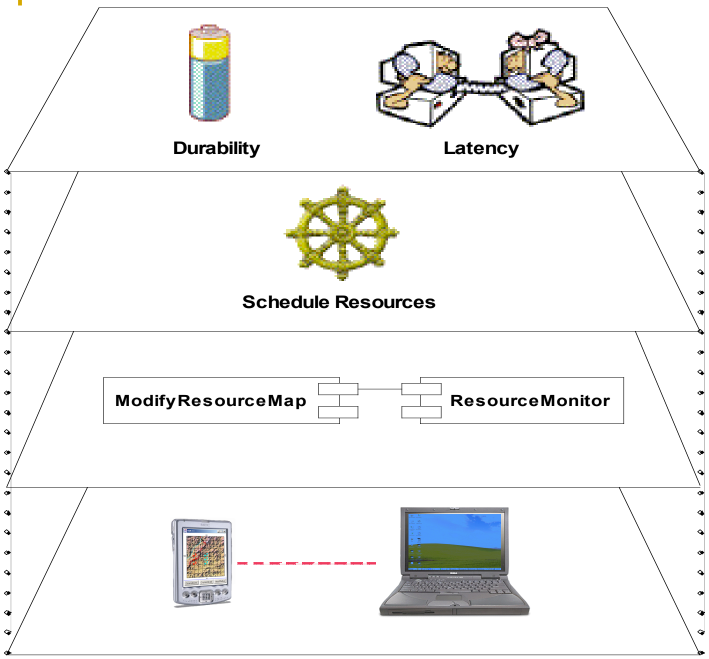


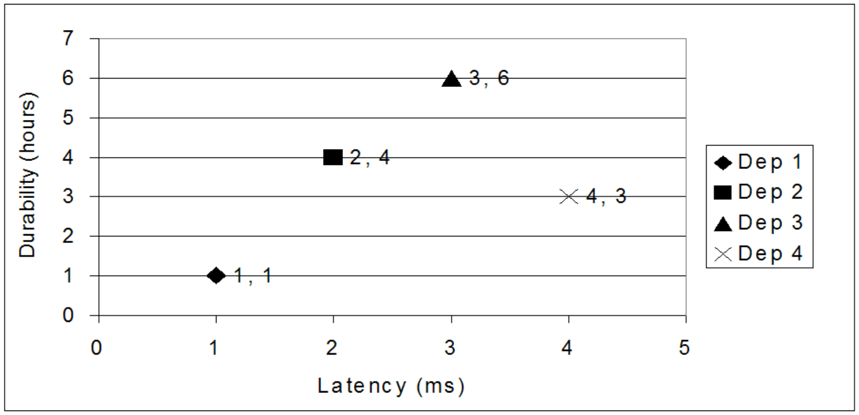


- Objective is to minimize latency
- The *optimal* deployment architecture is deployment 1

➢ Most all related approaches stop here

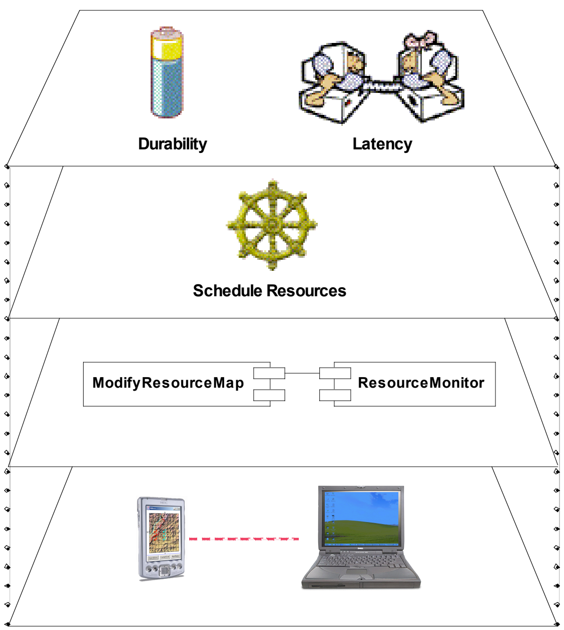
Conflicting QoS Dimensions



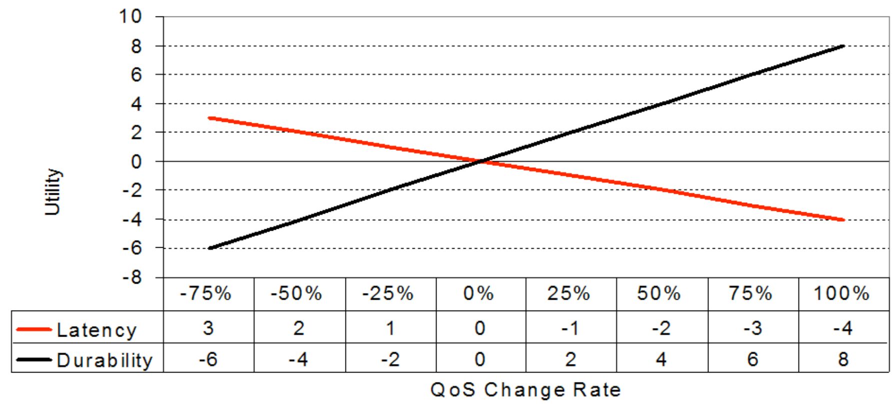


- Objective is to minimize latency and maximize durability
- There is no optimal deployment architecture!
- Phenomenon known as *Pareto Optimal* in multidimensional optimization

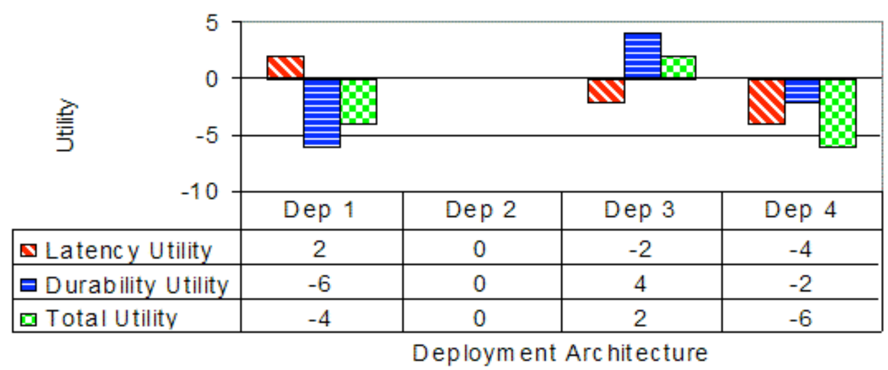
Resolving Trade-Offs between QoS Dimensions



