Automation Composition Management: Architecture and Design

- Introduction
- Terminology
  - Automation Composition Management Terminology
  - Participant Terminology
  - Terminology for Properties
  - Concepts and their relationships
- Capabilities
  - Automation Composition Instance States
- Architecture Details
- Using Automation Composition Management for Managing Control Loops
- Other Considerations
  - Management of Automation Composition Instance Configurations
  - Scalability
  - Sandboxing and API Gateway Support
- 5 APIs and Protocols
- 6 Design and Implementation

To get a feature, service, or capability working in modern networks is not straightforward. It is not as simple as deploying a microservice or running a workflow. Our features, services, and capabilities are now typically delivered using loose compositions of microservices, rules, algorithms, configurations, and workflows. Of course, we use workflows and deploy microservices, but how do we keep track of what workflow activated which service or what microservice instance enables a given capability. We must be able to deploy, keep track of, amend, and remove the compositions that combine to give us our features, services and capabilities, that is we must manage those compositions.

Consider Features A, B and C in the diagram above.

Feature A is realised as an Analytic Microservice, but it also requires counters to be configured in a collection service to enable its input stream of data. It also requires two policies to be present, and its result requires an Ansible playbook to be present.

Feature B is realised as an AI microservice, which is triggered by a set of triggers that are configured in the persistence service. The AI algorithm in Feature B triggers a workflow in the workflow service.

Feature C is realised as two microservices, an analytic microservice and a Machine Learning microservice. The feature also requires that certain counters are collected and certain Netconf configurations are enabled.

All three features are realised as Automation Compositions, as shown in the diagram below.
The ability to deploy features in a scalable, flexible and loosely coupled microservice architecture is of course a major step forward from layered architectures of the past. However, managing at "Feature" level in such architectures does present challenges. For example, to manage the three running instances of Features A to C above, 9 separate elements must be kept track of. There is nothing in the deployed system to say what element is related to what other element, and what element are working together to realise a feature.

**Automation Composition Management (ACM)** is a framework that supports Life Cycle Management of Automation Compositions. It supports deployment, monitoring, update and removal of Automation Compositions en-bloc, allowing users to manage their features, services, and capabilities as single logical units.

**Introduction**

The idea of using compositions to automate network management has been the subject of much research in the Network Management research community, see this paper for some background. However, it is only with the advent of ONAP that we have a platform that supports management of those compositions. Before ONAP, Automation Compositions have been implemented by hard-coding elements together and hard coding logic into elements.

Automation Composition Management in ONAP provides a complete open-source framework for Automation Composition Lifecycle Management. It provides TOSCA based Automation Composition definition and development, commissioning and run-time management. The elements that comprise an Automation Composition and the metadata needed to collect the elements together to create a composition are specified in a standardized way using the OASIS TOSCA modelling language. The TOSCA description is then used to commission, instantiate, and manage Automation Compositions in the run-time system.

The diagram above shows the architecture of ACM in ONAP.

The four components in ACM are the ACM Runtime (ACM-R) component, the ACM Design (ACM-D) component, the AC Participants, and the ACM Client (ACM-C).
ACM-R is a server that manages the life cycle of Automatic Compositions. It uses commissioned TOSCA Automation Composition definitions from ACM-D to manage the life cycle of Automation Compositions. It works with Automation Composition Participants to manage the life cycle of Automation Composition elements.

ACM-D is a design environment that allows users to define metadata for elements such as microservices, policies, workflows and to onboard that metadata into the Design Time Catalogue. It also allows users to assemble Automation Compositions by defining compositions and selecting onboarded metadata for Automation Composition Elements to be included in compositions. ACM-D can send an AC definition to the ACM-R or it can save the AC definition to a file for separate commissioning using the ACM client.

AC Participants are components that work with the ACM-R to manage the lifecycle of particular types of Automation Composition Elements.

ACM-C is a client that allows users to commission, set common and instance-specific parameters for Automation Configurations as well as changing the state and monitoring the life cycle of Autonomic Compositions and their elements.

