Autonomic Architecture to Support Next Generation Services

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

Keywords: architecture, autonomic, complexity, information model, next generation network, next generation service, ontology

Abstract

This presentation describes developing work between Motorola Labs and Waterford Institute of Technology concerning the development of an architecture to support autonomic operation of current and future services.

Slide 2 defines the fundamental problem preventing this as an increase in system, business, and behavioral complexity. Slide 3 focuses on the ever-important issue of convergence, and defines it as consisting of service, device, and network convergence. Slide 4 provides an example of the compelling nature of seamless mobility, and slide 5 further motivates this by summarizing some of the problems that a converged architecture will face when a variety of devices and content is provided.

Slide 6 describes the salient features of the DEN-ng (Directory Enabled Networks – next generation) information model, which is both the root of the TMF SID (Shared Information and Data model) and extensions to it. The DEN-ng model is best suited to this task because of its UML (Unified Modeling Language) compliance, rich usage of patterns and roles, and most importantly, its use of finite state machines to model behavior. Slide 7 discusses MDA, and points out some problems in its current definition that this research has identified and is trying to solve. Slide 8 introduces the notion of ontologies. Our approach is innovative, in that we use ontologies to provide a common set of meanings and semantics. When combined with information models, this enables new knowledge to be analyzed and incorporated into the model that is driving the system.

Slide 9 explores two key points – what gets persisted across seams, and enablers for that persistence. Slide 10 describes our model-driven Autonomic Diagnosis Engine, which is the heart of the architecture. Slide 11 our autonomic element, which is our building block for building an autonomic system.

Finally, Slide 12 describes future work, and a set of references for further study.
Why the Interest?

➤ System complexity keeps increasing
  - OSSs are built in a stovepipe fashion, which offers best-of-breed functionality but causes an integration nightmare
    » Different device programming models; different tasks require different skill levels
    » Increasing technology brings more complex systems
    » Users and administrators have to deal with the consequences!

➤ Business complexity is increasing
  - People want a pervasive presence and seamless access to content
  - Many businesses LOSE MONEY if they can’t react fast enough

➤ Behavioral complexity is increasing
  - Too complex to predict; too high a skill level required
  - Not enough people to do the job!

Complexity affects system functionality as well as how business is conducted. In autonomic systems, increased system, business, and behavioral complexity require system operation and functionality to be adjusted dynamically. Current Operational and Business Support Systems (OSSs and BSSs) use stovepipe systems, to enable best-of-breed functionality to be incorporated. Stovepipe systems resulted from the inexorability of Moore’s Law, which encouraged programmers and architects to exploit increases in technology to build more functional systems. Unfortunately, administrators and end-users of the system must then accept the increased complexity of system installation, operation and maintenance. In addition, they present several serious integration problems, one of which is the redefinition of the syntax, semantics, and structure of what should be common data. This happens because each stovepipe system views itself as the “center of the universe”, and hence doesn’t consider common meanings and structure of data. This prohibits the sharing and reuse of that data.

The increase of business complexity adversely affects a main goal of end-users, which is simplicity. Users are demanding increased connectivity; however, this results in much more complex interactions to take place. It is becoming impossible to predict how various entities will interact with each other in a system. Worse, the complexity of each component is growing so fast, that soon even the most skilled administrator will not be able to manage them. Ubiquitous computing, for example, motivates the move from a keyboard-based to a task-based model. This requires an increase in intelligence in the system, which is where autonomics will come in.

We need a set of tools to manage this increasing complexity. Self-governing behavior of autonomic systems will enable a better, more efficient, set of tools that enables network services and resources to adapt to changing demands.
NGN Architecture IS Convergence

“Convergence: The Act of Converging and Especially moving toward union or uniformity…” Merriam-Webster Online Dictionary

Seamless Communications requires digital convergence of three key technology areas to enable services/applications across multiple access technologies and devices

- **Service Convergence**: End user accessing the same service using one or more devices through one or more access mechanisms. No matter which device is being used, the service content and presentation is adapted according to the capabilities of the device.

- **Device Convergence**: End user accessing multiple services across multiple access mechanisms through a single device. No matter which service is being used, the device capabilities are adjusted to maximize the utilization and presentation of the service.

- **Network Convergence**: A unified network realizing multiple services using one or more devices through one or more access mechanisms.

The above three types of convergence must work together in order to provide a truly seamless experience for the end user.

Webster’s dictionary defines convergence as the act of converging and especially moving toward union or uniformity. In our industry, this means that three key areas are required to move in union or uniformly to enable the different environments that users operate in to provide a seamless experience to the user:

- **Service Convergence**: End user accessing the same service using one or more devices through one or more access mechanisms. No matter which device is being used, the service content and presentation is adapted according to the capabilities of the device.

