The Utility of Utility: Policies for Autonomic Computing

Jeff Kephart (kephart@us.ibm.com)
IBM Thomas J Watson Research Center
Hawthorne, NY, USA
IT is Becoming Too Complex!
Autonomic Computing and Agents

- **AC definition**
  - “Computing systems that manage themselves in accordance with high-level objectives from humans.” Kephart & Chess, IEEE Computer 2003
  - Self-configuring, self-healing, self-optimizing, self-protecting

- **Agents definition**
  - “An encapsulated computer system, situated in some environment, and capable of flexible, autonomous action in that environment in order to meet its design objectives.” Jennings, et al, A Roadmap of Agent Research and Development, JAAMAS 1998

- **Autonomic elements ~ agents**
- **Autonomic systems ~ multi-agent systems**
The focus of this talk

- I start from two premises:
  - Autonomic systems are “Computing systems that manage themselves in accordance with high-level objectives from humans.”
  - Autonomic systems ~ multi-agent systems

- Which leads to…

- How do we get a (decentralized) Multi-Agent System to act in accordance with high-level objectives?

- My claim
  - Objectives should be expressed in terms of utility
  - Utility is an essential piece of information that must be processed, transformed, and communicated by agents
Outline

- Autonomic Computing and Multi-Agent Systems
  - Utility Functions
    - As means for expressing high-level objectives
    - As means for managing to high-level objectives
  - Examples
    - Unity, and its commercialization
    - Power and performance objectives and tradeoffs
    - Applying utility concepts at the data center level

- Conclusions
How to *represent* high-level policies?

- Utility functions map any possible state of a system to a scalar value.
- They can be obtained from:
  - Service Level Agreement
  - Preference elicitation
  - Simple templates
- They are a very useful representation for high-level objectives:
  - Value can be transformed and propagated among agents to guide system behavior.

$U(\text{RT}) = \begin{cases} 
V_1 \text{ for } a_1 \\
V_2 \text{ for } a_2 \\
V_3 \text{ for } a_3 
\end{cases}$

Kephart and Walsh, Policy04
How to manage with high-level policies?

- **Elicit** utility function $U(S)$ expressed in terms of service attributes $S$

- **Model** how each attribute $S_i$ depends on controls $C$ and observables $O$
  - Models expressed as $S(C; O)$
  - E.g., $RT(\text{routing weights, request rate})$
  - Models from experiments, learning, theory

- **Transform** from service utility $U$ to resource utility $U'$ by substitution
  - $U(S) = U(S(C; O)) = U'(C; O)$

- **Optimize** resource utility. As observable $O$ changes, set $C$ to values that maximize $U'(C; O)$
  - $C^*(O) = \text{argmax}_C U'(C; O)$
  - $U'(O) = U'(C^*(O); O)$
Outline

▪ Autonomic Computing and Multi-Agent Systems

▪ Utility Functions
  – As means for expressing high-level objectives
  – As means for managing to high-level objectives

▪ Examples
  – Unity, and its commercialization
  – Power and performance objectives and tradeoffs
  – Applying utility concepts at the data center level

▪ Conclusions
Unity Data Center Prototype: Experimental setup

Resource Arbiter

Maximize Total SLA Revenue
5 sec

U(#srv)

U(#srv)

U(#srv)

U(#srv)

Demand (HTTP req/sec)

Trade3

Demand (HTTP req/sec)

Trade3

How App Mgr computes its external resource utility

Alternative to generating full curve: utility elicitation

Patrascu, Boutilier et al. New Approaches to Optimization and Utility Elicitation in Autonomic Computing, AAAI 2005

Elicit:
- \( U(\text{RT}) \) Service-level utility
- My controls
- Arbiter’s controls
- Observable

Model:
- \( U(\text{RT}(C; \text{srv}, \lambda)) \)

Transform:
- \( U’(C; \text{srv}, \lambda) = U(\text{RT}(C; \text{srv}, \lambda)) \)

Optimize:
- \( C^*(\text{srv}, \lambda) = \arg\max_C U’(C; \text{srv}, \lambda) \)

- \( U''(\text{srv}, \lambda) = U’(C^*(\text{srv}, \lambda); \text{srv}, \lambda) \)

