A Domain-Level Data Model for Automated Network Configuration

Keith A. Weinstein
MITRE Corporation
El Segundo, CA, USA

Waylon Wang, PhD
MITRE Corporation
San Diego, CA, USA

Kurt M. Peters, PhD
MITRE Corporation
Colorado Springs, CO, USA

Daniel P. Gelman
MITRE Corporation
San Diego, CA, USA

Jim Dimarogonas, PhD
MITRE Corporation
Mclean, VA, USA

***Abstract*— Emerging research activities and technologies such as NETCONF have been closing gaps in the realization of automated network management; however they focus on individual network devices rather than the network domain as a whole. A robust model that maps a network domain's aggregate properties to device-specific NETCONF entities would allow a service provider to abstract a network's design from the configuration of its devices. Furthermore, it would extend the automation capability of a network management system by enabling configuration operations in terms of high-level concepts such as aggregate policies, best practices, network topology, and service requirements. One community that would benefit from this paradigm is a joint tactical network service provider, who must configure networks consisting of heterogeneous subscriber bases, network devices, and organizational policies at moment's notice.**

Keywords- Network management; automated network configuration; enterprise architecture; NETCONF; knowledge-based system

I. INTRODUCTION

Automated network management is a growing research area that has yet to be fully realized. In the context of the FCAPS (fault, configuration, accounting, performance, security) management model [3][4], one of the most critical and complex functions to automate is configuration. Configuration management is vital because in most cases, a device's configuration reflects the implementation of all other functions. Configuration management is also extremely complex; in an enterprise environment, a network device's configuration options and format vary according to its make, model, operating system, and any installed software or hardware modules. This complexity is further increased by the fact that manual command-line interface (CLI) entry is still the most common method of editing and applying configurations; network administrators must become proficient in the command-line syntax of every platform they operate [1][2].

In a commercial environment, there is little utility in automating the configuration of network devices; thus the business incentive to develop automated network configuration software is not significant. This is primarily because commercial service providers already have the means to mitigate the complexity associated with configuring network devices. They can choose device vendors with homogenous operating systems and command-line syntaxes for which they are already trained. When planning new networks, they allow ample time for shaping their management environment, testing and measuring service-level agreements (SLAs), employing management software tools, and developing custom scripts that automate recurring tasks.

However, there are some cases, such as those in a joint tactical military environment, when these options are not available. Tactical network service providers face the challenge of quickly configuring networks whose components each have different subscriber requirements, device platforms, topologies, security policies, and IP addressing schemes that can change at moment's notice. The urgent nature of combat missions suggests that network operators should implement these configuration changes in near real-time to minimize service disruption; yet this is not humanly possible. It is in these environments where the need for an automated network configuration solution truly exists.

Although an end-to-end automated configuration solution is still considered a future concept, there are emerging technologies that may speed up its realization. One such example is the NETCONF protocol, defined in RFC 4741 [5]. NETCONF simplifies remote configuration operations by employing XML messages that specify the remote procedure call (RPC) transactions between a manager and device. Network device vendors can then publish data models for their configuration information as XML schema, thus ,eliminating the need to implement complex regular expression parsers for platform-specific command-line syntax. While NETCONF simplifies the format and delivery of individual network device configurations in a consistent manner, however, it does not address the issue of aggregating them into a unified configuration model for an entire network domain.

In order to progress towards an end-to-end automated configuration solution, we believe that the next logical step is to develop a robust network configuration data model that applies to a service provider's entire network domain and provides a high-level means of determining configurations for its devices. In this paper, we create a framework for modeling an automated system that computes device configurations from

a set of high-level inputs such as subscriber requirements, device topology, IP address resources, and security policies.

In the remainder of this paper, we summarize other work in Section II that relates to our contributions. Section III describes the methodology and provides examples that demonstrate the concepts of a domain-level data model that can be used as the basis for an extensible, modular framework for automating network configuration. Section IV presents a view of the realization of the model defining the detailed data structures and the associated operations for computing a set of configurations. In Section V we propose the future work for a system implementation that automates network configuration through the use of the proposed domain level data model.

