White Paper

Application Centric Infrastructure Object-Oriented Data Model: Gain Advanced Network Control and Programmability

What You Will Learn

This document discusses application centric infrastructure (ACI), a model that provides an object-oriented set of network constructs that are fully programmable through an open representational state transfer (REST) API. Using the ACI object model in conjunction with the API, customers can integrate a network deployment with management and monitoring tools and deploy new workloads programmatically.

Challenges of Current 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 an environment with static network configurations; static workloads; and predictable, slower rates of change in application scale. With data center networks virtualized and moving toward cloud and agile IT models, this model is no longer applicable.

Therefore, vendors are working to layer programmability onto existing offerings and device operating systems. Although this approach increases capabilities, however, it is not an ideal means of adding programmability. True programmability needs to be built in from the foundation, not applied as an afterthought. The hardware that powers the network needs to be built with programmability at the forefront using a model that developers understand and can quickly put into operation.

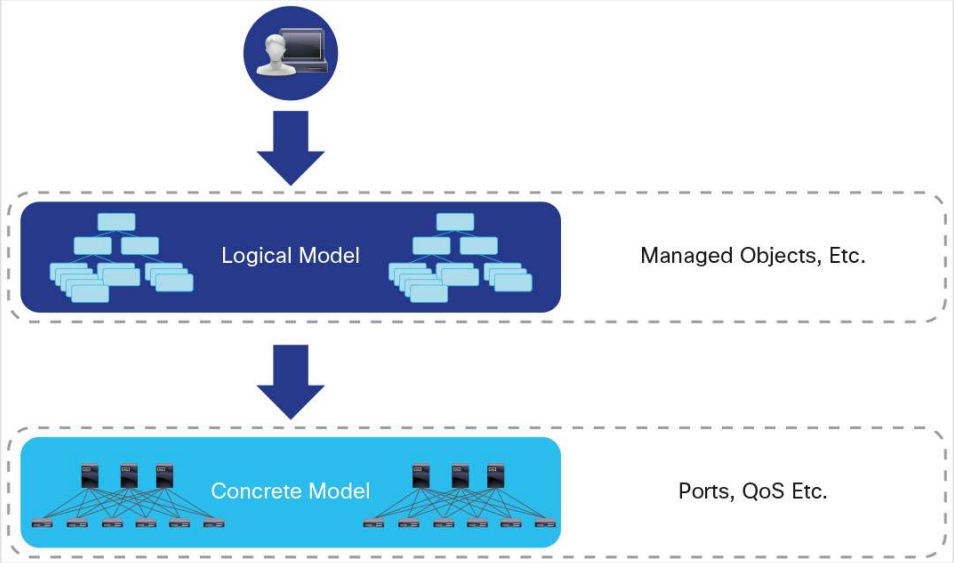
Application Centric Infrastructure Object-Oriented Data Model

With ACI, programmable network infrastructure is built from the foundation. This infrastructure operates as a single system at the solution level. With this approach, the data center network as a whole is cohesively connected and is treated as an intelligent transport system for the applications on which business today depends. To support this system view, the device operating system has been rewritten to expose functions as a series of objects and their attributes, providing an architecture for programmability at the foundation. This approach also maintains the stable code base of network services, such as routing protocols for Cisco® NX-OS Software, which are data center tested and trusted on Cisco Nexus® data center products.

At the top level, the ACI model is based on promise theory, which provides scalable control architecture. In this model, autonomous objects are 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 configuration details and the current state. With promise theory, the desired state changes are pushed down, and objects implement the changes, returning faults when required.

Underlying this high-level concept is the core of ACI programmability: the object model. The model is divided into two parts: logical and concrete (which represents the physical. A model-based framework provides an elegant way to formalize data representation. The ACI model provides comprehensive access to the underlying information, providing policy abstraction, physical models, and debugging and implementation data. Figure 1 shows the ACI model framework.

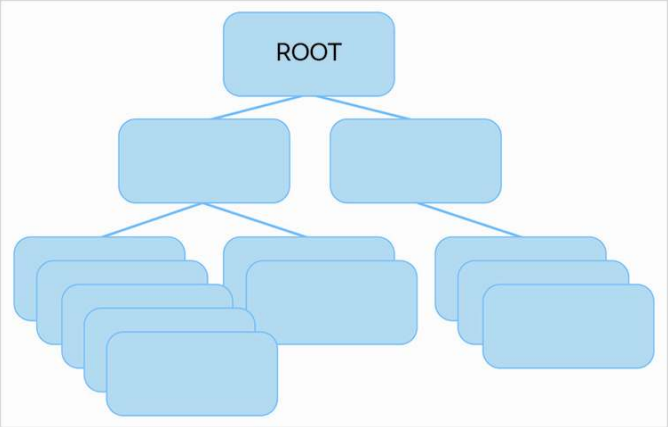
Figure 1. ACI Object-Oriented Data Model



As shown in Figure 1, the logical model is the interface for the system. Administrators and 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 concrete model, which becomes the hardware and software configuration.