Chess, Segal, Whalley and White, Unity: Experiences with a Prototype Autonomic Computing System, ICAC 2004
How the Arbiter determines optimal resource allocation

**Decision problem:**
Allocate resources

\[ \text{srv}^* = \arg \max_{\text{srv}} \sum \text{U''}_i(\text{srv}_i) \]
Effectively maximizes \( \sum \text{U}_i(\text{S}_i) \)

**Resource Arbiter**

\[ \text{U}'_1(\text{srv}_1) \]
\[ \text{U}'_2(\text{srv}_2) \]

**Max Utility**

**Number of servers**

**App Manager**

**WebSphere 5.1**

**DB2**

**Trade3**

**U(RT)**
How we commercialized Unity

Tivoli Intelligent Orchestrator

If I give you $n$ servers, how often will you exceed the response time goal?

If I give you $n$ servers, how valuable will that be?

I need 300M CPU cycles/sec

$U(S)$

WebSphere Extended Deployment

$V(n)$

The confusing old way

The clean new way

This was not actually that simple – product release cycles didn’t mesh, so we needed an evolutionary approach.

Das et al., ICAC 2006
Outline

- Autonomic Computing and Multi-Agent Systems
- Utility Functions
  - As means for expressing high-level objectives
  - As means for managing to high-level objectives
- Examples
  - Unity, and its commercialization
  - Power and performance objectives and tradeoffs
  - Applying utility concepts at the data center level
- Conclusions
Utility functions for interacting power-performance agents

- How to trade off power vs performance?
  - In an individual machine
  - In a rack of machines
  - In an entire data center

- Formulate a joint power-performance utility function $U(\text{performance, power})$
  - Maximize $U(s_{\text{perf}}, s_{\text{pwr}})$
  - Often just $U(s_{\text{perf}}) - \varepsilon\ pwr$

- How to optimize $U$?

- How can semi-autonomous power and performance agents cooperatively optimize $U$?
  - Mediated through coordinator?
  - Direct bilateral interactions?

---

Kephart, Chan, Das, Levine, Tesauro, Rawson, Lefurgy. Coordinating Multiple Autonomic Managers to Achieve Specified Power-Performance Tradeoffs. ICAC 2007. (Emergent phenomena can occur when autonomic managers don’t communicate effectively.)

Hanson et al. Autonomic Manager for Power, NOMS 2010

Steinder, Whalley, Hanson, Kephart, Coordinated Management of Power Usage and Runtime Performance, NOMS 2008

Power-aware dynamic server consolidation

**Goal:** Save power by dynamically migrating VMs so as to occupy fewer servers without sacrificing performance too much. Turn unused servers off.

Maximize $U(\text{RT}, \text{pwr})$

Steinder, Whalley, Hanson, Kephart, *Coordinated Management of Power Usage and Runtime Performance*, NOMS 2008
Experimental results
(3 different utility functions)

1. Always meet SLAs
2. Always maximize performance
3. Permit 10% performance degradation for 10% power savings

Conclusions. Substantial power savings (up to 65%) can be attained without violating SLA. Results are significantly affected by utility function choice.
Outline

- Autonomic Computing and Multi-Agent Systems
- Utility Functions
  - As means for expressing high-level objectives
  - As means for managing to high-level objectives
- Examples
  - Unity, and its commercialization
  - Power and performance objectives and tradeoffs
  - Applying utility concepts at the data center level
- Conclusions
The Physical Infrastructure that Supports IT is Complex!

### Monitoring and Control
- Utility
- Facility
- Alternative Power
- UPS
- Battery
- Network
- IT
- Security

### Power
- Generator
- Parallel or Transfer Eqpt
- CHP Fuel Cell, MicroTurbine or Turbine
- Medium Voltage >600VAC Eqpt
- Low Voltage 600VAC Eqpt
- DC Power