Terminology

This section describes the terminology used in the system.

Automation Composition Management Terminology

Automation Composition Type: A definition of an Automation Composition in the TOSCA language. This definition describes a certain type of an Automation Composition. The life cycle of instances of an Automation Composition Type are managed by ACM.

Automation Composition Instance: An instance of an Automation Composition Type. The life cycle of an Automation Composition Instance is managed by ACM. An Automation Composition Instance is a set of executing Automation Composition Elements on which Life Cycle Management (LCM) is executed collectively. For example, a set of microservices may be spawned and executed together to deliver a service.

Automation Composition Element Type: A definition of an Automation Composition Element in the TOSCA language. This definition describes a certain type of Automation Composition Element for an Automation Composition in an Automation Composition Type.

Automation Composition Element Instance: A single entity executing on a participant, with its Life Cycle being managed as part of the overall Automation Composition. For example, a single microservice that is executing as one microservice in a service.

Automation Composition Runtime: The ACM-R server that holds Automation Composition Type definitions and manages the life cycle of Automation Composition Instances and their Automation Composition Elements in cooperation with participants.

Participant Terminology

Participant Type: Definition of a type of system or framework that can take part in Automation Compositions and a definition of the capabilities of that participant type. A participant advertises its type to the Automation Composition Runtime.

Participant: A system or framework that takes part in Automation Compositions by executing Automation Composition Elements in cooperation with the Automation Composition Runtime. A participant chooses to partake in Automation Compositions, to manage Automation Composition Elements for ACM-R, and to receive, send and act on LCM messages for ACM-R.

Terminology for Properties

Common Properties: Properties that apply to all Automation Composition Instances of a certain Automation Composition Type and are specified when an Automation Composition Type is commissioned.

Instance Specific Properties: Properties that must be specified for each Automation Composition Instance and are specified when an Automation Composition Instance is initialized.

Concepts and their relationships

The UML diagram below shows the concepts described in the terminology sections above and how they are interrelated.
The Automation Composition Definition concepts describe the types of things that are in the system. These concepts are defined at design time and are passed to the runtime in a TOSCA document. The concepts in the Automation Composition Runtime are created by the runtime part of the system using the definitions created at design time.

Capabilities

We consider the capabilities of Automation Compositions at Design Time and Run Time.

At Design Time, three capabilities are supported:

1. **Automation Composition Element Definition Specification.** This capability allows users to define Automation Composition Element Types and the metadata that can be used on and configured on an Automation Composition Element Type. Users also define the Participant Type that will run the Automation Composition Element when it is taking part in an Automation Composition. The post condition of an execution of this capability is that metadata for an Automation Composition Element Type is defined in the Automation Composition Design Time Catalogue.

2. **Automation Composition Element Definition Onboarding.** This capability allows external users and systems (such as SDC or DCAE-MOD) to define the metadata that can be used on and configured on an Automation Composition Element Type and to define the Participant Type that will run the Automation Composition Element when it is taking part in an Automation Composition. The post condition of an execution of this capability is that metadata for an Automation Composition Element Type is defined in the Automation Composition Design Time Catalogue.

3. **Automation Composition Type Definition.** This capability allows users and other systems to create Automation Composition Type definitions by specifying a set of Automation Composition Element Definitions from those that are available in the Automation Composition Design Time Catalogue. These Automation Composition Elements will work together to form Automation Compositions. In an execution of this capability, a user specifies the metadata for the Automation Composition and specifies the set of Automation Composition Elements and their Participant Types. The user also selects the correct metadata sets for each participant in the Automation Composition Type and defines the overall Automation Composition Type metadata. The user also specifies the Common Property Types that apply to all instances of an Automation Composition Type and the Instance Specific Property Types that apply to individual instances of an Automation Composition Type. The post condition for an execution of this capability is an Automation Composition definition in TOSCA stored in the Automation Composition Design Time Catalogue.

Note that once an Automation Composition Definition is commissioned to the Automation Composition Runtime and has been stored in the Run Time Inventory, it cannot be further edited unless it is decommissioned.