- **Device Convergence**: End user accessing multiple services across multiple access mechanisms through a single device. No matter which service is being used, the device capabilities are adjusted to maximize the utilization and presentation of the service.

- **Network Convergence**: A unified network realizing multiple services using one or more devices through one or more access mechanisms.

The above three types of convergence must work together in order to provide a truly seamless experience for the end user.
Enabling Seamless Visualization

Here, we see a seamless transition from a TV to a PC to ultimately a cell phone. This enables the user to access content using whatever device is handy to them at a given point in time and context. This user friendliness results from increased intelligence, not derived through one “master policy server” but rather through a collection of intelligence spread throughout the system.

The obvious advantage of this example is that the content is no longer bound to a particular location or device – it can follow the user and be viewed and listened to using a variety of fixed and mobile devices.

Balanced against that is Asimov’s vision of “I Robot”, the movie about homicidal domestic robots that try to take over the world. While that is somewhat fanciful, increased intelligence has many different connotations – some good and some bad. For example, how do we ensure that this increased intelligence doesn’t result in the device learning bad habits?
This is a user-centric environment, enabling the user to communicate anywhere using anything. In particular, notice that there is now a multiplicity of input and output devices, enabling information and commands to be conveyed by and to the systems in a variety of ways.

But for the system administrators and operators, it represents a large challenge, due to the different protocols and functionality (e.g., processing capabilities and data display) of each device. These factors are daunting, but there are others that are even harder to quantify, such as the amount of trust assigned to a given sensor or type of information.

How do we remove the barriers between these different sources of information and enable them to communicate, share, and exchange data seamlessly?

Autonomics 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 can be supplied by humans or derived by machines. 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. This takes the form of a closed control loop – one in which 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 a self-learning process, which enables the system to develop greater knowledge of itself and its environment, both by experience as well as by incorporating new knowledge. Autonomic systems can be dynamically and automatically built from reusable autonomic elements, rearranged as necessary to provide new functionality. Autonomic systems are used to manage business, system, technical, and operational complexity. This also enables progressively greater adaptive systems that can dynamically respond to current and future needs.
Advantages of the DEN-ng Models

- DEN-ng views the world in terms of:
  - **Capabilities** - normalized functionality
  - **Constraints** - restrictions on what you can use
  - **Context** - environment in which different objects operate
- DEN-ng is built using **patterns** and **roles**:
  - Patterns make the model inherently extensible
  - Patterns are used to capture common relationships and occurrences of physical connections and structures
  - Roles (and other abstraction mechanisms) enable models to be reused for different applications
- DEN-ng uses a **finite state machine**:
  - Other models represent the current state of an object
  - DEN-ng builds different models of different states, and binds them together using a finite state machine

We need a consistent representation of knowledge. This paper uses the DEN-ng information model to represent knowledge. DEN-ng was built by extending the UML Metamodel to add concepts for representing behavior in general and autonomic computing in particular, such as contract-defined behavior, policies, and processes.

Focusing on network devices, the syntax and semantics of the language used by each vendor to control the functionality of a network device is different. Worse, the programming models used in different devices are different. Note that this is true even for different devices manufactured by the same vendor. Other information models in the industry, such as the DMTF’s Common Information Model (CIM), IETF MIBs (Management Information Bases), and various ITU M and G recommendations (models specifying various aspects of networks, such as transport), are limited to modeling the current state of a managed element. In contrast, DEN-ng models the lifecycle of components and solutions. Hence, it contains static as well as dynamic models that are linked together using the notion of a set of finite state machines (FSMs). Each FSM models a set of components that are defined as a reusable block of functionality. This functionality is normalized using capabilities, and described using patterns and roles. Constraints are also modeled – when combined with capabilities, this enables an intelligent view of what functions you can and can’t use as a function of the role of the client and/or current environmental conditions (among others). This, and other design points, are defined as the context of operation. This approach is unique in the industry.

In other words, DEN-ng is used to represent the characteristics and behavior of heterogeneous devices using a single, standardized language. This enables code to be automatically generated from the models. DEN-ng has mined information from these other works, and others as well (e.g., ebXML, where some limited patterns are used) and incorporated many of their concepts into the SID. This mining is done using an ontology. The ontology establishes a common set of meanings for information, including how data is related to other data. Without an ontology, the system only has

The advantage of DEN-ng is that, through its uses of abstractions (capabilities, roles, and others), it can build a set of reusable behaviors that represent interactions between entities in the form of a finite state machine. Thus, code can be generated to enable the system to implement specified behavior. This cannot be done with current approaches, since dynamic aspects of the system are not modeled.
Is MDA the Solution?