II. Related Work

A semi-automated tool called GeNUAdmin[8] system extracts network configuration information into a centralized database, performing updates on that database which are checked for consistency, and pushing the changes back into their respective configuration files. Simple consistency checks are performed to assure that added values are valid and that key values are unique.

A. V. Konstantinou, Y. Yemini and D. Florissi introduced a novel method for autonomic management organization, element instrumentation, and policy maintenance [7] and [9]. Management functions are organized in a two-layer architecture. The bottom layer organizes management information in a unified object-relationship model that is instantiated in a distributed transactional object modeler repository. The top layer unifies the traditional roles of managers and elements into a single management layer. Network devices use the modeler as a primary management repository, and effect autonomic behavior in terms of transactions over the shared model state. A language called JSpoon was introduced as a mechanism for extending element objects at design-time with management attributes and data modeling layer access primitives. JSpoon elements may be extended with additional autonomic functions at runtime using model schema plug-in extensions. They have further introduced an autonomic policy model and language in the form of acyclic spreadsheet change propagation rules, and declarative constraints. An Object Spreadsheet Language (OSL) was used to express autonomic behavior as dynamic computation of element configuration over the object-relationship graph model. The proposed organization has been implemented in a prototype system called NESTOR and demonstrated to various customers and sponsors including DARPA.

Our work focuses on domain-level modeling of the requirements, resources, policies and best practices that can be mapped to device-specific NETCONF statements, aiming at automating the entire network configuration process.

III. Methodology

In developing a network configuration data model, we consider an ontology-driven framework that defines a network domain from the service provider's perspective that enables the user to define the network resources and constraints without intimate knowledge of the underlying configurations of the individual devices We also define a system level perspective that implements a knowledge-based system consisting of facts, rules and variables that can be used in an inference engine to compute device configuration solutions .

A. Scope

When designing a system that automates a complex task such as configuring all devices within a network domain, it is critical to first clearly define the system's goals and the scope in which it operates. Given the requirements outlined in the introduction, we first define the following terms that form the context of the system:

- *Service Provider.* The service provider is the entity responsible for managing a network. In the context of network configuration, a service provider is responsible for configuring a set of network devices in a known topology that satisfy both the subscriber's information exchange requirements (IERs) and any network policies mandated by the service provider's organization.
- *Network Domain.* We define a network domain as the scope of responsibility for a service provider. This includes all subscribers, hardware resources (such as network devices and their physical links), logical resources (such as IP addresses, autonomous system numbers and domain namespace), and network boundaries (such as peering and transit points with external service providers).
- *Configuration State.* The configuration state of a network domain is the set of device configuration settings for a service providers network domain.. This is effectively the solution computed by the automated network configuration system. When inputs such as subscriber requirements, policies, or topology data change, the data model must transition to a new configuration state.

In this context, we define the user of our automated network configuration system as the service provider, and the purpose of the system is to compute a configuration state for the network domain. Figure 1 depicts the context of this perspective.

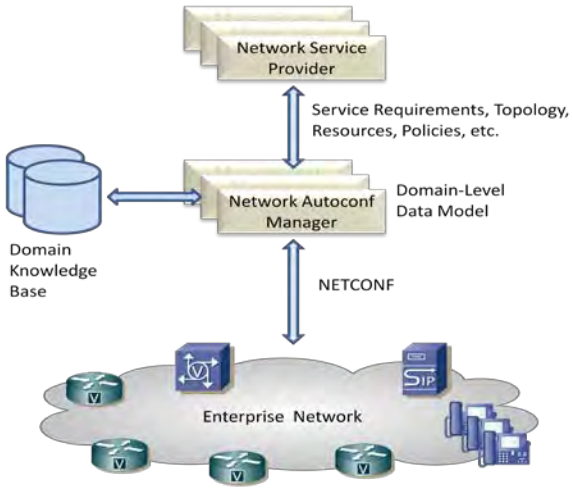

Figure 1. Context of the Domain-Level Data Model

B. Modeling Approach

In developing the data model, we provide two perspectives of the system's input. The first perspective represents a user view of the network domain. Because the service provider is the primary user of this system, the structure of all inputs must be presented in a manner that appears rational and practical for input by a service provider. The second perspective provides a system view of the data. In this view, we represent the data as a knowledge base of facts and rules that satisfy a set of goals. Under this perspective, the complex relationships between all entities become a set of constraints for which the system must find a solution. The solution itself then becomes the set of device configuration settings represented as literal values.