- Explicitly consider
  - system *users*
  - system's *utility* to its users

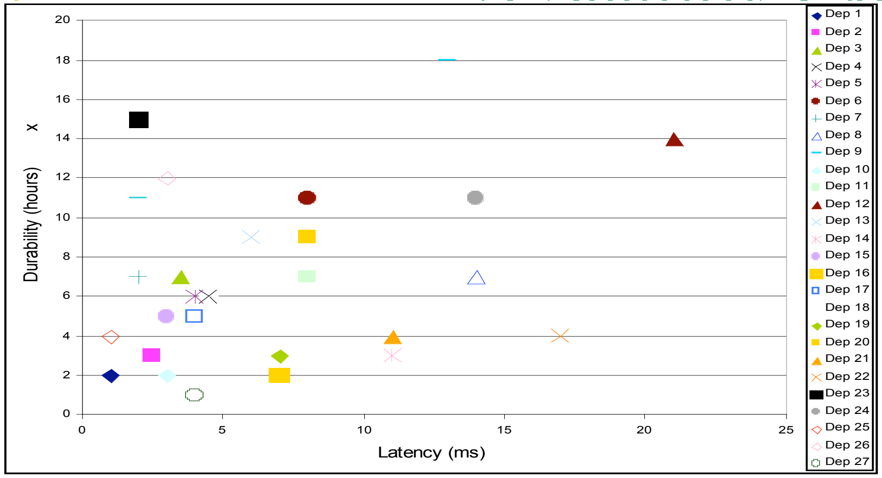


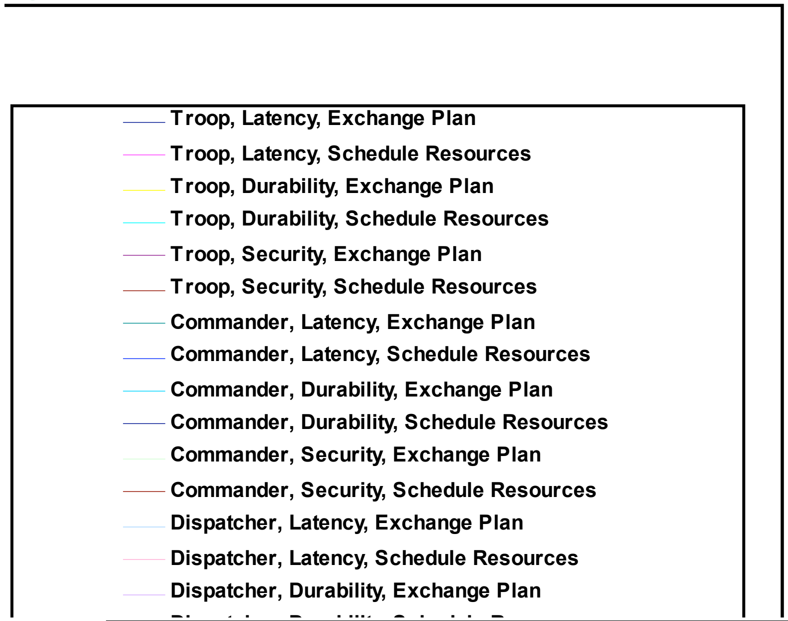
- A *utility function* denotes a user's preferences for a given rate of improvement in a QoS dimension

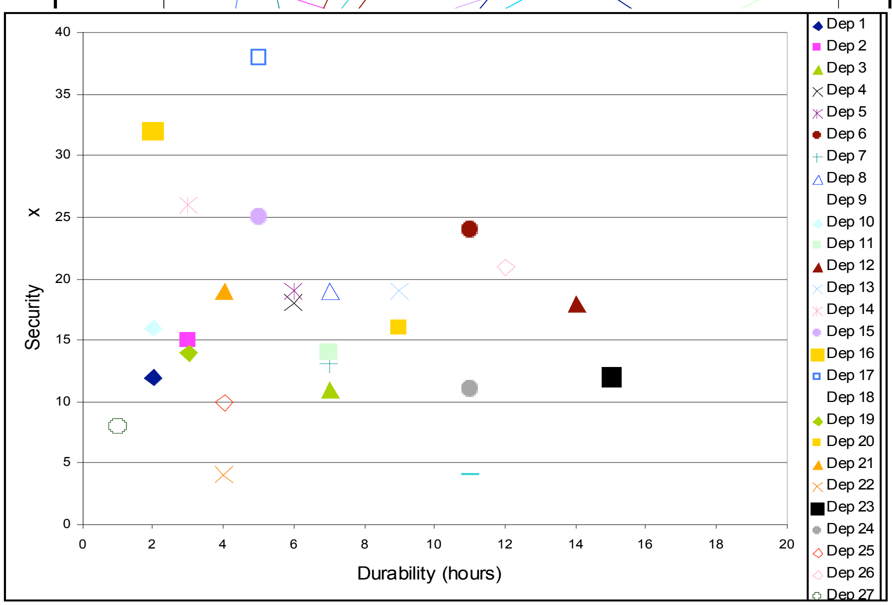


- Allows expression of multidimensional optimization in terms of a single scalar value

A Slightly Larger Scenario

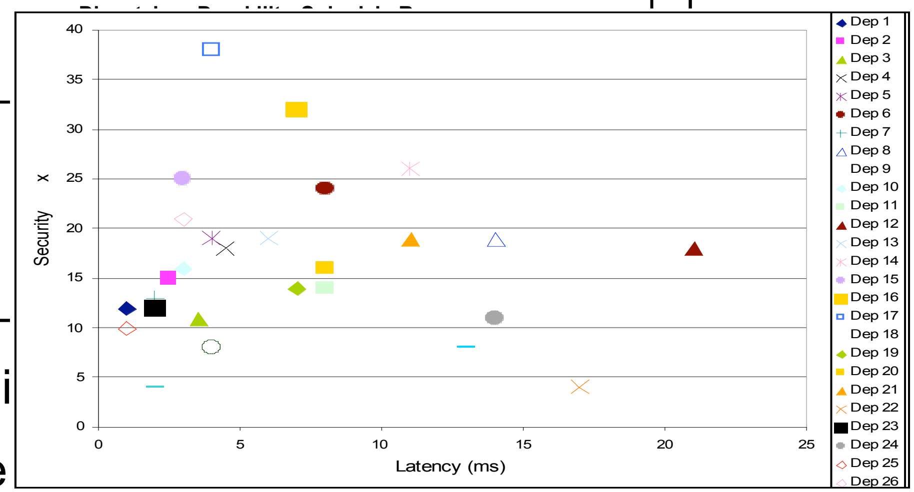






onsi

fere



- Eyeballing the solution quickly becomes impossible!