### Cooling
- H2O
- Ice
- Pumps
- Chiller
- Computer Room Air Conditioners
- Central UPS
- Power Distribution Units

### IT and Networks
- Compute
  - Main Frames
  - Volume Servers
  - Blade Servers
- Storage
  - SATA Disk
  - Tape
  - Blended
- Network
  - Corporate Networks
  - VoIP
  - Integrated Blade/Switch
- In-Row Power
  - Modular UPS
  - Rack Mount PDUs
- In Row Cooling
  - Rear Door Heat Exchanger
  - Liquid Cooling Racks
  - Overhead Cooling
Trading off energy vs. temperature in a data center

- Cooling costs can account for ~50% of a data center’s energy consumption, due to zealous overcooling

- Let’s try using a utility function $U(E, T)$ to manage the energy-temperature tradeoff
  - Elicitation not trivial – we tried several forms, both multiplicative and additive

\[ U(E, T) = U_E(E) + U_T(T) \]
From utility to optimization

- **Elicit** $U(E, \{T\})$
- **Model** $E(\theta_1, \theta_2)$ and $T(\theta_1, \theta_2)$
  - Via experiments varying fan speeds
  - Could also run CFD calculations
- **Transform** utility to $U'(\theta_1, \theta_2)$
- **Optimize** $U'(\theta_1, \theta_2)$
  - Set fan speeds to $(\theta_1, \theta_2)^*$

Experimental measurements
How Optimal Fan Speeds Depend upon $T_{\text{max}}$

If ($T_{\text{max}} < 76\,^\circ\text{F}$)
\[ (\theta_1, \theta_2)^* = (60\%, 60\%) \]

If ($76 < T_{\text{max}} < 84\,^\circ\text{F}$)
\[ (\theta_1, \theta_2)^* = (0\%, >60\%) \]

If ($T_{\text{max}} > 84\,^\circ\text{F}$)
\[ (\theta_1, \theta_2)^* = (0\%, 60\%) \]

Energy savings = 10 to 12\%
Snorkels change the model $T(\theta_1, \theta_2)$; so the transformation to $U'(\theta_1, \theta_2)$ changes.

$(\theta_1, \theta_2)^*$ shifts from (60%, 60%) to (0%, 60%) for extra savings (12% → 14%)
Conclusions

- Utility functions help achieve the central goal of autonomic computing
  - “Computing systems that manage themselves in accordance with high-level objectives from humans.”
  - Theoretically well-grounded
  - Proven to work in practice in many domains

- Humans express objectives in terms of value

- Value is propagated, processed, and transformed by agents
  - Guides agent’s internal decisions
  - Guides agent’s communication with others

- Key technologies needed include
  - Utility function elicitation
  - Learning
  - Modeling / what-if modeling
  - Optimization
  - Agent communication, mediation
The next frontier? An autonomic data center economy

**Figure 1: The Data Center Market Model**

(Exploring market-based resource allocation for data centers.)
Backup
Multiplicative Utility Functions

- Administrator wishes to minimize overall energy consumption subject to a constraint on temperature
  - E.g. $T(x) < T_{\text{max}}$ at all positions $x$.

- Consider the multiplicative form: $U(E,T) = U_E(E) \cdot U_T(T(x))$
  - Energy utility $U_E(E) = \pi (E - E_0)$, where $\pi$ is $\$/\text{kW-year}$
  - Temperature utility $U_T(T(x))$ is a dimensionless step function, with the entire temperature distribution $T(x)$ as its argument
Dimensionless Temperature Utility Function
Practical Considerations

- Let’s think about $U_T(T(x))$ a little more
  - $U = 1$ if $T(x) < T_{\text{max}}$ for all $x$
  - $U = 0$ otherwise

- But we can’t measure $T(x)$ for all $x$

- One solution: just consider a finite set of measurements $\{T(x_i)\}$
  - Set could be readings from all temperature sensors
  - Or just the reading from a single representative temperature sensor $i$
  - Or just the maximum temperature in a region of interest (maybe entire DC)

- Example if we use many or all temperature sensors:
  - We can represent $U_T(T(x))$ as the product of scalar step functions
  - $U_T(T(x)) = \prod_i U_{T,i}(T_i)$
  - $U_{T,i}(T_i) = 1$ if $T_i < T_{\text{max}}$ ; 0 otherwise
Sanity check

- **Case A:** $T_i > T_{\text{max}}$ for all 10 sensors
  - $U_T(T(x)) = 0 \times 0 \times 0 \times 0 \times 0 \times 0 \times 0 \times 0 \times 0 \times 0 = 0$

- **Case B:** $T_i > T_{\text{max}}$ for just sensor #10
  - $U_T(T(x)) = 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 0 = 0$

Since $U_T(T(x))$ is 0 in Case A and B, utility $U(E, T) = 0$ in both

Yet most admins would prefer Case B to Case A!