From the service provider's perspective, the configuration state of the network can be defined by domain level abstractions such as the network topology, service requirements, logical resources, and policies, supplemented with a set of best practices and domain knowledge applicable to network configuration. Figure 2 demonstrates the relationship between these entities.

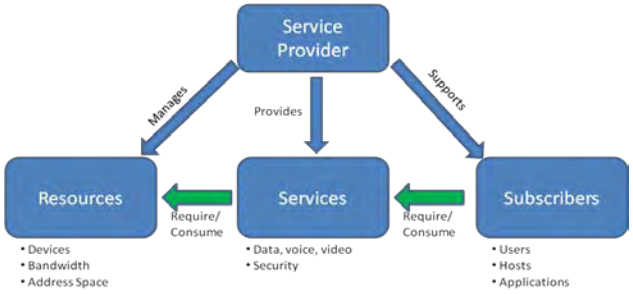

Figure 2. Service Provider's Perspective

1) User (Service Provider) Perspective

As any software development effort typically begins by defining user requirements, it logically follows that the initial data should be characterized as the user's input. Because the intent of our system is to allow a service provider to abstract the high-level network properties from the details of the configuration state, the inputs should be characterized accordingly. Figure 2 shows the logical model of the elements and relationships within a network domain. Based on this we identify five main entities: service requirements, network topologies, logical resources, network policies, and domain knowledge.

a) Service Requirements

The root of all network design can be traced to the set of requirements imposed by the network's subscribers. At a very high level, these are often referred to as IERs. IERs define the content of data that must be exchanged between actors of a domain. In the context of network configuration, however, we are only concerned with the information needed by the network to implement a specific IER. We use the term service requirements to describe this information. Service requirements would likely comprise the following data:

- *Subscriber communities:* the grouping of end hosts represented as source and destination pairs (or in multicast or broadcast instances, a source may have multiple destinations). Hosts within a subscriber community should also be described by their physical location, such as the nearest network device and interface from which they connect.
- *Communication protocols.* These are the protocols required to support circuit-switched or packet-switched service between members of a subscriber community. In the context of TCP/IP communications, this may include the transport protocol (TCP or UDP), and the source/destination ports.
- *Service quality*. Service requirements may be further refined to include qualitative characteristics such as data rates, priority, guaranteed service, or availability rates.

b) Network Topology

We define the network topology as the set of information that describes all network devices, their interfaces, and the physical links that connect them. This information is necessary to determine which devices will require configurations and how they will be utilized to support service requirements. The primary entities of a network topology include:

- *Devices.* All network devices to be configured by the system must be declared. The devices should also be described by their capabilities and the set of configuration options they can hold.
- *Interfaces*. In this context, we refer to the physical interfaces of a network device (such as an Ethernet interface). A physical interface typically supports a set of link-layer and physical-layer protocols; this knowledge must be available to the system.
- *Links.* Here we define the physical connections between devices (further specified by the exact interfaces) and their characteristics. These characteristics may include protocols, data rates, and transmission media.
- *Sites*. We define a site as a geographically local grouping of devices (and internal links) from which a

set of subscriber hosts obtain their service. Typically, a site's boundaries are its Wide Area Network (WAN) links.

c) Logical Resources

Most telecommunication protocols require a set of logical identifiers in order to function properly. These include IP addresses, autonomous system numbers (ASNs), virtual circuit identifiers (in the case of ATM or Frame Relay), DNS namespaces, and any other resource that can be logically assigned to hosts, circuits, or interfaces. An effective configuration system should be able to dynamically allocate these resources in accordance with the constraints of the service provider's network policies.

d) Network Policies

Network policies are constraints imposed by the service provider (or the service provider's parent organization). They further restrict the set of available configuration options. Examples include:

- Security policies. These include any restrictions on device protocols, subscriber communications, or even topology. For example, a policy dictating that all subscriber hosts must reside behind a firewall is a topology restriction.
- Best practices. These are policies that may not be explicitly required but can further constrain the configuration options if multiple solutions are available. One example of a best practice is to allocate an additional 20% of IP addressing space for a user LAN to anticipate growth. Another best practice may require that the site name be appended to the host name of a device.

e) Domain Knowledge

An automated configuration system must be able to apply intrinsic rules that reflect the domain business logic. In the context of network configuration, this includes all basic knowledge of concepts such as routing protocols, device capabilities, IP addressing rules, and anything dictated as a standard. Any network standards defined in a Request for Comments (RFC) would be represented as part of the system's domain knowledge. The sum of all these facts and rules within the network is the domain knowledge.