Proposed Solution

A framework that provides

- an extensible system model
  - inclusion of arbitrary system parameters
  - definition of QoS dimensions using the parameters
  - specification of users' QoS preferences
- multiple QoS improvement algorithms
  - different algorithms suited to different classes of systems
- extensible tool support
  - deployment, execution, and runtime redeployment
  - parameter monitoring and visualization

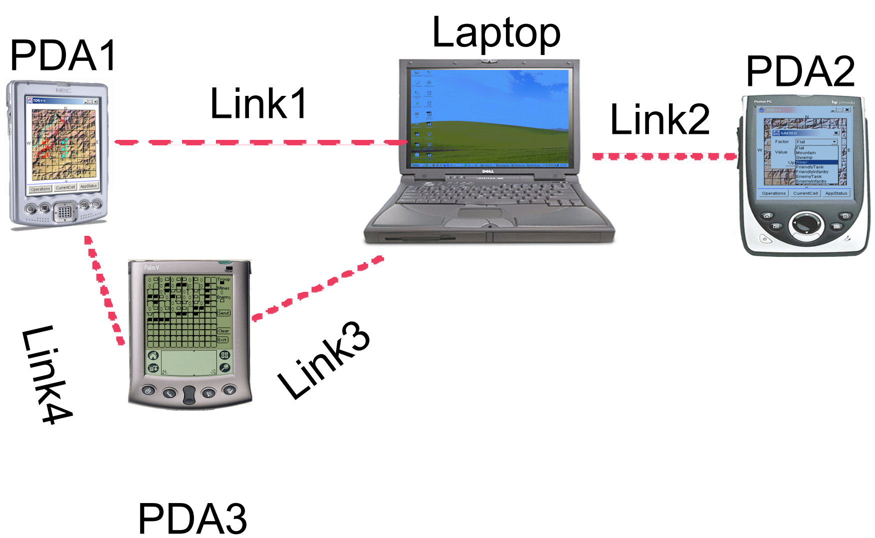
Proposed Solution

A framework that provides

- **an extensible system model**
  - inclusion of arbitrary system parameters
  - definition of QoS dimensions using the parameters
  - specification of users' QoS preferences
- multiple QoS improvement algorithms
  - different algorithm suited to different classes of systems
- extensible tool support
  - deployment, execution, and runtime redeployment
  - parameter monitoring and visualization

Model of the Hardware System

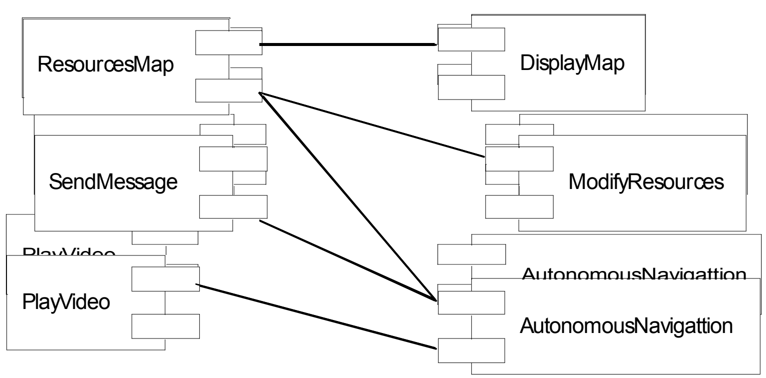
- A set *H* of hardware nodes
  - *H={PDA1, PDA2, PDA3, Laptop}*
- A set *HP* of host parameters
  - *HP={memory, battery}*
- A function *hParam:H×HP→R*
  - *hParam(PDA1, memory)=20MB*
- A set N of network links
  - N={Link1, Link2, Link3, Link4}
- A set NP of network link parameters
  - NP={reliability, bandwidth}
- A function nParam:N×*NP→R*
  - *nParam(Link1, bandwidth)=256kb/s*

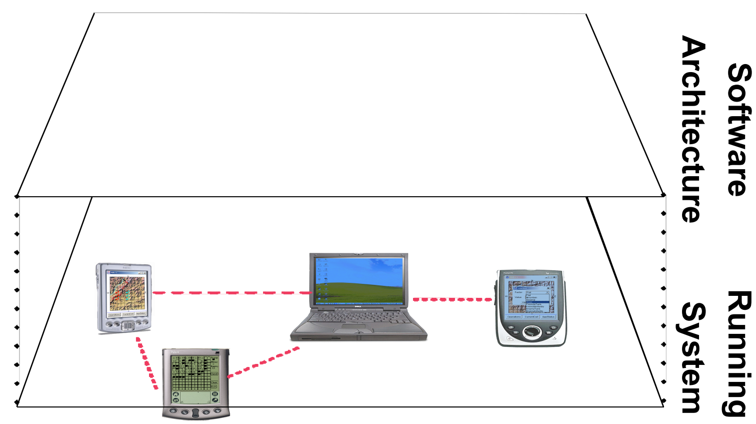


Model of the Software Architecture

- A set *C* of software components
  - *C={ResourcesMap, DisplayMap, …}*
- A set *CP* of component parameters
  - *CP={size, CPU usage}*
- A function *cParam:C×CP→R*
  - *cParam(DisplayMap, size)=50Kb*
- A set *I* of logical links
  - *I={renderMap, updateMap, …}*
- A set *IP* of logical link parameters
  - *IP={frequency, average event size, …}*
- A function *IParam:I×IP→R*
  - *IParam(renderMap,freqency)=20*