At Run Time, the following participant related capabilities are supported:

1. **System Pre-Configuration.** This capability allows participants to register and deregister with ACM-R. Participants explicitly register with ACM-R when they start. Automation Composition Priming is performed on each participant once it registers. The post condition for an execution of this capability is that a participant becomes available (registration) or is no longer available (deregistration) for participation in an Automation Composition.

2. **Automation Composition Priming on Participants.** A participant is primed to support a Automation Composition Type. Priming a participant means that the definition of an Automation Composition and the values of Common Property Types that apply to all instances of an Automation Composition type on a participant are sent to a participant. The participant can then take whatever actions it need to do to support the Automation Composition type in question. Automation Composition Priming takes place at participant registration and at Automation Composition Commissioning. The post condition for an execution of this capability is that all participants in this Automation Composition type are commissioned, that is they are prepared to run instances of their Automation Composition Element types.
At Run Time, the following Automation Composition Life Cycle management capabilities are supported:

1. **Automation Composition Commissioning.** This capability allows version controlled Automation Composition Type definitions to be taken from the Automation Composition Design Time Catalogue and be placed in the Commissioned Automation Composition Inventory. It also allows the values of Common Property Types that apply to all instances of an Automation Composition Type to be set either by taking default values assigned in the AC definition or using the ACM client. Further, the Automation Composition Type is primed on all concerned participants. The post condition for an execution of this capability is that the Automation Composition Type definition is in the Commissioned Automation Composition Inventory and the Automation Composition Type is primed on concerned participants.

2. **Automation Composition Instance Life Cycle Management.** This capability allows an Automation Composition Instance to have its life cycle managed.
   a. **Automation Composition Instance Creation:** This capability allows a Automation Composition Instance to be created. The Automation Composition Type definition is read from the Commissioned Automation Composition Inventory and values are assigned to the Instance Specific Property Types defined for instances of the Automation Composition Type by the ACM client. an Automation Composition Instance that has been created but has not yet been instantiated on participants is in state UNINITIALIZED. In this state, the Instance Specific Property Type values can be revised and updated as often as the user requires. The post condition for an execution of this capability is that the Automation Composition instance is created in the Instantiated Automation Composition Inventory but has not been instantiated on Participants.
   b. **Automation Composition Instance Update on Participants:** Once the user is happy with the property values, the Automation Composition Instance is updated on participants and the Automation Composition Elements for this Automation Composition Instance are initialized or updated by participants using the Automation Composition metadata. The post condition for an execution of this capability is that the Automation Composition instance is updated on Participants.
   c. **Automation Composition State Change:** The user can now order the participants to change the state of the Automation Composition Instance. If the Automation Composition is set to state RUNNING, each participant begins accepting and processing Automation Composition events and the Automation Composition Instance is set to state RUNNING in the Instantiated Automation Composition inventory. The post condition for an execution of this capability is that the Automation Composition instance state is changed on participants.
   d. **Automation Composition Instance Monitoring.** This capability allows Automation Composition Instances to be monitored. Users can check the status of Participants, Automation Composition Instances, and Automation Composition Elements. Participants report their overall status and the status of Automation Composition Elements they are running periodically to ACM-R. ACM-R aggregates these status reports into an aggregated Automation Composition Instance status record, which is available for monitoring. The post condition for an execution of this capability is that Automation Composition Instances are being monitored.
   e. **Automation Composition Instance Supervision.** This capability allows Automation Composition Instances to be supervised. ACM-R expects participants to report on Automation Composition Elements periodically. The ACM-R checks that periodic reports are received and that each Automation Composition Element is in the state it should be in. If reports are missed or if an Automation Composition Element is in an incorrect state, remedial action is taken and notifications are issued. The post condition for an execution of this capability is that Automation Composition Instances are being supervised by ACM-R.
   f. **Automation Composition Instance Removal from Participants:** A user can order the removal of an Automation Composition Instance from participants. The post condition for an execution of this capability is that the Automation Composition instance is removed from Participants.
   g. **Automation Composition Instance Deletion:** A user can order the removal of an Automation Composition Instance from ACM-R. Automation Composition Instances that are instantiated on participants cannot be removed from ACM-R. The post condition for an execution of this capability is that the Automation Composition instance is removed from Instantiated Automation Composition Inventory.

