White Paper

Network Programmability with Cisco Application Centric Infrastructure

What You Will Learn

This document examines the programmability support on Cisco® Application Centric Infrastructure (ACI). The Cisco ACI programmability model allows complete programmatic access to the application centric infrastructure. Cisco ACI provides read and write access, through standard Representational State Transfer (REST) APIs, to the underlying object model, which is a representation of every physical and logical attribute of the entire system. With this access, customers can integrate network deployment into management and monitoring tools and deploy new workloads programmatically.

Challenges with Current Approaches to Network Programmability

Most networks in use today were built on hardware with tightly coupled software intended to be managed and administered through the command-line interface (CLI). These systems worked well in a world of static network configurations, static workloads, and predictable slower change rates for application scaling. As data center networks have been virtualized and begun moving to cloud and agile IT models, this model no longer works.

Therefore, vendors are working to layer programmability onto existing offerings and device operating systems. Although this approach increases capabilities, it is not an ideal method for incorporating programmability. This model creates management complexity by introducing an entirely new point of management, typically identified as a Network Controller, which tries to artificially map application and user policies to inflexible network constructs. Further these Network Controllers and the models they expose are limited to network functions and cannot extend to supporting rest of the infrastructure. True programmability needs to be incorporated at the foundation, not applied as an afterthought. The infrastructure components and the constructs they expose needs to be designed with programmability at its foundation using a model that developers can understand and use quickly.

Cisco ACI Programmability with Object-Oriented Data Model and REST APIs

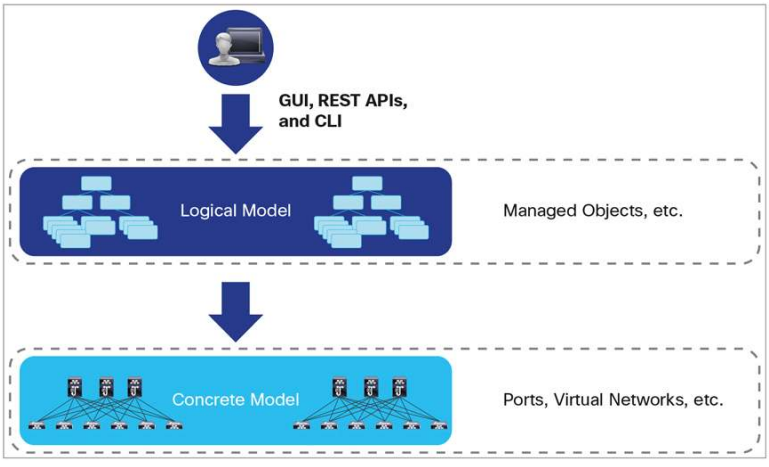
Cisco has taken a foundational approach to building a programmable network infrastructure with the Cisco ACI solution. This infrastructure operates as a single system at the fabric level, controlled by the centralized Cisco Application Policy Infrastructure Controller (APIC). With this approach, the data center network as a whole is tied together cohesively and treated as an intelligent transport system for the applications that support business. On the network devices that are part of this fabric, the core of operating system has been written to support this system view and provide an architecture for programmability at the foundation.

Instead of opening up a subset of the network functionality through programmatic interfaces, like previous generation Software Defined Networking (SDN) solutions, the entire infrastructure is opened up for programmatic access. This is achieved by providing access to Cisco ACI object model, the model that represents the complete configuration and runtime state of every single software and hardware component in entire infrastructure. Further this object model is made available through standard REST interfaces, making it easier to access and manipulate the object model, and hence, the configuration and runtime state of the system.

At the top level, the Cisco ACI object model is based on promise theory, which provides a scalable control architecture, with autonomous objects responsible for implementing the desired state changes provided by the controller cluster. This approach is more scalable than traditional top-down management systems, which require detailed knowledge of low-level configurations and the current state. With promise theory, desired state changes are pushed down, and objects implement the changes, returning faults when required.

Beneath this high-level concept is the core of Cisco ACI programmability: the object model. The model can be divided into two major parts: logical and physical. Model-based frameworks provide an elegant way to represent data. The Cisco ACI model provides comprehensive access to the underlying information model, providing policy abstraction, physical models, and debugging and implementation data. Figure 1 depicts the Cisco ACI model framework. The model can be accessed over REST APIs, thus opening up the system for programmability.