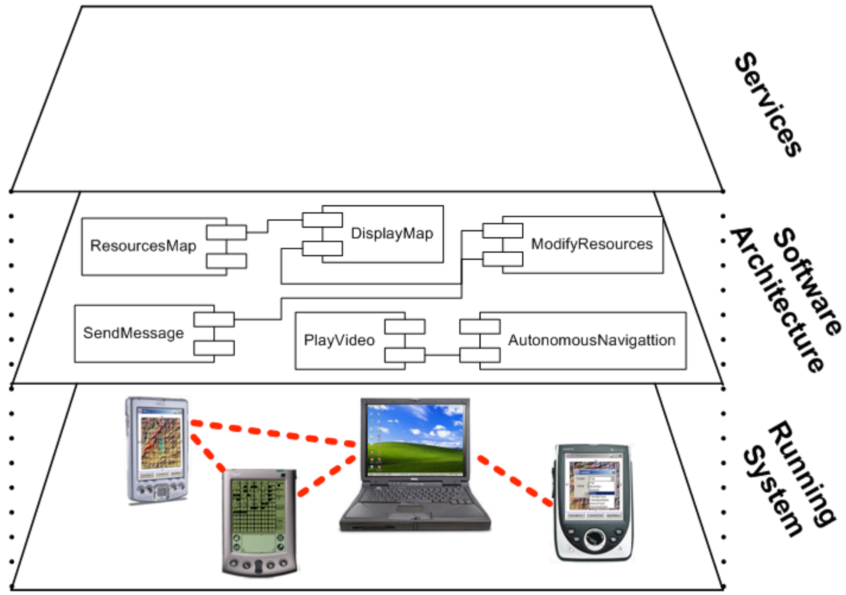
- A set *DepSpace={d1, d2, …}* of all possible deployment mappings





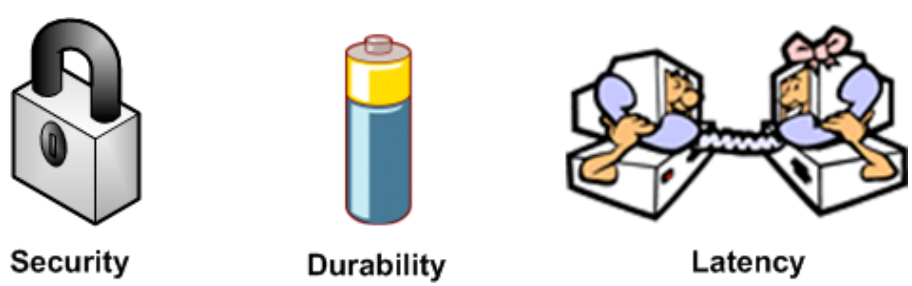
Model of the System Services

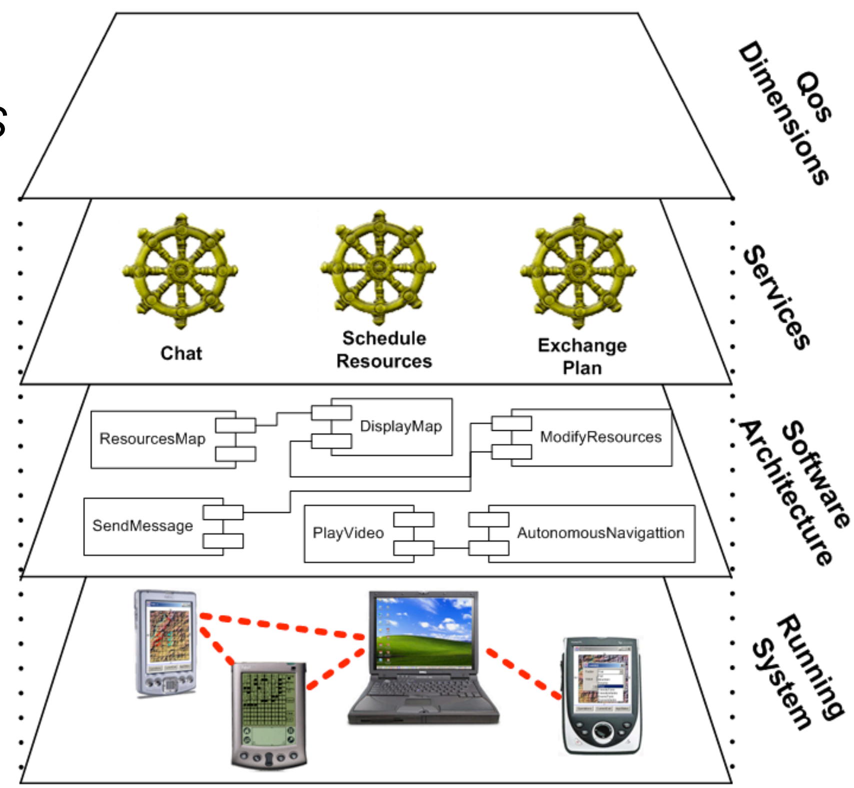
- A set *S* of service
  - *S={Chat, Scheduler Resources, Exchange Plan}*
- A function *sParam:S* × *{H* ∪ *C* ∪ *N* ∪ *I}* × *{HP* ∪ *CP* ∪ *NP* ∪ *IP}* → *R* of values for service-specific system parameters
  - *sParam(Schedule Resources, renderMap, frequency of execution)* = *3*



Model of the QoS Dimensions

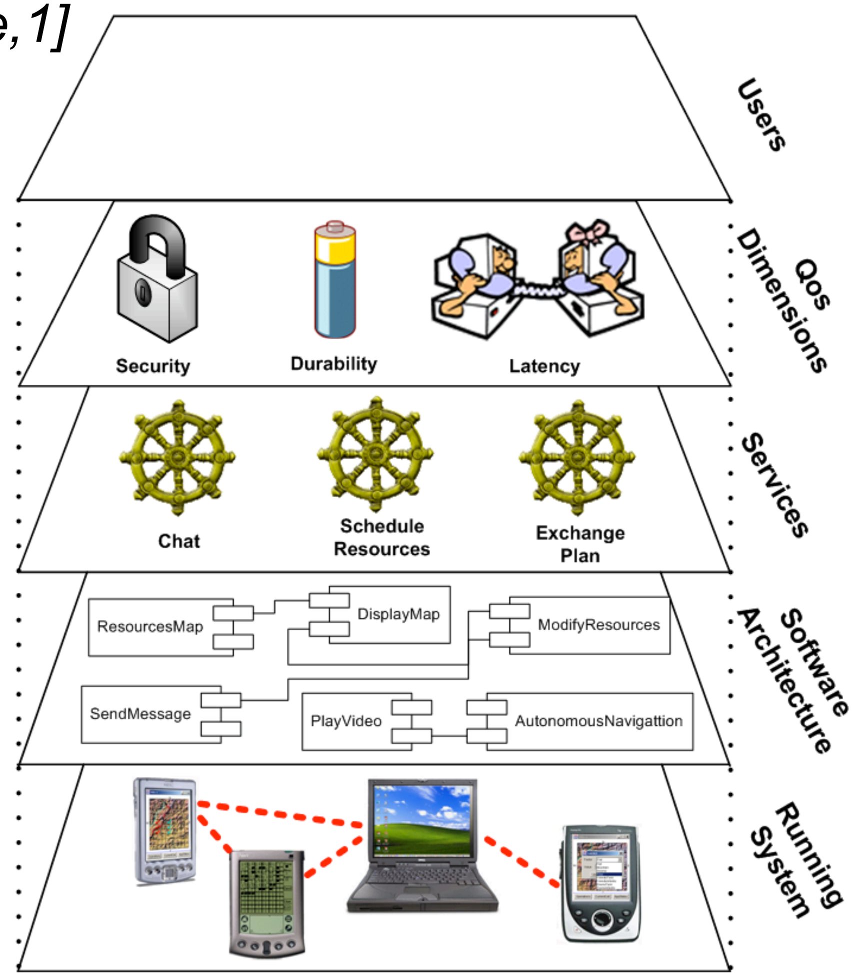
- A set *Q* of quality of service dimensions
  - *Q={security, durability, latency}*
- A function *qValue:S×Q×DepSpace → R that quantifies the achieved level of QoS*
  - *qValue(chat, latency, d1)=5ms*
- *A function qType:Q → {-1,1}*
  - *-1 denotes it is desirable to minimize the QoS*
  - *1 denotes it is desirable to maximize the QoS*