3. **Automation Composition Decommissioning.** This capability allows version controlled Automation Composition Type definitions to be removed from the Commissioned Automation Composition Inventory. an Automation Composition Definition that has instances in the Instantiated Automation Composition Inventory cannot be removed. The post condition for an execution of this capability is that the Automation Composition Type definition removed from the Commissioned Automation Composition Inventory.

Note that the system dialogues for run time capabilities are described in detail on the System Level Dialogues page.

**Automation Composition Instance States**

When an Automation Composition definition has been commissioned, instances of the Automation Composition can be created, updated, and deleted. The system manages the lifecycle of Automation Composition instances and Automation Composition element instances following the state transition diagram below.
Architecture Details

The diagram below shows the architecture of TOSCA based Automation Composition Management in ONAP.
Following the ONAP Reference Architecture, the architecture has a Design Time part and a Runtime part.

The Design Time part of the architecture allows a user to specify and/or onboard metadata for Automation Composition Elements. It also allows users to assemble Automation Compositions. The Design Time Catalogue contains the metadata primitives and Automation Composition definition primitives for composition of Automation Compositions. As shown in the figure above, the Design Time component provides a system where Automation Compositions can be designed and defined in metadata. This means that an Automation Composition can have any arbitrary structure and the Automation Composition developers can use whatever analytic, policy, or control participants they like to implement their Automation Composition. At assembly time, the user parameterises the Automation Composition and stores it in the design time catalogue. This catalogue contains the primitive metadata for any Automation Composition Elements that can be used to assemble an Automation Composition. An Automation Composition SDK is used to assemble an Automation Composition by aggregating the metadata for the Automation Composition Elements chosen to be used in an Automation Composition and by allowing the AC designer to define parameters for and references between these elements.

Assembled Automation Compositions are commissioned on the run time part of the system, where they are stored in the Commissioned Automation Composition inventory and are available for instantiation. The Commissioning component provides a CRUD REST interface for Automation Composition Types, and implements CRUD of Automation Composition Types. Commissioning also implements validation and persistence of incoming Automation Composition Types. It also guarantees the integrity of updates and deletions of Automation Composition Types, such as performing updates accordance with semantic versioning rules and ensuring that deletions are not allowed on Automation Composition Types that have instances defined.

The Instantiation component manages the Life Cycle Management of Automation Composition Instances and their Automation Composition Elements. It publishes a REST interface that is used to create Automation Composition Instances and set values for Common and Instance Specific properties. This REST interface is public and is used by the ACM Client. It may also be used by any other client via the public REST interface. The REST interface also allows the state of Automation Composition Instances to be changed. A user can change the state of Automation Composition Instances as described in the state transition diagram shown above. The Instantiation component issues update and state change messages via DMaaP to participants so that they can update and manage the state of the Automation Composition Elements they are responsible for. The Instantiation component also implements persistence of Automation Composition Instances, Automation Composition elements, and their state changes.

The Monitoring component reads updates sent by participants. Participants report on the state of their Automation Composition Elements periodically and in response to a message they have received from the Instantiation component. The Monitoring component reads the contents of the participant messages and persists their state updates and statistics records. It also publishes a REST interface that publishes the current state of all Participants, Automation Composition Instances and their Automation Composition Elements, as well as publishing Participant and Automation Composition statistics.

The Supervision component is responsible for checking that Automation Composition Instances are correctly instantiated and are in the correct state (UNINITIALIZED/READY/RUNNING). It also handles timeouts and on state changes to Automation Composition Instances, and retries and rolls back state changes where state changes failed.

