Seamless Mobility – A Compelling Blending of Ubiquitous and Autonomic Computing

John Strassner
Fellow of the Technical Staff and Director, Autonomic Computing
Motorola Labs, 1301 East Algonquin Road, MS IL02-2240
Schaumburg, IL 60196 USA
Email: john.strassner@motorola.com

Associate Professor, Waterford Institute of Technology
Why the Interest?

• The complexity of system design and management keeps increasing
  ➢ Stovepipe systems: best-of-breed functionality but integration nightmares
  ➢ Exploitation of increased technology overwhelms users and admins with complexity
    » Different devices have different programming models and interaction paradigms
    » Different management tasks and integration types require different skill levels

• The complexity of business is also increasing
  ➢ People are demanding a pervasive presence
  ➢ Many types of businesses LOSE MONEY if they can’t react fast enough
  ➢ The varieties of threats, problems, and non-optimized behavior keeps increasing

• Component and system behavior complexity and interactions are increasing
  ➢ Everything is interconnected
  ➢ Different management actions require different policies and interaction paradigms

• Pre-defined behavior cannot be built to accommodate this complexity
  ➢ Too complex to predict, too high a skill level required, not enough people!

• We need self-governing components and systems
Ubiquitous Computing Definition

“Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user”

- Mark Weiser
Some Important Implications

• Ubiquitous Computing is a societal vision, not just a computing vision
  ➢ Must plan for how it affects a community

• A person will have many computing devices
  ➢ How do they cooperate?
  ➢ How does one not interfere with others?
  ➢ How is seamless operation between them achieved?

• Invisibility is hard
  ➢ Implies a consistent UI, or consistent interfaces
  ➢ Implies an ability to aggregate information and synthesize results
Autonomic Computing Definition

“Autonomic Computing encompass all aspects of making end-to-end user services self-governing in nature, automatically and completely implementing the users’ and owners’ high-level directives.”

- An autonomic system is a self-governing system, where the governance model is expressed using policies; policies are bound to business goals.
- Self-governance is accomplished through the use of self-knowledge to model the capabilities of the system and the constraints placed on the system as a function of context.
- A closed control loop enables the system senses changes in itself and its environment, analyzes those changes to ensure that business goals and objectives are still being met, plans changes to be made if business goals and objectives are threatened, executes those changes, and observes the result.
- This control loop is augmented by self-learning and reasoning processes, which enables the system to develop greater knowledge of itself and its environment, both by experience as well as by incorporating new knowledge.
Enabling Seamless Visualization
Now Imagine This Environment
Three Big Challenges

• Context-Awareness
  ➢ Applications operate within a given environment to perform a specific purpose per-user
  ➢ Interaction locality is important
  ➢ Requires proactive decisions and learning

• Task-Based, not Keyboard-Based, Actions
  ➢ User can supply data using a variety of means, such as speech and pointing

• Automated Data Synthesis
  ➢ Information from multiple sources is selectively aggregated to perform the user’s task
  ➢ Format and protocol converted for the user
Four Basic Functions Needed

- Supplying contracted functionality (service and/or resource and/or access…)
- Providing layered functionality via contracts to enable services to be easily aggregated
- Component collaboration to dynamically compose new functionality using contracts
- Providing federated functionality, such as peer-to-peer services, using contracts
Common Design Criteria

Who (Role):

Using CLI

Who is Authorized to Configure it

A Router

What (Thing):

<<concept>>

Why (Reason):

Because there was an SLA violation

Who (Role):

<<concept>>

How (Method):

Using CLI

Who is Authorized to Configure it

A Router

What (Thing):

<<concept>>

When (TimePeriod):

<<concept>>

Where (Location):

Its Physical Location

<<concept>>

When it can be changed

<<concept>>

107
Business to System Interactions

Business Process Model

Resource Mgmt System
- Determine Free Ports
- Generate WorkOrder
- Determine Route

Business Workflow System

Shared Information and Data Model
Knowledge Management

• Known knowledge
  ➢ Issue a command, expect a response
  ➢ Model enables validation of command and response

• Discovered knowledge
  ➢ New information that is not modeled in the system
  ➢ Need to identify its relevance and its effect on the system

• Autonomic systems need both!
  ➢ Modeling known knowledge enables system behavior orchestration
  ➢ Incorporating relevant unknown knowledge
    » Makes the model more relevant to the current application
    » Verifies that currently used policies and processes are correct
    » Allows context to drive behavior
Why Ontologies?