The logical model consists of the objects that can be manipulated and the attributes of those objects. Within the ACI framework, this framework is known as the management information tree (MIT.) Each node in the MIT represents a managed object or group of objects, each with its own unique attributes. These objects are organized in a hierarchy, creating logical object containers. Figure 2 shows the logical hierarchy of the MIT object model.

Figure 2. Management Information Tree



Objects in the MIT

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 depending on its current location in the MIT. The relationship between a managed object's relative name and its distinguished name is shown in Figure 3.

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

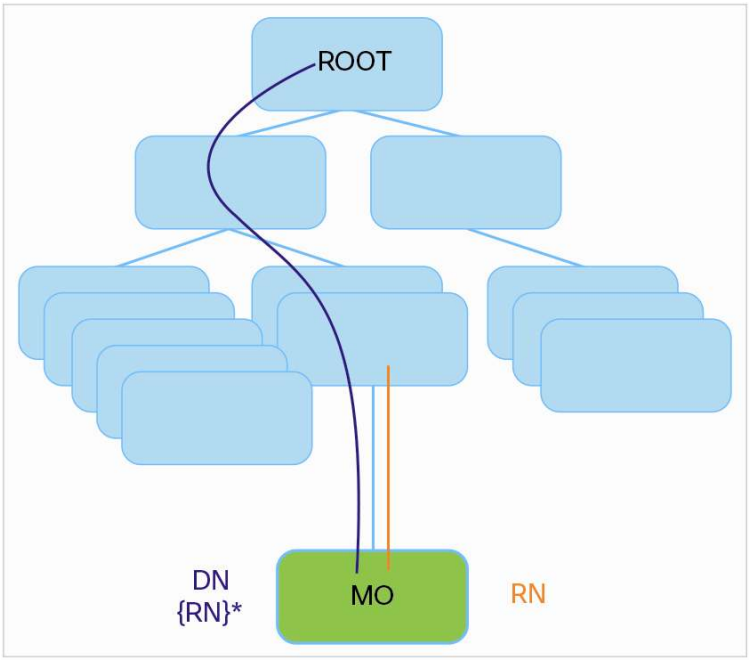


Figure 3 depicts the distinguished name, which represents any given managed object universally, and the relative name, which represents the object locally under its parent managed object. All objects in the tree exist under the root object.

Because of the hierarchical nature of the tree and the attribute system used to identify object classes, the tree can be queried for managed object information in several ways. Queries can be run:

- On an object itself using the distinguished name
- On a class of objects such as switch chassis
- On a tree level, to discover all members of an object

Figure 4 shows two tree-level queries.

Figure 4. Tree-Level Queries

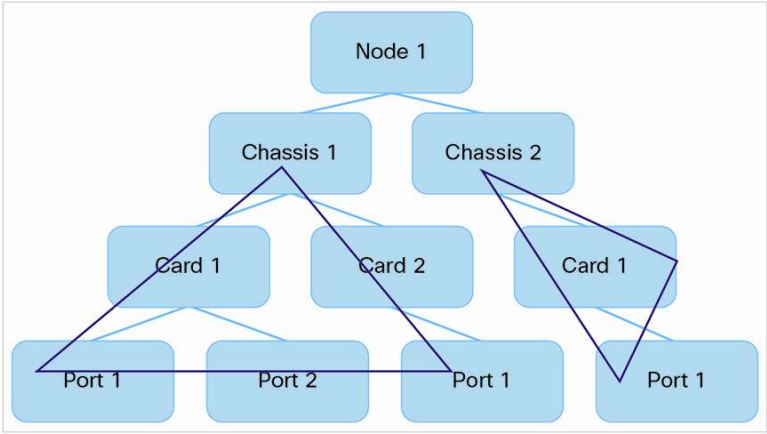
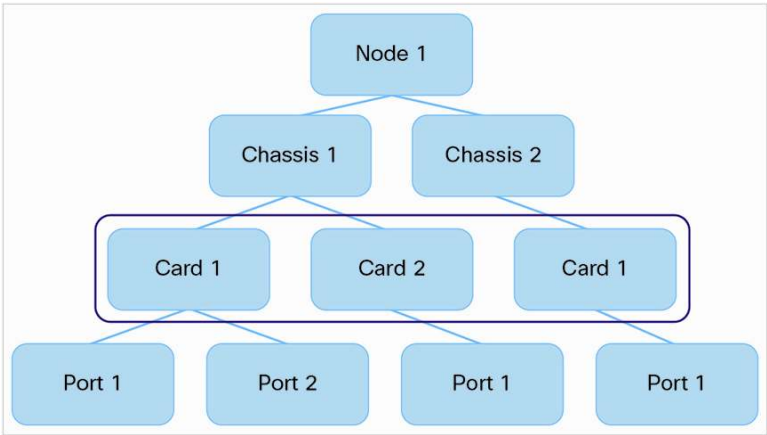