- Captures desired behavior through modeling
  - Code generated, driven by the model, to respond to changes
  - Time-consuming, error-prone manual tasks can be automated
- But – MDA has a LOT of complexity
  - Lack of patterns, hard to implement
  - Difficult to decouple the logic from how it is persisted
- COTS products unable to handle this level of complexity
- UML is a great standard, but
  - Symbology is currently deficient in many areas (e.g., Use Cases)
  - Not suited to representing certain tasks (e.g., policy)
  - Certain constituencies don’t like UML!
  - PIM-PSM is a poor concept

Model Driven Architecture (MDA) defines an architecture, expressed as a set of models, that separates the specification of system functionality from the specification of the implementation of that functionality. In MDA, platform-independent models (PIMs) are initially expressed in UML. Such models can be subsequently translated to platform-specific models (PSMs) through a set of mapping functions. There are a set of standards that support MDA, but these are beyond the scope of this paper.

Fundamentally, MDA seeks to integrate, through formal modeling, the entire lifecycle of a system: from business modeling to system design, component construction, through implementation and deployment. By providing this traceable evolutionary path, MDA enables the system to further evolve. For more information, please go to www.omg.org/mda.

However, there are some problems with using MDA as it is currently defined. Since we are interested in generating code, we’ll focus on three major code-specific problems.

First, existing MDA tools have a LOT of complexity. Most distressing is the lack of patterns and various abstraction mechanisms that were used to build DEN-ng. One has to wonder, if these concepts were good enough to construct the model with, why not use them in the code? Off the shelf code generators built to the MDA specification were generally unable to handle the complexity of real-world applications, let alone autonomic ones. More importantly, quality assurance mechanisms are missing, implementation patterns are missing, and most importantly, these generators lacked a set of abstraction mechanisms to control how the code was generated and to enable it to be fine-tuned. Finally, we wanted to separate the logic from the mechanisms used to persist data used by the logic.

There are other problems, such as the lack of concrete notions of policy management and process automation. More importantly, the concept of PIMs mapping to PSMs isn’t granular enough. What’s really needed is a mapping between different levels of abstractions – whether this is an information model to a standards-based data model, or a standards-based data model to a vendor-specific data model. In my book Policy Based Network Management, I define this as model mapping.
Why Ontologies?

- A standard way to represent different knowledge
- Enable independently developed knowledge to be integrated, and new knowledge to be integrated
  - Concentrate just on network resource and service semantics
- Complementary Purposes
  - Facts represented by models; Inferencing done by ontologies
- A common set of semantics and meanings
  - Knowledge can be harmonized using a shared vocabulary at different levels of abstraction
- Insight into the behavior of a given domain
  - Which data is relevant, and what other data to look for, depending on context
  - How and when to extend the model

An ontology is a common set of terms and relationships that are used to represent a domain of knowledge. Ontologies provide a common vocabulary and definition of rules for use by independently developed resources, processes, and services, which enables agreements among people and organizations sharing common resources and services to be made by expressing relevant concepts unambiguously. For example, it can be used to relate concepts in two different knowledge sources, such as a MIB and a CLI, or equate the business concepts of an SLA with the networking concepts of a link failure.

Thus, an ontology is the basis for sharing information between disparate systems. There are two types of knowledge that is of interest to the managed system – known knowledge (which can be modeled beforehand) and unknown knowledge. An example of the former is when a command is issued. Since the semantics of the command are known as well as its effect, the nature of the response is also known. An example of the latter is newly discovered knowledge, which is not yet modeled in the system. Autonomic systems need both! Modeling known knowledge enables the behavior of entities to be orchestrated and controlled so long as the behavior is that which is either expected or anticipated. Incorporating relevant unknown knowledge makes the model more relevant to the current application. For example, it verifies that current policies are correct, and enables the system to adjust to changes.

Given the wide diversity of management information, coupled with the different uses of different types of management information, our approach uses a multiplicity of ontologies, one for each domain that is relevant in the managed environment. Ontologies are used to establish semantic similarity (or degrees thereof) between the information sources. Note that degrees of semantic similarity enable various assertions to be made about the changing state of the system which, when compared to the current state of the system, enables us to adjust its behavior according to the underlying model.

The cost of establishing semantic similarity between different information models is high at first, and then tapers off as the result of the mappings increase to establish a more powerful mapping base. It is also dependent on subject matter experts to do the modeling. We are currently working on automating the semantic mapping, so that cost will decrease in time.