How could we modify the utility function to prefer B over A?

One solution: *soften* the temperature constraint …
Modifying $U_T(T)$ to express a soft constraint

*Soften* the scalar step function $U_{T,i}(T_i)$

$$U_{T,i}(T_i) = \frac{1}{1 + e^{-\alpha(T_{\text{max}} - T_i)}}$$

- **Case A**: All 10 sensors at the same temperature
- **Case B**: 9 sensors at 75F; vary temp of sensor #10

Soft step function favors Case B over Case A
Further variations on $U_T(T)$

ASHRAE specifies both a minimum and a maximum temperature.

We can represent the scalar temperature utility as a two-sided soft step function.

$\alpha = 2$
Additive Utility Functions

- Administrator explicitly considers economic costs of energy consumption and temperature-induced equipment lifetime reduction.

- This suggests an alternative *additive* form: \( U(E, T) = U_E(E) + U_T(T(x)) \)
  - Energy utility \( U_E(E) = \pi (E - E_0) \)
  - Temperature utility \( U_T(T(x)) \) must now have same dimension: cost/yr
Cost-based Temperature Utility Function

Practical Considerations

- Somehow $U_T(T(x))$ must capture the cost of running equipment at $x$ at temperature $T(x)$

- Cost of device $i$ per year is $C_i / L(T)$
  - $C_i = \text{purchase cost}$
  - $L(T) = \text{lifetime if run consistently at temperature } T$
  - Inverse width $\alpha$ hard to ascertain from published data – widely different reports
    - Seagate drive lifetime reduced 4x for 35C increase in $T$
    - Google reports little degradation until 40C

- $U_T\{T_i\} = \sum_i C_i \left( \frac{1}{L_0} - \frac{1}{L(T_i)} \right)$
Power-aware load balancing

**Goal:** Save power by routing web traffic to minimal number of app servers w/o sacrificing performance too much.

Maximize $U(\text{RT, pwr})$

- **Perf Mgr**
  - Model: $\text{RT}(w, n; \lambda)$
  - Optimize: $U'(w, n; \lambda)$

- **Power Mgr**
  - Model: $Pwr(n)$

- **System**
  - $n$, $\text{RT}$, $\lambda$
  - $n^*, w^*$ (load-balancing weights)
  - $\text{On/Off}$, $n$, pwr, #cycles

---

Experimental Results

- **Elicit** utility function
  - \( U(RT) = 1/0 \) if SLA met/unmet
  - \( U(RT, Pwr) = U(RT) - \varepsilon Pwr \)

- **Model** (offline experiments)
  - \( RT (n; \lambda), Pwr (n; \lambda) \)

- **Transform**
  - \( U' (n; \lambda) = U (RT (n; \lambda), Pwr (n; \lambda)) \)

- **Optimize** (pre-computed policy)
  - \( n^*(\lambda) = \arg\max_n U' (n; \lambda) \)

- A few extra tweaks
  - Use forecasted \( \lambda \) to compute \( n^*(\lambda) \)
  - Add extra to \( n^* \) a bit to account for latencies (several minutes)
  - Heuristics to ensure that we don’t turn servers on and off too often

---

Avg Power Savings = 27%
No SLA violation
Multiplicative Utility-Function-Driven Cooling

$T_{\text{max}} = 80.6\,^\circ\text{F}, \alpha = 2.0$

Energy saved (kWh/yr)

Max Temp (F)

(60%, 60%)

Savings:
9.1 kWh, $8K/yr
(~12%)

Temp utility (max of all T)

$U(E, T) \, ($/yr)
Alternative Approach: Machine Learning

App Mgr can use reinforcement learning (RL) to compute external resource utility
- State = \( \lambda \)  
- Action = \( n \)  
- Reward = \( V(RT) \)

- It learns long-range value function \( V(state, action) = V(\lambda, n) \)
- It reports \( V(n) \) for current or predicted value of \( \lambda \)

Tesauro et al., AAAI 2005
Does Reinforcement Learning work?