Model of the System Users

- A set *U* of users
  - *U={Troop, Commander, Dispatcher}*
- A function *qosRate:U×S×Q → [MinRate,1]*
  - represents the rate of change in QoS
- A complementary function *qosUtil:U×S×Q → [0,MaxUtil]*
  - *represents the utility for that rate of change*
- A user's priority can be expressed as the ratio of *MaxUtil* to *MinRate*



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

$$overallUtil(d,d') = \sum_{u=1}^{|U|}\sum_{s=1}^{|S|}\sum_{q=1}^{|Q|}\left(\frac{\left(\frac{qValue(s,q,d)-qValue(s,q,d')}{qValue(s,q,d')}\right)}{qosRate(u,s,q)} * qosUtil(u,s,q) * qType(q)\right)$$

Amount of improvement over deployment *d'*

Rate of improvement

Utility of improvement

+1 for maximizing QoS

-1 for minimizing QoS

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
- …

Framework Instantiation

- The engineer needs to specify the "loosely" defined elements of the model
  - Define the pertinent properties of the application scenario
  - Define QoS dimensions in terms of system properties

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

  - Define system parameter constraints

Proposed Solution

A framework that provides

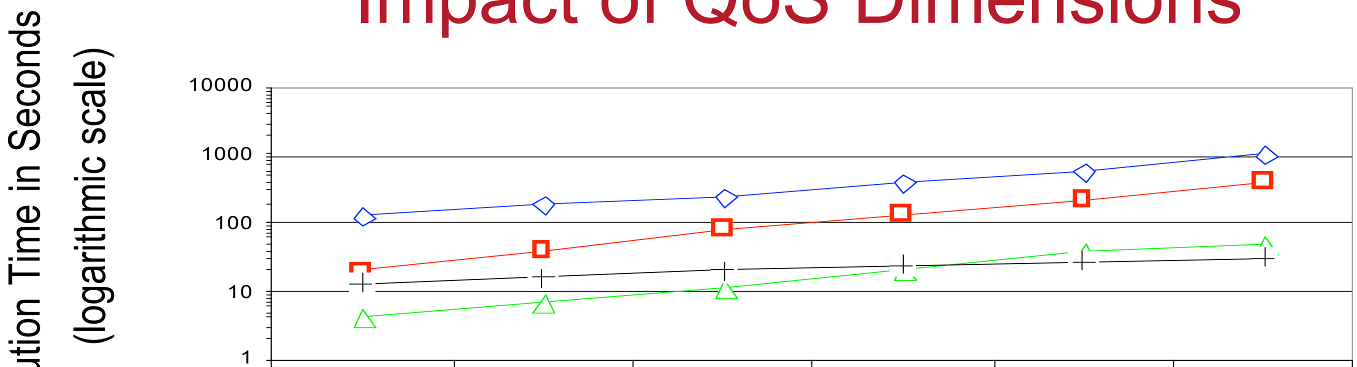
- an extensible system model
  - inclusion of arbitrary system parameters
  - definition of QoS dimensions using the parameters
  - specification of users' QoS preferences
- **multiple QoS improvement algorithms**
  - different algorithm suited to different classes of systems
- extensible tool support
  - deployment, execution, and runtime redeployment
  - parameter monitoring and visualization

Algorithms

- MINLP – polynomial (?)
  - Represented the problem as a set of (non-)linear constraint functions
  - Does not guarantee the optimal solution or convergence
- MIP – exponential: $O(2^{|H|^2|C|^2})$
  - Devised an approach to transform our MINLP problem to MIP
  - Developed heuristics to decrease complexity to $O(|H|^{|C|})$
- Greedy – polynomial: $O(\ |S|^3\ (|C|\ |U|\ |Q|)^2\ )$
  - An iterative algorithm that leverages several heuristics for
    - Ranking elements of our problem (services, hosts, components, …)
    - Assigning software components to hardware hosts
  - Makes local decisions that often maximize the global objective
- Genetic – linear: $O(\#populations \times \#evolutions \times \#individuals \times |S|\ |U|\ |Q|)$
  - An individual represents a solution composed of a sequence of genes
  - A population contains a pool of individuals which are evolved via cross-overs and mutations
  - The accuracy on the representation depends on the ability to promote “good” genes
    - Bad representation does not promote “good” genes ➔ random search
- Market-Based (Auctioning)
  - Under development and evaluation