• Modern OSSs/BSSs need more than just a nifty network model
  ➢ Need to reconnect the Network with the Business
  ➢ Services need to be rolled out without adversely affecting deployed Products and Services

• Goal of ontologies is to represent any type of knowledge in a standard way
  ➢ Enable independently developed knowledge to be integrated
    » Ontology per knowledge domain, constituency, etc.
    » Mapping ontologies to a common form to exchange meaning
  ➢ Concentrate just on sharing network resource and service semantics
    » SID static models represent facts
    » SID dynamic models represent behavior using facts

• Complementary Purposes
  ➢ O-O models represent “facts”
  ➢ Ontologies enable inferencing about facts
  ➢ Integration reconnects the Business with the Network
Seamless Mobility Architecture

• Data, control and management planes working together to realize a common goal

• Persistence across seams requires consistent
  - user interactions across sessions (e.g., preservation of preferences)
  - access to applications, content, information & services

• Persistence Enablers
  - Invisible Management
  - Federated business operations
  - Multi-network operation of devices
  - Session handovers across networks and/or devices
  - Uniform ways to describe and exchange capabilities of components
  - Support for group interactions (workgroups, social groups)
  - End to end trust models
Seamless Manageability

Stovepipes
Best-of-breed systems
Device-centric

Managed Stovepipes
Cooperation through Integration
Middleware-centric

Paradigm Shifts Required:
Knowledge-driven decision-making and business-based governance model

Seamless Manageability
Context- and task-aware mgmt
User-centric, Business-Driven

THE FUTURE

Seamless Manageability

Autonomic Computing Element

- Policy Server
- Observe
- Plan
- Understand
- Learn
- Execute

Model-Based Translation Layer

Managed Resource

Managed Resource

Autonomic Manager

Observe

Plan

Learn

Execute

Today

Tomorrow
Autonomic Component Architecture

Component Context

Application Context

Solution Context

Environment Context

Component Capabilities and Constraints

Functional Aspect

Operational Aspect

Management Aspect

Control Aspect

Component Rules

Application Rules

Solution Rules
EVERYTHING is controlled by the Policy Server, which is distributed inside and out.

An enhanced NGOSS control loop.

Dedicated self-learning is part of each step of the control loop.

Use the DEN-ng Model to translate from vendor-specific data and commands to vendor-neutral equivalents.

Contracts

Don’t rely on instrumenting the resource directly!

EVERYTHING is controlled by the Policy Server, which is distributed inside and out.

An enhanced NGOSS control loop.

Dedicated self-learning is part of each step of the control loop.

Use the DEN-ng Model to translate from vendor-specific data and commands to vendor-neutral equivalents.

Contracts

Don’t rely on instrumenting the resource directly!
Key Points of Our Architectural Style

- NGOSS methodology emphasizes lifecycle development and management
  - DEN-ng models knowledge; code is generated based on the model
  - Contracts are the unit of interoperability; they are extended and morph in their lifecycle
  - Policy and process management used to control behavior
- Motorola Labs methodology emphasizes self-knowledge, learning, and reasoning
  - A system must incorporate new knowledge and understand it
  - Control loop governed through a selectable set of automated learning and reasoning behaviors
Summary

• Business, system, and behavioral complexity abound
  ➢ Complexity must be hidden from the user
  ➢ User must not be bogged down with management tasks

• Autonomic computing manages complexity by
  ➢ Representing the characteristics and behavior of managed elements
  ➢ Updating the model with new, non-modeled data

• Autonomics imposes a governance model
  ➢ Orchestrates behavior through policy management
  ➢ Enables context changes to drive network resources and services
  ➢ Business rules and goals drive network resources and services