A Participant is an executing component that partakes in Automation Compositions. More explicitly, a Participant is something that implements the Participant Monitoring and Participant Instantiation messaging protocol over DMaaP for Life Cycle management of Automation Composition Elements. A Participant runs Automation Composition Elements and manages and reports on their life cycle following the instructions it gets from ACM-R in messages delivered over DMaaP.

In the figure above, five participants are shown. A Configuration Persistence Participant manages Automation Composition Elements that interact with the ONAP Configuration Persistence Service to store common data. The DCAE Participant runs Automation Composition Elements that manage DCAE microservices. The Kubernetes Participant hosts the Automation Composition Elements that are managing the life cycle of microservices in Automation Compositions that are in a Kubernetes ecosystem. The Policy Participant handles the Automation Composition Elements that interact with the Policy Framework to manage policies for Automation Compositions. A Controller Participant such as the CDS Participant runs Automation Composition Elements that load metadata and configure controllers so that they can partake in Automation Compositions. Any third party Existing System Participant can be developed to run Automation Composition Elements that interact with any existing system (such as an operator’s analytic, machine learning, or artificial intelligence system) so that those systems can partake in Automation Compositions.

Using Automation Composition Management for Managing Control Loops

A Control Loop is a special case of an Automation Composition, it is an Automation Composition in which specific Automation Composition elements must be present. Typically, there is a Monitoring (or Collection), an Analysis, a Plan, and an Execution (Controller) element. The flow of the Control Loop moves from one element to the next, iterating over time. The MAPE reference pattern for Control Loops defined by IBM in the 1990s is shown on the left hand side of the diagram below. The architectural pattern for control loops in ONAP is shown on the right hand side of the diagram.
ONAP control loops are supported as Automation Compositions by ACM. When a user assembles a Automation Composition using the Automation Composition Elements from the ONAP Control Loop architectural pattern, ACM-R can commission, instantiate and manage the life cycle of an ONAP Control Loop just like any other composition.

Other Considerations

Management of Automation Composition Instance Configurations

In order to keep management of versions of the configuration of Automation Composition instances straightforward and easy to implement, the following version management scheme using semantic versioning is implemented. Each configuration of an Automation Composition Instance and configuration of an Automation Composition Element has a semantic version with 3 digits indicating the major.minor.patch number of the version.

Note that a configuration means a full set of parameter values for an Automation Composition Instance.

Change constraints:

1. An Automation Composition or Automation Composition Element in state RUNNING can be changed to a higher patch level or rolled back to a lower patch level. This means that hot changes that do not impact the structure of an Automation Composition or its elements can be executed.
2. An Automation Composition or Automation Composition Element in state PASSIVE can be changed to a higher minor/patch level or rolled back to a lower minor/patch level. This means that structural changes to Automation Composition Elements that do not impact the Automation Composition as a whole can be executed by taking the Automation Composition to state PASSIVE.
3. An Automation Composition or Automation Composition Element in state UNINITIALIZED can be changed to a higher major/minor/patch level or rolled back to a lower major/minor/patch level. This means that where the structure of the entire Automation Composition is changed, the Automation Composition must be uninitialized and reinitialized.
4. If an Automation Composition Element has a minor version change, then its Automation Composition Instance must have at least a minor version change.
5. If an Automation Composition Element has a major version change, then its Automation Composition Instance must have a major version change.

Scalability

The system is designed to be inherently scalable. ACM-R is stateless, all state is preserved in the Instantiated Automation Composition inventory in the database. When the user requests an operation such as an instantiation, activation, passivation, or an uninitialization on an Automation Composition Instance, ACM-R broadcasts the request to participants over DMaaP and saves details of the request to the database. ACM-R does not directly wait for responses to requests.

When a request is broadcast on DMaaP, the request is asynchronously picked up by participants of the types required for the Automation Composition Instance and those participants manage the life cycle of its Automation Composition Element instances. Periodically, each participant reports back on the status of operations it has picked up for the Automation Composition Element instances it controls, together with statistics on the Automation Composition Elements over DMaaP. On reception of these participant messages, ACM-R stores this information to its database.