While domain knowledge may not be explicitly entered as direct input from the user, it must be an item that can be accessed, edited, and updated as a means of increasing the system's knowledge base.

2) System Perspective

A system perspective model is important because it provides a means in which the data can be processed by an inference engine. While the service provider's view is hierarchically structured and simple for a network planner to understand and specify, the system view represents the relationships between all individual entities as facts, rules, and variables. This representation is useful for applying logical processing algorithms such as unification and constraint logic programming (CLP). Additionally, design features such as navigability and composability can help optimize both the performance and scalability of the inference engine which must process this data solve for a set of device configuration settings.

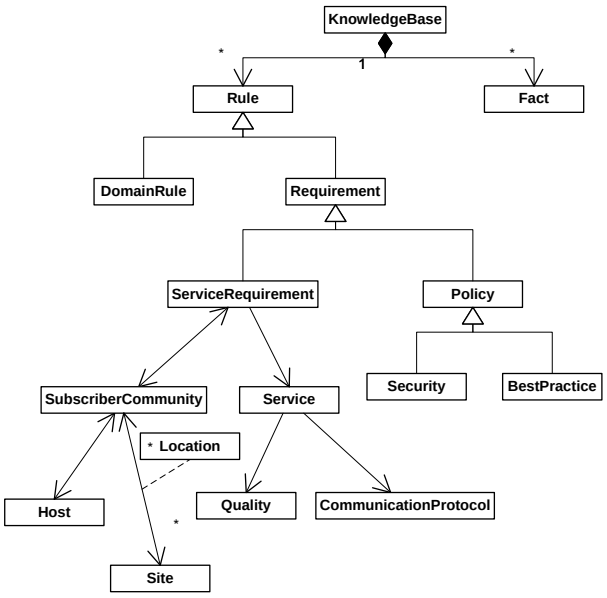

Figure 3. System Perspective Model

a) Declarative Logic

A knowledge-based (expert) system typically consists of a user interface, knowledge base, and inference engine [10]. In this context, the system perspective provides a data model for the knowledge base as a declarative set of fact, rules, and variables:

Facts are statements that are unconditionally true without dependency on other relationships. For example, network topology data is a set of facts that declare what devices are in the network and what links exist between them.

Rules declare the chain of conditions and dependencies that map goals (the network's configuration solution) to the instantiation of variables (device configuration settings). The statement that network routability between a source and destination requires a both a physical path and a static or dynamic route configured on the devices between them can be represented as a rule.

Variables in the context of this system are the configuration settings for all devices in the domain. The domain of each variable is the range of atomic values that each configuration setting may hold. The domain of a configuration variable may be a Boolean (such as whether or not a protocol should be enabled on a device), constrained enumeration (such as an Ethernet interface setting of {full-duplex, half-duplex, or auto-negotiation}), a numerical value (such as an IP address or process identification number), or a string (such as a hostname). Because an inference engine performs unification algorithms to solve for a matching instantiation of all variables,

it is important for the rules and facts to constrain the variable domains as much as possible.

b) Navigability

It is important to link data structures in a way that allows efficient lookup on both ends of an entity relationship when applicable. For example, the data structure for a subscriber (host) on the network would likely have an association with its default gateway (device) in order to facilitate routing configurations. At the same time, however, if a device were to be configured with DHCP service to provide IP addresses for its local subscribers, its data structure may also need a navigable association with its connected hosts in order to determine the configuration of its addressing scope.

c) Composability

Network design is a very dynamic field in which new capabilities are constantly developed and deployed throughout the lifecycle of a network. As a result, it is expected that network device vendors will update their operating systems to accommodate them, and their configuration models (such as those specified in a NETCONF XML schema) will likely change.