However, we must also be cognizant that nothing in life is free! There is no one unifying standard, and there likely never will be, just as there will never be a single programming language or protocol. Furthermore, it is important that we enable our system to be used with existing systems with their own knowledge sources. Hence, we felt that the above cost is justified.
An architecture for supporting next generation services, such as Motorola’s seamless mobility, requires persistence of information and context to be preserved across disparate components of the architecture (called “seams” in Seamless Mobility). This in turn requires a set of technology enablers.

Consistent user interactions that are intuitive and easy to use are not enough – they must be able to adapt to the environment (i.e., take the task, location, device, and other parameters into account). This includes preferences that are shared across devices and boundaries. Consistent access to applications and services means that authentication and access rights persist across devices and networks, enabling application interaction to remain consistent across devices. Universal access to content and information means that acquiring, using and sharing content appears “continuous” based regardless of device or location. This means that functionality like access policies must remain consistent across seams.

In order to enable persistence across multiple seams, devices must be able to operate over multiple networks. This enables session handovers to take place on demand, whether it is wired to wireless, packet-based to circuit-based, or another form. To do this, management must be not just invisible, but also autonomic. We need networks that are self-governing. Discovery of their capabilities and functions is a service; managing their configuration must be a service performed on demand, when it is needed.

Most importantly, the operation of the business needs to be federated. This means that the barriers between different businesses and business units blur, and becomes a single, atomic whole. Hence, identities and policies are shared, and services and resources are aggregated and delivered on demand, where they are needed.

These mechanisms need to support end-to-end trust models that apply at the individual, group, and business level. Seamless operations can apply to an individual, a group of people that is either designated by a business function or forms dynamically, and other group operations, so the architecture needs rules to govern various types of groups.

The above is not possible without continual monitoring and control, in order to ensure the timeliness and reliability of notification delivery of functionality and changing events.

There are two ways to build a complex system. The first is to build it as a single monolithic entity. The second is to build behavior from collections of elements that each do a simpler set of tasks very well, and build complex behavior from simpler behavior. The latter is one of the most important points of autonemics, and is strongly related to biology and sociology. Rather than building a very complex design that is housed in one single unit, autonemics teaches us to build a set of simpler components and put them together to achieve the desired set of functions.

Our architecture provides a self-governing system, whose governance model is expressed using policies. Policies can be supplied by humans or machines. Self-governance is accomplished through the use of self-knowledge to model system capabilities and environmental constraints as a function of context. This takes the form of a closed control loop – 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, and executes those changes, and observes the result.
This is a block diagram of Motorola’s approach to defining an autonomic system using a programmable grid of Autonomic Elements (AEs). In essence, we wrap existing resources and services using a façade driven by the DEN-ng model. This enables existing resources that don’t allow new software to be inserted to control their operation to still be controlled by the autonomic system. This is a very important consideration, as it ensures that legacy devices, services and operations can be either incorporated into an autonomic system, or slowly replaced over time. In other words, instead of relying on embedding sensors and effectors into the resource, we build a translation layer, based on the model, that translates vendor-specific data and commands to a vendor-neutral form that is used by Motorola’s autonomic software. Note the similarity of this problem with our original OSS problem!

Autonomic systems will of course use advanced languages and protocols to realize autonomic operation. The Model-Based Translation Layer enables these new languages, protocols and interfaces to be translated to a form which specific vendor devices can understand. In particular, it enables data and commands in multiple distinct formats from multiple different vendors to be harmonized using a combination of object-oriented information modeling and semantic knowledge. This enables the meaning of received data and their significance to the operation of the system to be better understood. Thus, a single autonomic computing system can get data from and issue commands to different devices, even if those devices cannot communicate directly with each other.

This approach builds on the TeleManagement Forums NGOSS program. In particular, since the DEN-ng model is used, the interaction of contracts and policies with other modeled entities are naturally defined because all are defined using the DEN-ng model. Contracts are used everywhere information is exchanged between components, which standardizes this process. Policies are used to formalize how decisions are made in the system.

Note the use of the model in this approach. Its use is not restricted to one portion of the control loop; rather, its use is pervasive. Similarly, policy isn’t restricted to just the planning stage; rather, its use is pervasive as well. For example, policies can be used to adjust what is being observed and learned, as well as provide richer execution mechanisms. Of course, policies are critical for controlling how new information is analyzed, understood, and incorporated.