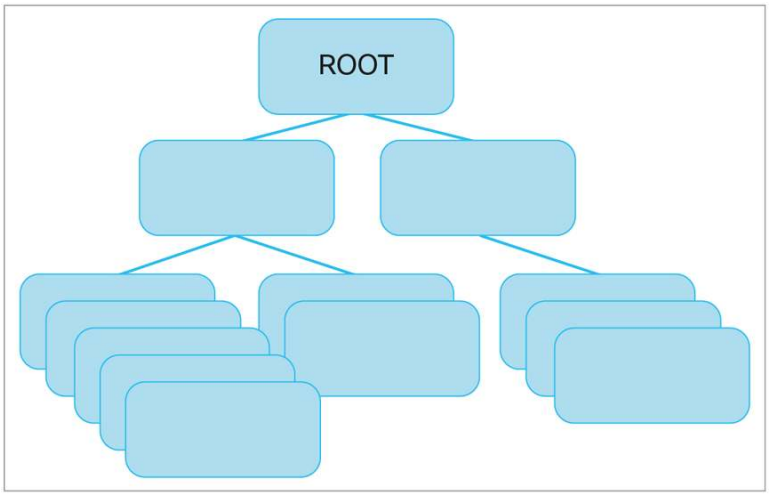
Figure 1. Cisco ACI Object-Oriented Data Model and REST APIs



As shown in Figure 1, the logical model is the interface with the system. Administrators or upper-level cloud management systems interact with the logical model through the API, CLI, or GUI. Changes to the logical model are then pushed down to the physical model, which typically becomes the hardware configuration.

The logical model itself consists of the objects - configuration, policies and runtime state - that can be manipulated and the attributes of those objects. In the Cisco ACI framework, this model is known as the management information tree (MIT). Each node in the MIT represents a managed object or group of objects. These objects are organized in a hierarchical way, creating logical object containers. Figure 2 depicts the logical hierarchy of the MIT object model.

Figure 2. Management Information Tree (MIT)



Objects in the MIT

The Cisco ACI uses an information-model-based architecture in which the model describes all the information that can be controlled by a management process. Object instances are referred to as managed objects (MOs). Every managed object in the system can be identified by a unique distinguished name (DN). This approach allows the object to be referred to globally.

In addition to its distinguished name, each object can be referred to by its relative name (RN). The relative name identifies an object relative to its parent object. Any given object's distinguished name is derived from its own relative name appended to its parent object's distinguished name. Distinguished names are directly mapped to URLs. Either the relative name or the distinguished name can be used to access an object, dependent on the current location in the MIT. The relationship among managed objects, relative names, and distinguished names is shown in Figure 3.

Figure 3. Managed Objects, Relative Names, and Distinguished Names

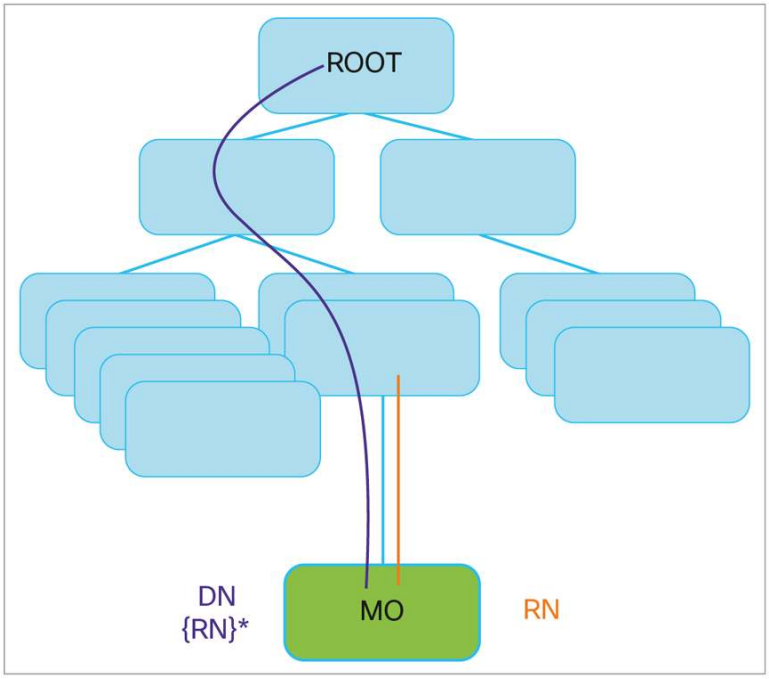


Figure 3 depicts the distinguished name, which uniquely represents any given managed object instance, and the relative name, which represents it locally underneath its parent managed object. All objects in the tree exist under the root object.

Because of the hierarchical nature of the tree and the attribute system used to identify object classes, the tree can be queried in several ways for managed object information. Queries can be performed on an object itself through its distinguished name, on a class of objects such as Switch Chassis, or on a tree-level, discovering all members of an object. Figure 4 shows two tree-level queries.