Algorithms' Performance

Impact of QoS Dimensions

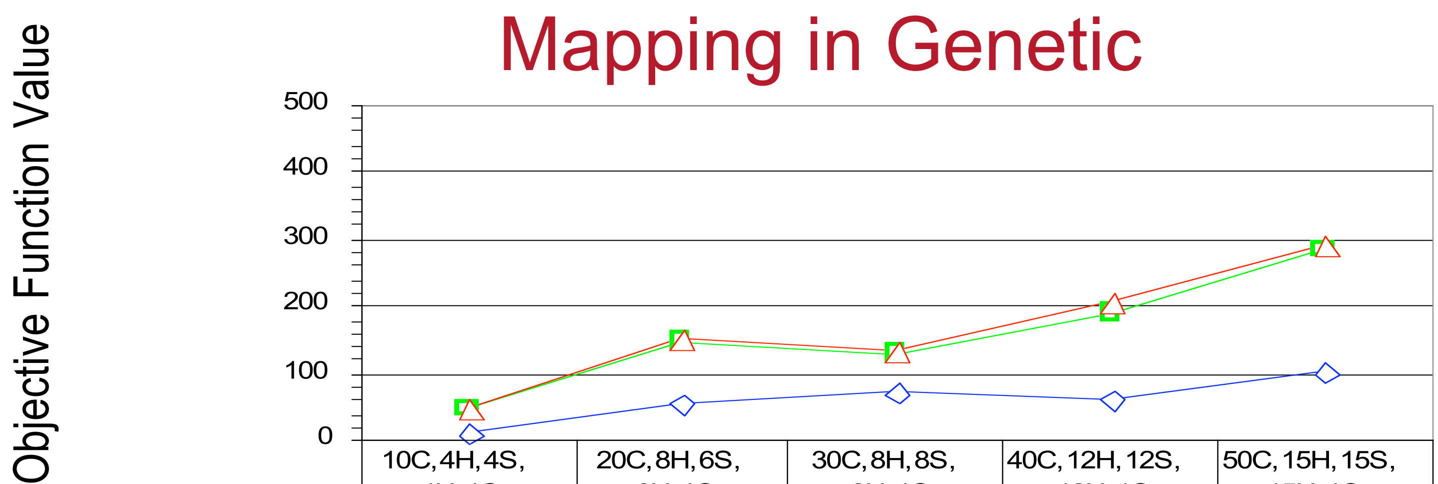


	1 QoS	2 QoS	3 QoS	4 QoS	5 QoS	6 QoS
MIP	130	192	250	400	602	1017
MINLP	20	41	81	132	226	410
Greedy	4	7	11	20	37	49
Genetic	13	16	20	25	27	31

Number of QoS dimensions

Problem Size

Impact of Heuristics



	10C, 4H, 4S, 4U, 4Q	20C, 8H, 6S, 6U, 4Q	30C, 8H, 8S, 8U, 4Q	40C, 12H, 12S, 12U, 4Q	50C, 15H, 15S, 15U, 4Q
Genetic without mapping	13	57	73	65	102
Genetic with mapping	48	150	130	192	285
Genetic with mapping and three parallel executing populations	51	156	132	210	294

Problem Size

Algorithms in Practice

QoS	MIP				MINLP				Greedy				Genetic			
	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

Algorithmic Trade-Offs

- Architectural style
  - E.g., Client-Server vs. Peer-to-Peer
  - MIP algorithm for very constrained architectures
  - One of the optimization algorithms for flexible and large architectures
- Large number of QoS dimensions
  - Genetic outperforms the greedy
  - Genetic is only linearly affected by the number of QoS dimensions
- Stable vs. unstable systems
  - For small and stable systems, MIP algorithm is worth the time and resources required to compute a solution
  - For large and unstable systems, genetic or greedy is more applicable
- Resource constrained systems
  - Genetic algorithm can execute in parallel on multiple devices
    - Sharing the overhead among many hosts
- Centralized vs. decentralized systems
  - Market-based algorithms could also be leveraged in a decentralized setting