Finally, note that each function in the control loop is governed explicitly by a Policy Server. Policy is used to govern how decisions are made, not just in the control loop process, but also throughout the entire autonomic system. For example, policies control how autonomic elements communicate, bind together to form collections of resources and services that are composed and dissolved on demand, and many other functions.
Motorola Labs Is Building an Autonomic Grid

This is a block diagram of Motorola’s approach to defining an autonomic system. The first thing to note is that Motorola Labs is building a grid to provide autonomic functionality. The reason why this approach was chosen is deceptively simple: if we can program the grid, then we can enable different resources and services to be assigned and removed dynamically to provide different functions for different users. We can also compose more complex functionality from simpler resource and service elements. Most grids are built using homogeneous computing elements; this architecture uses heterogeneous computing elements because they more closely conform to the differing functionality provided by network elements.

Each element in the grid has the same autonomic manager, although the functionality of the Observe, Understand, Plan, Execute, and Learn processes used (as well as the functionality of the model-based translation layer) vary according to the functionality of the managed resource. The Observe portion is responsible for monitoring the state of the Managed Resource, as well as for gathering the correct data as directed by the Autonomic Manager (e.g., to prove a current hypothesis). The Understand portion is responsible for understanding the environment through understanding sensor data and received events. This is done by augmenting received information with semantic data, based on the current state of the Managed Resource. The Plan portion is responsible for evaluating any differences between the current functionality of each AE in the system and its optimal functionality. The Execute portion is responsible for ensuring that the business goals of the system are met in an optimal manner by each AE. The FSM defines the set of states that the Managed Resource needs to transition through to correct and/or optimize its behavior; policies define the correct sequence of actions to be taken to achieve that state transition. Finally the Learn portion is responsible for studying the operation of each of the above four components, correlating actions taken with observed conditions, building up a knowledge base so that it can suggest better actions in the future, and making decisions using various types of learning behaviors, hypothesis and theory generation, and abductive, inductive, and predictive processing.

The autonomic element of this architecture can intelligently process data based on policies, which are distributed throughout the system. This not only protects against any single autonomic element failing, it enables each of the autonomic elements to distribute important learned information to the rest of the system (note the strong similarity between this approach and hive/colony philosophies).
Conclusions and Future Work

- Complexity is increasing; next generation services will exacerbate this
- Knowledge must be unified – we use a combination of information models and ontologies
  - Underlying unified information and data models based on DEN-ng
  - Ontologies enable the semantics of different data to be understood
  - New knowledge can be harmonized and integrated
- Autonomic building blocks are based on collections of smart functions
  - System changes its offered functionality in response to changing user needs and environment
  - Behavior is orchestrated according to business goals
- Challenges for implementing seamlessness
  - Making computing resources and their management invisible
  - Aggregating required information on demand
  - Adapting services and resources provided based on context
  - Customer hasn’t complained $\Rightarrow$ all is OK vs. Instrumentation indicates SLA violation!
  - Automating the semantic similarity process

Important conclusions are: (1) modeling and ontologies are well-suited to providing a consistent, coherent, extensible knowledge base; (2) both are needed in order to incorporate new knowledge; (3) New knowledge can be used to assert beliefs about how the system is operating, and test whether the current set of policies and processes are optimal or not; (4) autonomic systems are best made up of a collection of individually simple, yet collectively intelligent, components.

Future work will center on: (1) formalizing a means to embed knowledge in and around elements, and to realize efficient communication between a set of changing network elements; (2) correlating information on demand to drive the reconfiguration of services and resources, based on context; (3) rationalizing different beliefs about the system (e.g., why the Customer isn’t receiving the Service contracted for); (4) automation of knowledge discovery, integration, analysis and integration.

I’d like to acknowledge Dr. Sven van der Meer and Michäel Ó Foghlú from the Waterford Institute of Technology for their insightful comments and suggestions.

The following are recommended references:

**TMF Documents**
- GB927, NGOSS Lifecycle and Methodology, TMF053, Technology Neutral Architecture (and Addenda)

**Other Documents**
- Model Driven Architecture: www.omg.org/mda
- Zachman Framework: www.zif.a.com
  - For patterns, please see: M. Fowler, “Analysis Patterns – Reusable Object Models”, ISBN 0-201-89542-0
- Strassner, J. and Reilly, J., “Learning the SID”, Tutorial for TMW 2005 Spring Conference
- Strassner, J. and Reilly, J., “Using the SID”, Tutorial for TMW 2005 Spring Conference
- Strassner, J., “Modeling and Managing the Lifecycle of Telecom Systems”, CITSA 2004
- Strassner, J., “MDA for Telecom Systems Using DEN-ng”, ICETE 2004 Conference