The participant to use on an Automation Composition can be selected from the registered participants in either of two ways:

1. Runtime-side Selection: ACM-R selects a suitable participant from the list of participants and sends the participant ID that should be used in the Participant Update message. In this case, ACM-R decides on which participant will run the Automation Composition Element instance based on a suitable algorithm. Algorithms could be round robin based or load based.
2. Participant-side Selection: ACM-R sends a list of Participant IDs that may be used in the Participant Update message. In this case, the candidate participants decide among themselves which participant should host the Automation Composition Element instance.

This approach makes it easy to scale Automation Composition life cycle management. As Automation Composition Instance counts increase, more than one ACM-R can be deployed and REST/supervision operations on Automation Composition Instances can run in parallel. The number of participants can scale because an asynchronous broadcast mechanism is used for runtime-participant communication and there is no direct connection or communication channel between participants and ACM-R servers. Participant state, Automation Composition Instance state, and Automation Composition Element state is held in the database, so any ACM-R server can handle operations for any participant. Because many participants of a particular type can be deployed and participant instances can load balance Automation Composition element instances for different Automation Composition Instances of many types across themselves using a mechanism such as a Kubernetes cluster.

Sandboxing and API Gateway Support
At runtime, interaction between ONAP platform services and application microservices are relatively unconstrained, so interactions between Automation Composition Element instances for a given Automation Composition Instance remain unconstrained. A proposal to support access-controlled access to and between ONAP services will improve this. This can be complemented by intercepting and controlling services accesses between Automation Composition Elements for Automation Composition Instances for some/all Automation Composition types.

API gateways such as Kong have emerged as a useful technology for exposing and controlling service endpoint access for applications and services. When an Automation Composition Type is onboarded, or when Automation Composition Instances are created in the Participants, ACM-R can configure service endpoints between Automation Composition Elements to redirect through an API Gateway. Authentication and access-control rules can then be dynamically configured at the API gateway to support constrained access between Automation Composition Elements and Automation Composition Instances.

The diagram below shows the approach for configuring API Gateway access at Automation Composition Instance and Automation Composition Element level.

At design time, the Automation Composition type definition specifies the type of API gateway configuration that should be supported at Automation Composition and Automation Composition Element levels.

At runtime, ACM-R can configure the API gateway to enable (or deny) interactions between Automation Composition Instances and individually for each Automation Composition Element. All service-level interactions in/out of an Automation Composition Element, except that to/from the API Gateway, can be blocked by networking policies, thus sandboxing an Automation Composition Element and an entire Automation Composition Instance if desired. Therefore, an Automation Composition Element will only have access to the APIs that are configured and enabled for the Automation Composition Element /Instance in the API gateway.

For some Automation Composition Element Types the Participant can assist with service endpoint reconfiguration, service request/response redirection to/from the API Gateway, or annotation of requests/responses.

Once the Automation Composition instance is instantiated on participants, the participants configure the API gateway with the Automation Composition Instance level configuration and with the specific configuration for their Automation Composition Element.

Monitoring and logging of the use of the API gateway may also be provided. Information and statistics on API gateway use can be read from the API gateway and passed back in monitoring messages to ACM-R.
Additional isolation and execution-environment sandboxing can be supported depending on the Automation Composition Element Type. For example: ONAP policies for given Automation Composition Instances/Types can be executed in a dedicated PDP engine instances; DCAE or K8S-hosted services can executed in isolated namespaces or in dedicated workers/clusters; etc..

5 APIs and Protocols

The APIs and Protocols used by ACM for Automation Compositions are described on the pages below:

1. System Level Dialogues
2. Defining Automation Compositions in TOSCA for ACM
3. REST APIs for ACM Automation Compositions
4. The ACM Automation Composition Participant Protocol

6 Design and Implementation

The design and implementation of ACM is described for each executable entity on the pages below:

1. The ACM Runtime Server
2. AC Participants
3. The ACM Client
4. Building and running ACM
5. Testing ACM