Figure 4. Tree-Level Queries

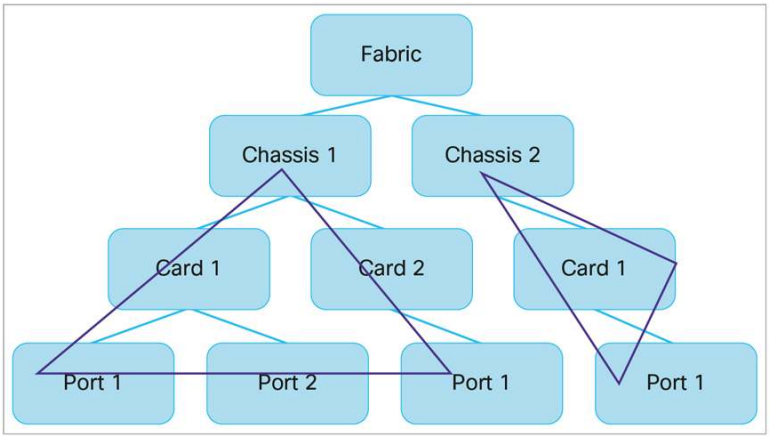
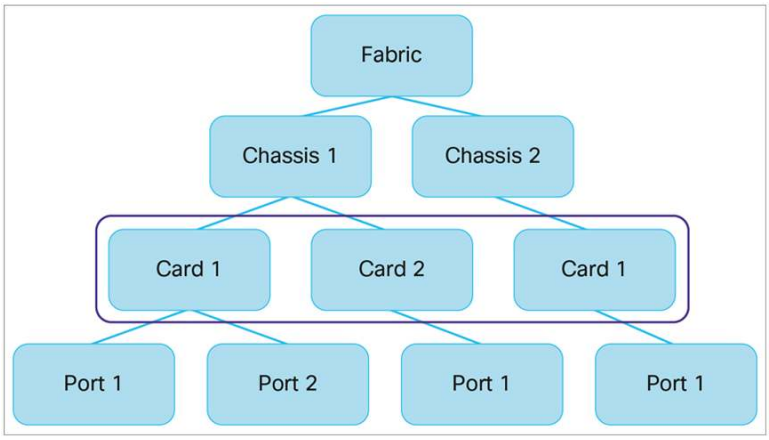


Figure 4 shows two chassis being queried at the tree level. Both queries return the referenced object and its child objects. This approach is a useful tool for discovering the components of a larger system.

The example in Figure 4 discovers the cards and ports of a given switch chassis. Figure 5 shows another type of query: the class-level query.

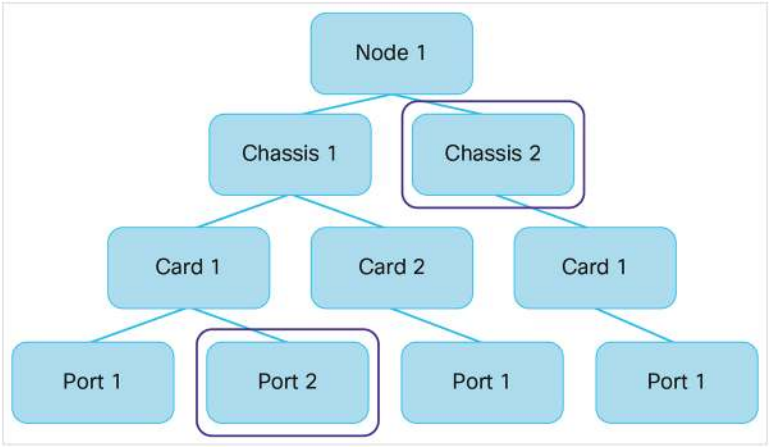
Figure 5. Class-Level Queries



As shown in Figure 5, class-level queries return all the objects of a given class. This approach is useful for discovering all the objects of a certain type available in the MIT. In this example, the class used is Cards, which returns all the objects of type Cards.

The third query type is an object-level query. In an object-level query a distinguished name is used to return a specific object. Figure 6 depicts two object-level queries: one for Node 1 in Chassis 2, and one for Node 1 in Chassis 1 in Card 1 in Port 2.