Figure 4 shows two chassis being queried at the tree level. Both queries return the referenced object and its child objects. This type of query is useful for discovering the components of a larger system. In this example, the query discovers the cards and ports of a given switch chassis.

Figure 5 shows another type of query: the class-level query.

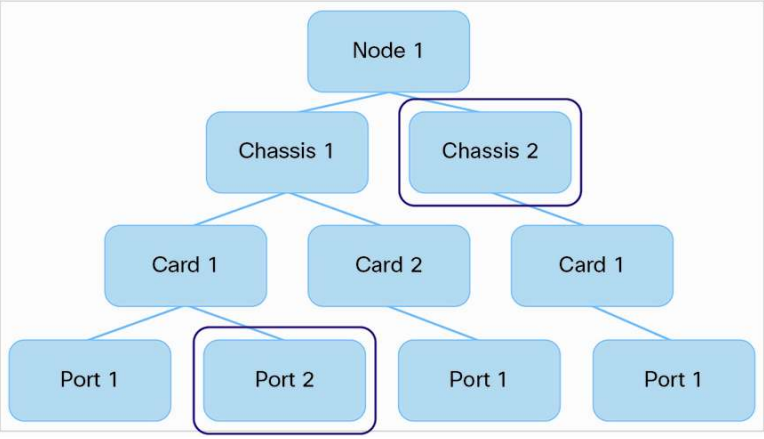
Figure 5. Class-Level Queries



As shown in Figure 5, class-level queries return all the objects of a given class. This type of query is useful for discovering all the objects of a certain type available in the MIT. In this example, the query searches for the card class, which returns all objects of type “cards.”

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

Figure 6. Object-Level Queries

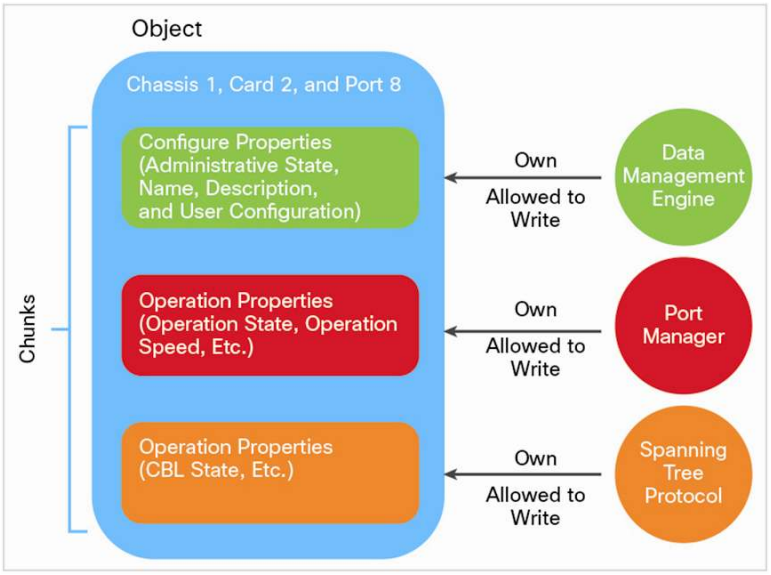


All MIT queries have options to return the entire subtree or a partial subtree. Additionally, the role-based access control (RBAC) in the system dictates which objects are returned: for example, only objects for which the user has rights to view are ever returned.

Managed-Object Properties

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

Figure 7. Managed-Object Properties



In Figure 7, the sample object has three processes, which write to property chunks within the object. The DME, which is the interface between the controller (and thus the user) and the object; the port manager, which handles port configuration; and the Spanning Tree Protocol all interact with chunks within this object. The object itself is presented to the user through the API as a single entity compiled at runtime.

Conclusion

The ACI object-oriented data model is designed from the foundation for network programmability based on application connectivity and policy. At the switch level, the operating system has been rewritten into a fully object-based model. The solution as a whole is managed by the policy controller, which provides a REST API that is programmable using XML and JavaScript Object Notation (JSON). 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 network solution. Objects are organized logically into a hierarchical model and stored in the MIT. This approach provides an advanced framework for network control and programmability 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 C11-729586-00 11/13