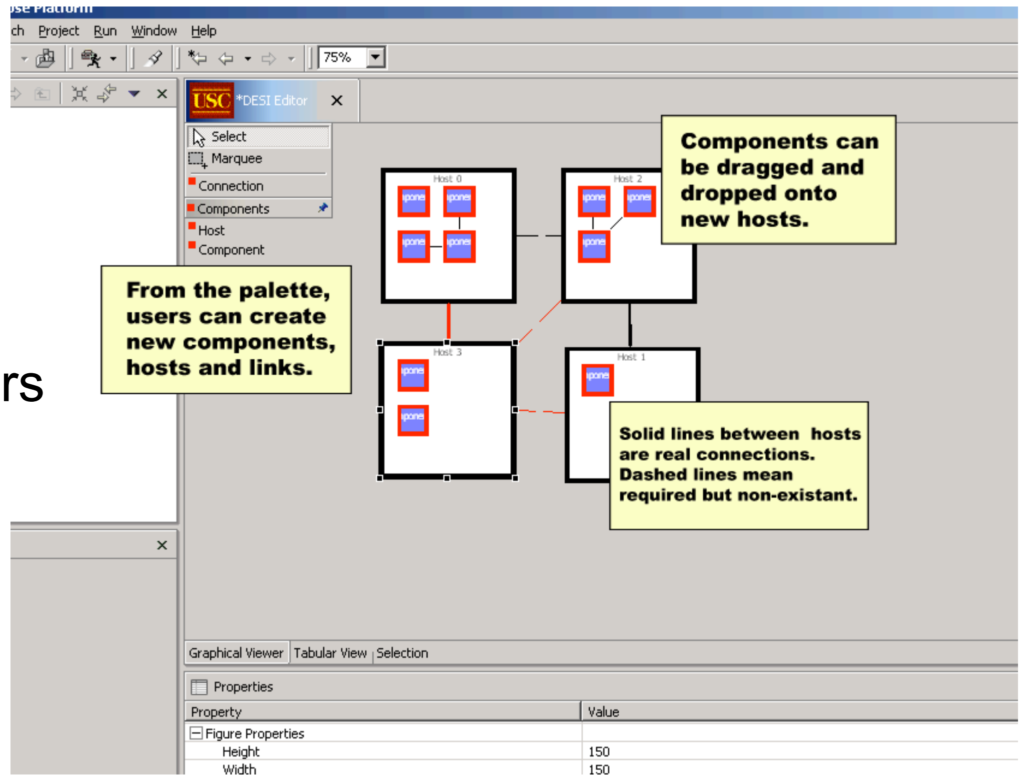
Proposed Solution

A framework that provides

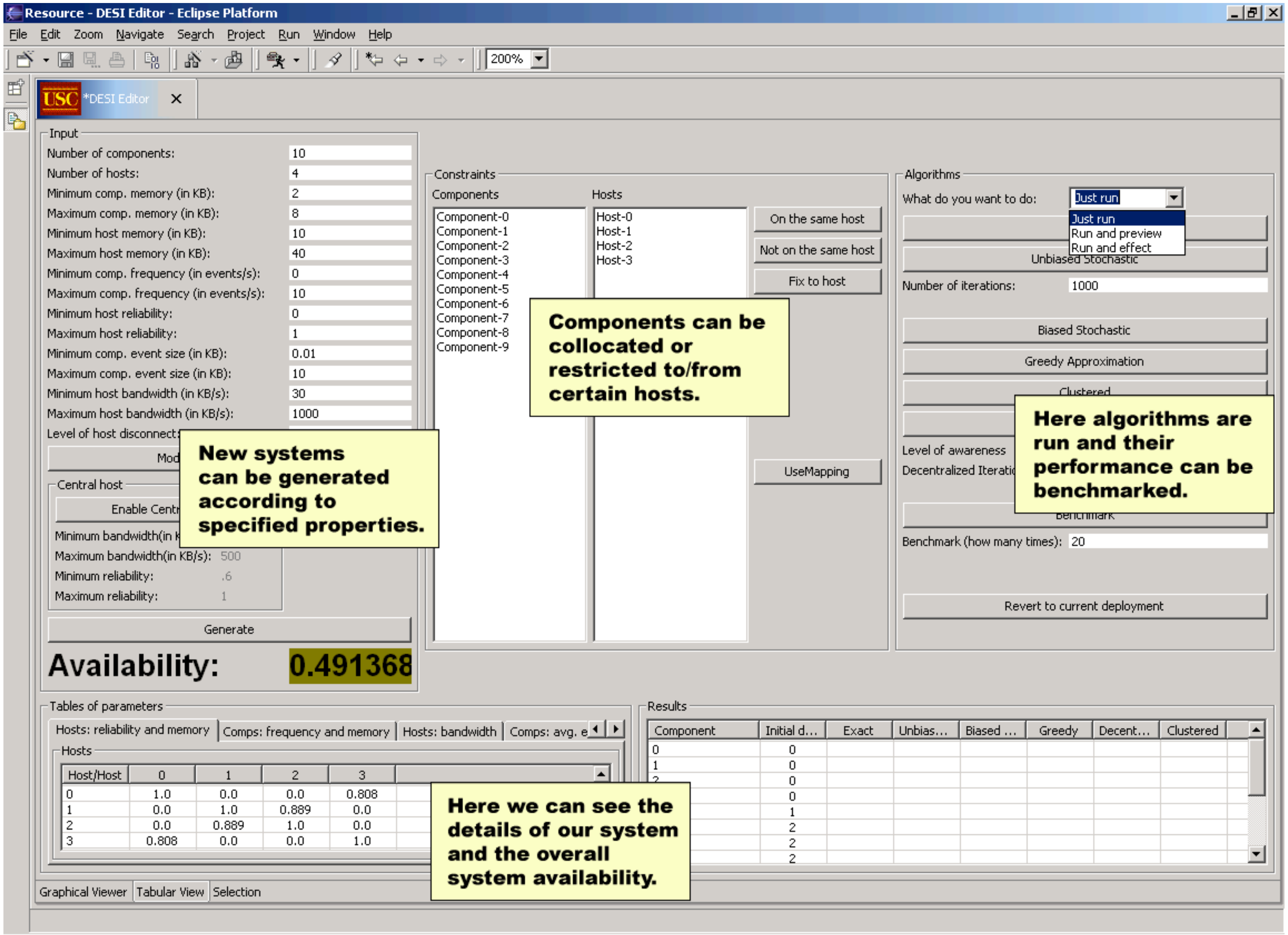
- an extensible system model
  - inclusion of arbitrary system parameters
  - definition of QoS dimensions using the parameters
  - specification of users' QoS preferences
- multiple QoS improvement algorithms
  - different algorithm suited to different classes of systems
- **extensible tool support**
  - deployment, execution, and runtime redeployment
  - parameter monitoring and visualization

Modeling and Analysis Support – DeSi

- DeSi is a visual environment for analyzing deployment architectures
- It allows for modeling a distributed system in terms of four basic elements
  - Software components
  - Hardware devices
  - Network links
  - Logical (interaction) links
- Each of these elements has an associated set of parameters
  - Accessed via property sheets
- DeSi is extensible
  - Allows for modeling of new parameters and properties
  - Views are completely separated from the model