Value table: Iteration 000

Animation
Commercializing Unity

Maximize Total SLA Revenue

5 sec

U(#srvrs)

U(#srvrs)

Demand (HTTP req/sec)

Trade3

WebSphere Extended Deployment

WebSphere Extended Deployment

Server

Server

Server

Server

Server

Server

Server

Server

Server

Server

Server
Utility-based Interactions between WXD and TIO: Step 1

Resource Allocations: n

- TIO cannot make well-founded resource allocation decisions
- WS XD can’t articulate its needs to TIO
- PoB not commensurate with utility

**Utility(current n)**

*Original WXD 5.1*
Utility-based Interactions between WXD and TIO: Step 2

Resource Allocations: n

- **WebSphere XD 6.0.2**

  ![Image](image-url)

  **Objective Analyzer**

  **TIO 3.1**

  **Policy Engine**

  - **TIO 3.1**
  - **Fitness(n)**
  - **PoB(n)**

  **Resource Utility(n)**

**Utility(current n)**

**Intermediate**

**WXD 6.0.2**

- WS XD research team added ResourceUtil interface of WXD
- We developed a good heuristic for converting ResourceUtil to PoB in Objective Analyzer
  - Interpolate discrete set of ResourceUtil points and map to PoB
  - This PoB better reflects WS XD’s needs

- Intermediate (commercially available)
Utility-based Interactions between WXD and TIO: Step 3

Resource Allocations: n

- We modified TIO to use ResourceUtil(n) directly instead of PoB(n)
- Most mathematically principled basis for TIO allocation decisions
- It enables TIO to be in perfect synch with the goals defined by WS XD
- Basic scheme can work, not just for XD, but for any other entity that may be requesting resource, provided that it can estimate its own utilities
Commercializing *Unity*

- Barriers are not just technical in nature

- Strong product line legacies must be respected; otherwise
  - Difficult for the vendor
  - Risk alienating existing customer base

**Solution**: Infuse agency/autonomicity gradually into existing products
  - Demonstrate value incrementally at each step

- We worked with colleagues at IBM Research and IBM Software Group to implement the *Unity* ideas in two commercial products:
  - Application Manager: IBM WebSphere Extended Deployment (WXD)
  - Resource Arbiter: IBM Tivoli Intelligent Orchestrator (TIO)
Visions of Autonomic Computing

Hal 9000, 2001

*Machines will take over all management tasks, rendering humans superfluous*

Terminator

*Machines will free people to manage systems at a higher level*

Wrong!

Right!
Finding the **optimal control parameters**

Even if service-level utility remains fixed, resource-level utility depends upon environment. Thus system responds to environmental changes.

\[ U'(cpu, b; \lambda) \]

\[ U(\text{RT}, \text{RPO}) \]

- \( \lambda = 0.002 \)
  - \( b^* = 0.875 \)
  - \( \text{cpu}^* = 2.49 \)
  - \( U^* = 152.7 \)
  - \( \text{RT}^* = 99.58 \)

- \( \lambda = 0.01 \)
  - \( b^* = 1.199 \)
  - \( \text{cpu}^* = 3.65 \)
  - \( U^* = 137.4 \)
  - \( \text{RT}^* = 95.44 \)

- \( \lambda = 0.05 \)
  - \( b^* = 2.053 \)
  - \( \text{cpu}^* = 8.58 \)
  - \( U^* = 75.9 \)
  - \( \text{RT}^* = 88.69 \)
Multi-agent management of performance and power

- We have explored using utility functions to manage performance and power objectives and tradeoffs in multiple scenarios
- Two separate agents: Performance and Power
- Various control parameters, various coordination and communication mechanisms
  - Power controls: clock frequency & voltage, sleep modes, …
  - Performance controls: routing weights, # servers, VM placement …
  - Coordination: unilateral, bilateral, mediated, …
- Examples
  - Energy-aware load balancing
  - Energy-aware server consolidation
  - Optimal power capping