An automated network configuration system risks obsoleteness if it does not easily employ these updates in a modular fashion. The knowledge base itself should be composable by allowing the external import of new capability models, and the system's domain knowledge should map any new capabilities to its existing rules.

In order to achieve a truly composable design, the internal domain knowledge of the system should have an external interface in which a system administrator can develop and import these new capabilities without "hard-coding" them into the knowledge base.

IV. IMPLEMENTATION

In this section, we illustrate a few of the concepts put forward in the previous section with example implementations. We first explore the overall mapping of data from the user perspective model to the system perspective model. We then provide an example of the computation operations an inference engine might take to satisfy rules in the knowledge base. Finally, we present an example device configuration model that facilitates the generation of device configuration files.

A. Mapping the User Perspective to the System Perspective

Figure 4 illustrates an example in how one might describe the relationships between user-provided data (such as subscriber requirements and network topology) to the rules needed for processing device configurations. The user would provide an instantiation of a `Requirement` relationship class that maps two or more `SubscriberCommunity` objects as sources and destinations, each consisting of one or more hosts. The requirement itself then includes the set of services (such as web, e-mail applications, etc) that depend on underlying protocols. These protocols are the boundary between the user and system perspective, and we employ the OSI model to represent the dependency chain from the application layer down to the link layer. An inference engine processing these relationships would eventually determine which devices need to be configured for routing and/or switching, and certain configuration variables (such as routing parameters and access control lists) could be instantiated with literal values to satisfy the top-level subscriber requirements.

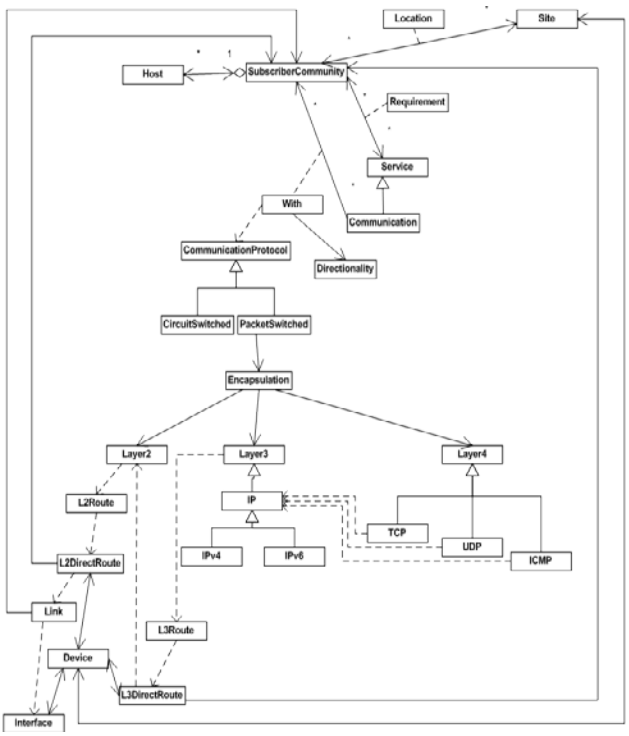

Figure 4. Relationships Between User and System Perspectives

B. Operations/Methods

While Figure 4 presents an object-oriented example of the data model used to determine the required capability of a network-layer route between subscribers, an inference engine must be able to process its objects as facts, rules, and variables. Programming languages such as Prolog or Lisp are well-suited for this task. Figure 5 provides an example (albeit incomplete) rule set in Prolog notation used to configure a static route on a router. For simplicity we use the Cisco IOS output notation "`ip route <Network/Mask> <Next Hop>`" rather than NETCONF in the example.

```
1: l3Routable(Host,Net) :- l3Connected(Host,Gateway), hasL3Route(Gateway,Net).
2: hasL3Route(Gateway,Net) :- staticRoute(Gateway,Net,NextHop), l3Connected(Gateway,NextHop).
3: staticRoute(_,Net,NextHop) :- write('ip route '), write(Net), tab(1), write(NextHop).
4: l3Connected(Host,Gateway) :- hasInterface(Gateway,X), addressAssigned(X,IP),
                                sameSubnet(IP,Host), l2Routable(IP,Host).
5: l2Routable(A,B) :- l2Connected(A,switch), l2Routable(switch,B).
6: l2Connected(X,Y) :- link(X,Y).
```

Figure 5. Rule Set for a Static Route

C. Device Configuration Model

While the user perspective and system perspective models facilitate both a top-down and middle-out design approach, respectively, we also consider the need for a bottom-up design approach in determining the ultimate output of our system: device configurations. Within the system perspective, a robust device configuration model can describe the capabilities of a network device and the means in which they are implemented.