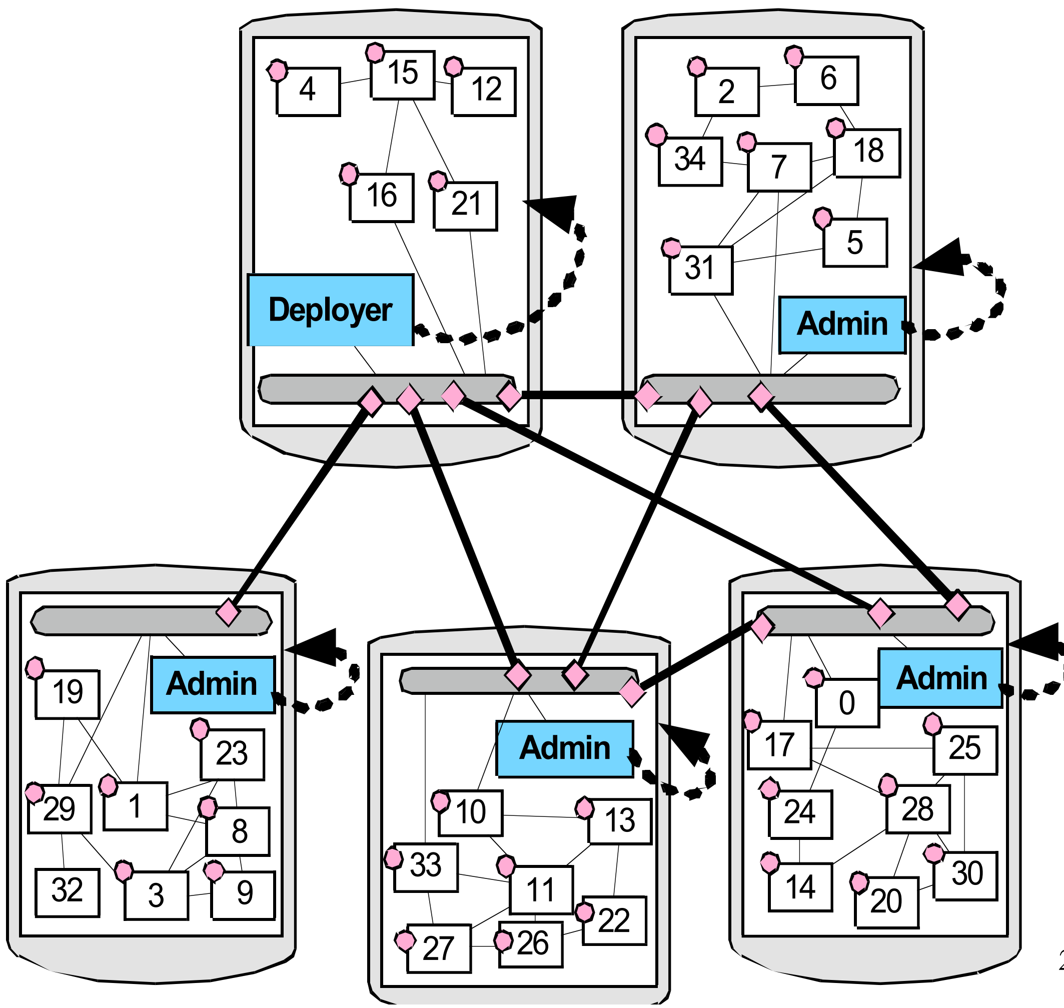


DeSi – Control Panel

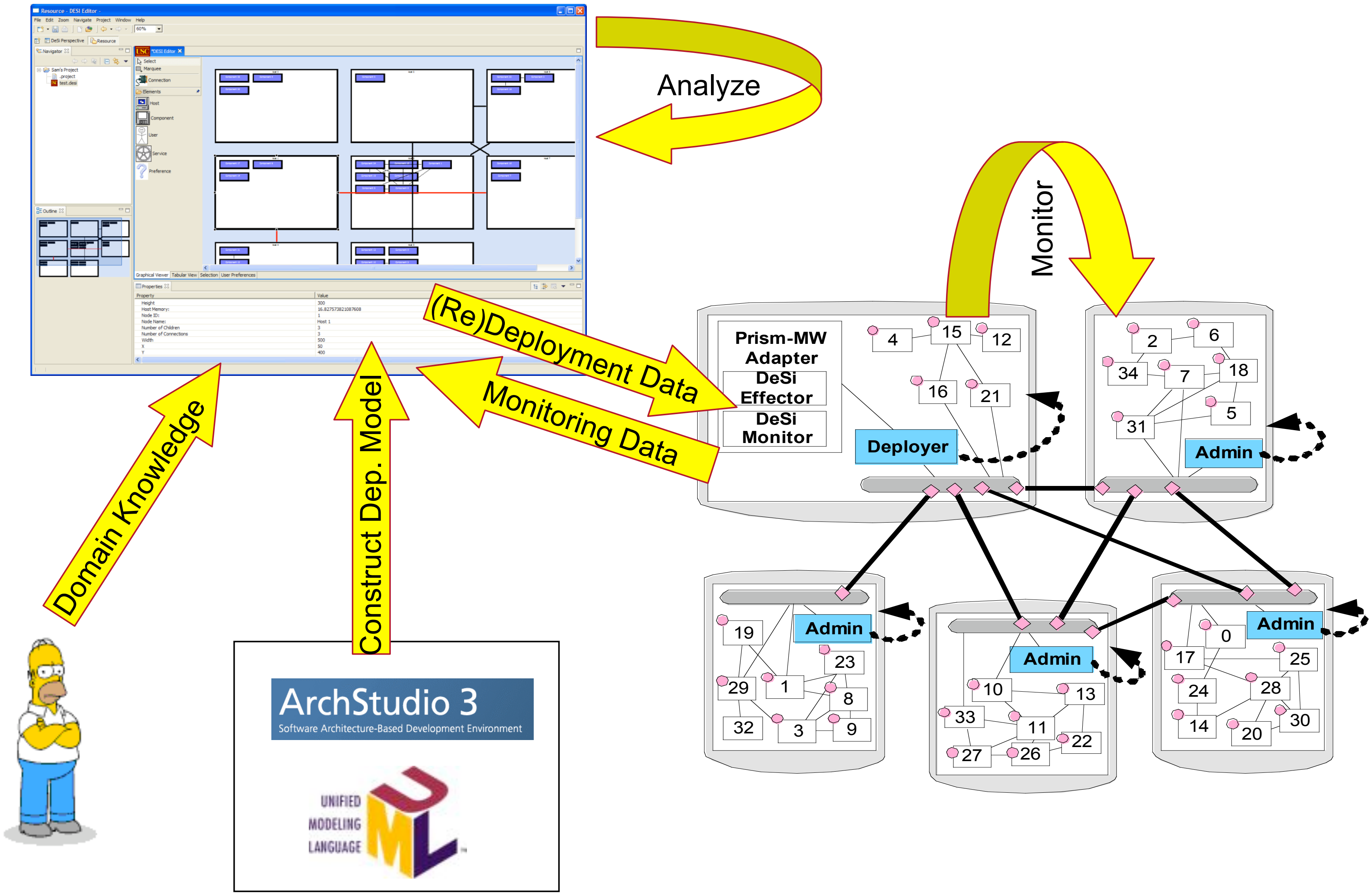


Implementation and Execution Support – Prism-MW

- Prism-MW is an extensible architectural middleware
  - PL-level constructs architectural concepts
    - components
    - connectors
    - ports, etc.
  - Facilities for monitoring and (re)deployment of a distributed system
  - Allows for the addition of new monitoring and deployment facilities



Tool Suite Integration



Contributions

- Address system deployment as a multidimensional optimization problem
  - Leverages users' preferences to resolve inherent trade-offs in conflicting QoS dimensions
- Explicitly consider system's high-level services and their internal architecture
- An extensible modeling approach that can be leveraged across different application scenarios
  - Specify arbitrary system parameters
  - Define arbitrary QoS dimensions in terms of system parameters
- A suite of generic multidimensional optimization algorithms
  - Operate on top of an instantiated model of a system
- A suite of customizable tools
  - A number of extension points are leveraged to configure the tools to the application scenario at hand
  - Promotes reuse and cross-evaluation of solutions to this problem

On-Going Work

- Further profiling of the algorithms
  - Determine which algorithms are suitable to what classes of systems
- Several on-going enhancements to DeSi
  - Addition of new modeling elements: users, user preferences, services, etc.
- Complete the integration of Prism-MW, DeSi, and ArchStudio
- Develop the support for autonomically selecting appropriate redeployment algorithms
- Evaluate the approach on real distributed systems
  - Troops Deployment System (TDS)
  - Midas

Questions