Figure 6. Class-Level Queries

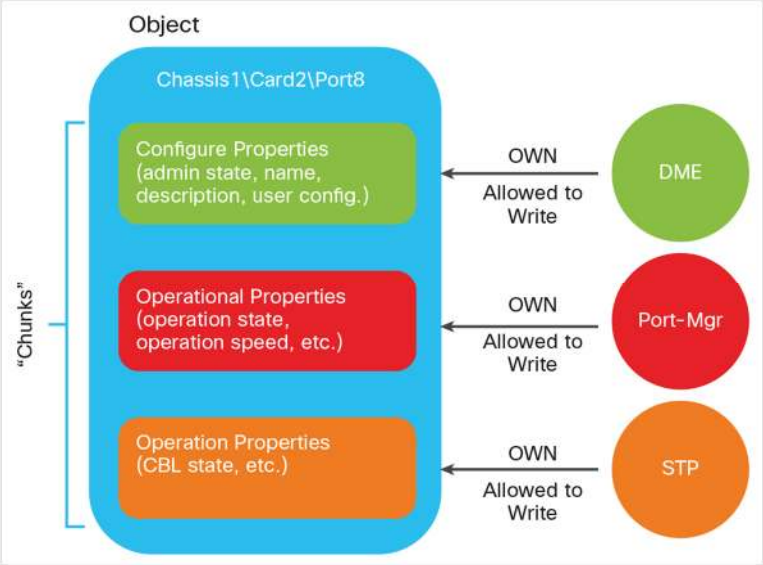


For all MIT queries, you can optionally return the entire subtree or a partial subtree. Additionally, the role-based access control (RBAC) mechanism in the system dictates which objects are returned; only the objects that the user has rights to view will ever be returned.

Managed-Object Properties

Managed objects in Cisco ACI contain properties that define the managed object. Properties in a managed object are divided into chunks managed by given processes within the operating system. Any given object may have several processes that access it. All these properties together are compiled at runtime and presented to the user as a single object. Figure 7 shows an example of this relationship.

Figure 7. Managed-Object Properties



In Figure 7, the example object has three processes that write to property chunks in the object. The Data Management Engine (DME), which is the interface between the Cisco APIC (thus the user) and the object, the Port Manager, which handles port configuration, and the Spanning Tree Protocol (STP) all interact with chunks of this object. The object itself is presented to the user through the API as a single entity compiled at runtime.

Accessing the Object Data through REST Interfaces

REST is a software architecture style for distributed systems such as the World Wide Web. REST has emerged over the past few years as a predominant web service design model. REST has increasingly displaced other design models such as Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL) due to its simpler style. The Cisco APIC supports REST interfaces for programmatic access to the entire Cisco ACI solution.

The object-based information model of Cisco ACI makes it a very good fit for REST interfaces: URLs and URIs map directly to distinguished names identifying objects on the tree, and any data on the MIT can be described as a self-contained structured text tree document encoded in XML or JavaScript Object Notation (JSON). The objects have parent-child relationships that are identified using distinguished names and properties, which are read and modified by a set of create, read, update, and delete (CRUD) operations.

Objects can be accessed at their well-defined address, their REST URLs, using standard HTTP commands for retrieval and manipulation of Cisco APIC object data. The URL format used can be represented as follows:

```
<system>/api/[mo|class]/[dn|class][:method].[xml|json]?{options}
```

The various building blocks of the preceding URL are as follows:

- **System:** System identifier; an IP address or DNS-resolvable host name
- **mo | class:** Indication of whether this is a managed object or tree (MIT) or class-level query
- **class:** Managed-object class (as specified in the information model) of the objects queried; the class name is represented as <pkgName><ManagedObjectClassName>
- **dn:** Distinguished name (unique hierarchical name of the object in the MIT tree) of the object queried
- **method:** Optional indication of the method being invoked on the object; applies only to HTTP POST requests
- **xml | json:** Encoding format
- **options:** Query options, filters, and arguments

With the capability to address and access an individual object or a class of objects with the REST URL, you can achieve complete programmatic access to the entire object tree and, thereby, to the entire system.

Software Development Kits for Programming Environments

The REST APIs for Cisco ACI allow easy integration into any programmatic environment, regardless of the language and development methodology used. To further accelerate development in commonly used programming environments, software development kits (SDKs) for Cisco ACI will be made available. The Cisco ACI-pysdk, a Python-based SDK is one such SDK for a Python programming environment. The Python libraries and APIs that are part of the SDK abstract underlying REST API calls and provide easy and rapid integration into software suites that are Python based.

Conclusion

The Cisco ACI object-oriented data model is designed from the foundation for network programmability. At the device level, the operating system has been rewritten as a fully object-based switch operating system for Cisco ACI. The components of Cisco ACI are managed by the Cisco APIC, which provides full-function REST APIs. On top of this API are both a CLI and a GUI for day-to-day administration.

The object model enables fluid programmability and full access to the underlying components of the infrastructure using REST APIs. Objects are organized logically into a hierarchical model and stored in the MIT. This approach provides a framework for network control and programmability with a degree of openness that is not found in other systems.

For More Information

http://www.cisco.com/go/aci.

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at www.cisco.com/go/offices.

Printed in USA C11-729999-00 11/13