Figure 6 depicts the high-level relationship between a device and the capabilities it supports, such as routing protocols and security features, but also illustrates the underlying mechanism to implement those features. We model a device as an object that contains both an operating system and a set of modules, which can represent any hardware or software components that enhance its capabilities (such as blades containing additional interfaces or software enhancement packages). The `ConfigModel` object depends on the operating system and modules to determine the set of configuration items available for the device. In essence, the modules tell the configuration model what can be configured, and the operating system tells it how to configure them. The configuration model then provides the `Device` object with the set of configuration capabilities which can be queried from the inference engine. When instantiated, these configuration capabilities are implemented as `ConfigItem` objects and can then be aggregated into a configuration file using the `toNetconf()` method.

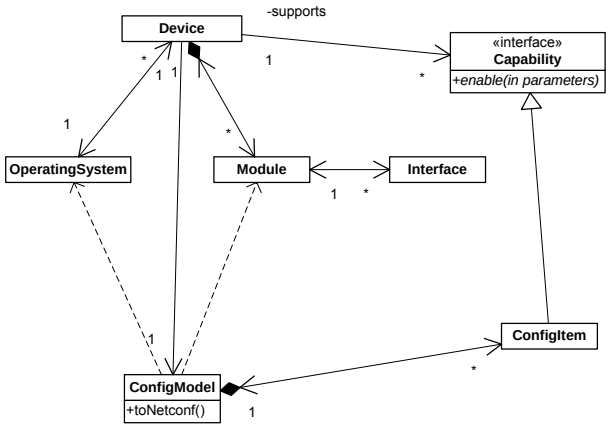

Figure 6. Device Configuration Model

V. Future Work

It is envisioned that a model employing the concepts in this paper can be used as a component in an enterprise automated network configuration system. By dynamically automating the configuration of networks, the need for detailed network configuration planning can be drastically reduced. When combined with a dynamic discovery and remote delivery component, it may be possible to realize a modular "plug-and-play" concept in which a network can be initialized and re-configured in near real-time.

References

[1] M. Lopes, A. Costa and B. Dias, "Automated Network Services Configuration Management," *IFIP/IEEE Intl. Symposium on Integrated Network Management— Workshops, 2009*.

[2] P. Szegedi, S. Figuerola, M. Campanella, V. Maglaris, and C. Cervelló-Pastor, "With Evolution for Revolution: Managing FEDERICA for Future Internet Research," *IEEE Communications Magazine, July 2009*.

[3] Telecommunicatons Information Networking Architecture Consotium (TINA-C) Specifications, www.tinac.com, 2000

[4] A. Clemm, "Network Management Fundamentals", CiscoPress, 2006

[5] Network Configuration (NETCONF) Protocol, IETF RFCs 4741-4744, 2006

[6] T. Eilam, M.H. Kalantar, A.V. Konstantinou, G. Pacifici, J. Pershing, A. Agrawal, "Managing the configuration complexity of distributed applications in Internet data centers", IEEE Communications Magazine, March 2006

[7] A. V. Konstantinou, Y. Yemini, D. Florissi, Towards Self-Configuring Networks, DARPA Active Networks Conference and Exposition, IEEE Press, San Francisco, CA, May 2002

[8] M. Harlander, "Central system administration in a heterogeneous unix environment," presented at 8th USENIX System Administration Conference (Lisa VIII), 1994.

[9] Y. Yemini, A. V. Konstantinou, and D. Florissi, "NESTOR: An Architecture for Network Self-Management and Organization", IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 18, NO. 5, MAY 2000

[10] Ivan Bratko, "PROLOG Programming for Artificial Intteligence", 3rd Ed., 2001