ANIMA
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

P. Peloso
L. Ciavaglia
Nokia
March 21, 2016

A Day in the Life of an Autonomic Function
draft-peloso-anima-autonomic-function-01.txt

Abstract

While autonomic functions are often pre-installed and integrated with the network elements they manage, this is not a mandatory condition. Allowing autonomic functions to be dynamically installed and to control resources remotely enables more versatile deployment approaches and enlarges the application scope to virtually any legacy equipment. The analysis of autonomic functions deployment schemes through the installation, instantiation and operation phases allows constructing a unified life-cycle and identifying new required functionality. Thus, the introduction of autonomic technologies will be facilitated, the adoption much more rapid and broad. Operators will benefit from multi-vendor, inter-operable autonomic functions with homogeneous operations and superior quality, and will have more freedom in their deployment scenarios.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

Table of Contents

1. Problem statement

While autonomic functions are often pre-installed and integrated with the network elements they manage, this is not a mandatory condition. Allowing autonomic functions to be dynamically installed and to control resources remotely enables more versatile deployment approaches and enlarges the application scope to virtually any legacy equipment. The analysis of autonomic functions deployment schemes through the installation, instantiation and operation phases allows constructing a unified life-cycle and identifying new required functionality.

An Autonomic Service Agent (ASA) controls resources of one or multiple Network Elements (NE), e.g. the interfaces of a router for a Load Balancing ASA. An ASA is a software, thus an ASA need first to be installed and to execute on a host machine in order to control resources.

There are 3 properties applicable to the installation of ASAs:

The dynamic installation property allows installing an ASA on demand, on any hosts compatible with the ASA.

The decoupling property allows controlling resources of a NE from a remote ASA, i.e. an ASA installed on a host machine different from the resources' NE.

The multiplicity property allows controlling multiple sets of resources from a single ASA.

These three properties provide the operator a great variety of ASA deployment schemes as they decorrelate the evolution of the infrastructure layer from the evolution of the autonomic function layer. Depending on the capabilities (and constraints) of the infrastructure and of the autonomic functions, the operator can devise the schemes that will better fit to its deployment objectives and practices.

Based on the above definitions, the ASA deployment process can be formulated as a multi-level/criteria matching problem.

The primary level, present in the three phases, consists in matching the objectives of the operator and the capabilities of the infrastructure. The secondary level criteria may vary from phase to

phase. One goal of the document is thus to identify the specific and common functionality among these three phases.

This draft will explore the implications of these properties along each of the 3 phases namely Installation, Instantiation and Operation. This draft will then provide a synthesis of these implications in requirements for functionalities and life-cycle of ASAs. Beforehand, the following section will deal with the network operator's point of view with regards of autonomic networks.

2. Motivations from an operator viewpoint

Only few operators would dare relying on a pure autonomic network, without setting objectives to it. From an operator to the other, the strategy of network management vary, as much for historical reasons (experience, best-practice, tools in-place, organization), as much for differences in the operators goals (business, trade agreements, politics, risk policy). Additionally, network operators do not necessarily perform a uniform network management across the different domains composing their network infrastructure. Hence their objectives in terms of availability, load, and dynamics vary depending on the nature of the domains and of the types of services running over each of those domains.

To manage the networks according to the above variations, ASAs need to capture the underlying objectives implied by the operators. The instantiation phase is the step in-between installation and operation, where the network operator determine the initial ASA behavior according to its objectives. This step allows the network operator to determine which ASAs should execute on which domains of its network, with appropriate settings. At this stage, thanks to the intent-policy setting objectives to groups of ASAs, the network management should become far simpler (and less error-prone) than setting low-level configurations for each individual network resources.

2.1. Illustration of increasingly constraining operator's objectives

This paragraph describes the following example of operator intents with regards to deployments of autonomic functions. The autonomic function involved is a load balancing function, which uses monitoring results of links load to autonomously modify the links metrics in order to balance the load over the network. The example is divided into steps corresponding to an increasing implication of the operator in the definition of objectives/intents to the deployment of autonomic functions:

Step 1 The operator operates its network and benefits from the autonomic function on the nodes which have the installed ASAs.

Step 2 Then the operator, specifies to the autonomic function an objective which is to achieve the maximum number of links with a load below 6O%.

Step 3 The network is composed of five domains, a core transport network and four metropolitan networks, each interconnected through the core network, the operator sets a different objective to the autonomic function for each of the five domain.

Step 4 As inside metropolitan domains the traffic variations are steeper and happen in a periodic fashion contrary to the traffic in the core domain, the network operators installs an additional autonomic function inside each of these domains. This autonomic function is learning the traffic demands in order to predict traffic variations. The operators instructs the load balancing function to augment its monitored input with the traffic predictions issued by the newly installed autonomic function.

Step 5 As the algorithm of the load balancing autonomic function is relying on interactions between autonomic function agents, the operator expects the interactions to happen in-between ASAs of each domain, hence the load will be balanced inside each of the domain, while previously it would have been balanced over the whole network uniformly.

Step 6 Finally, the network operator has purchased a new piece of software corresponding to an autonomic function achieving load balancing with a more powerful algorithm. For trial sake, he decides to deploy this new load balancing function instead of the previous one on one of its four metropolitan domains.

This short example illustrates some specificities of deployment scenarios, the sub-section below sets itself at providing a more exhaustive view of the different deployment scenarios.

2.2. Deployment scenarios of autonomic functions

The following scenarios illustrate the different ways the autonomic functions could be deployed in an ANIMA context. Subsequently, requirements for the autonomic functions and requirements these autonomic functions impose on other components of the ANIMA ecosystem are listed.

These various deployment scenarios are better understood by referring to the High level view of an Autonomic Network, Figure 1 of

[I-D.behringer-anima-reference-model]. The figure is slightly extended for the purpose of the demonstration as follows:

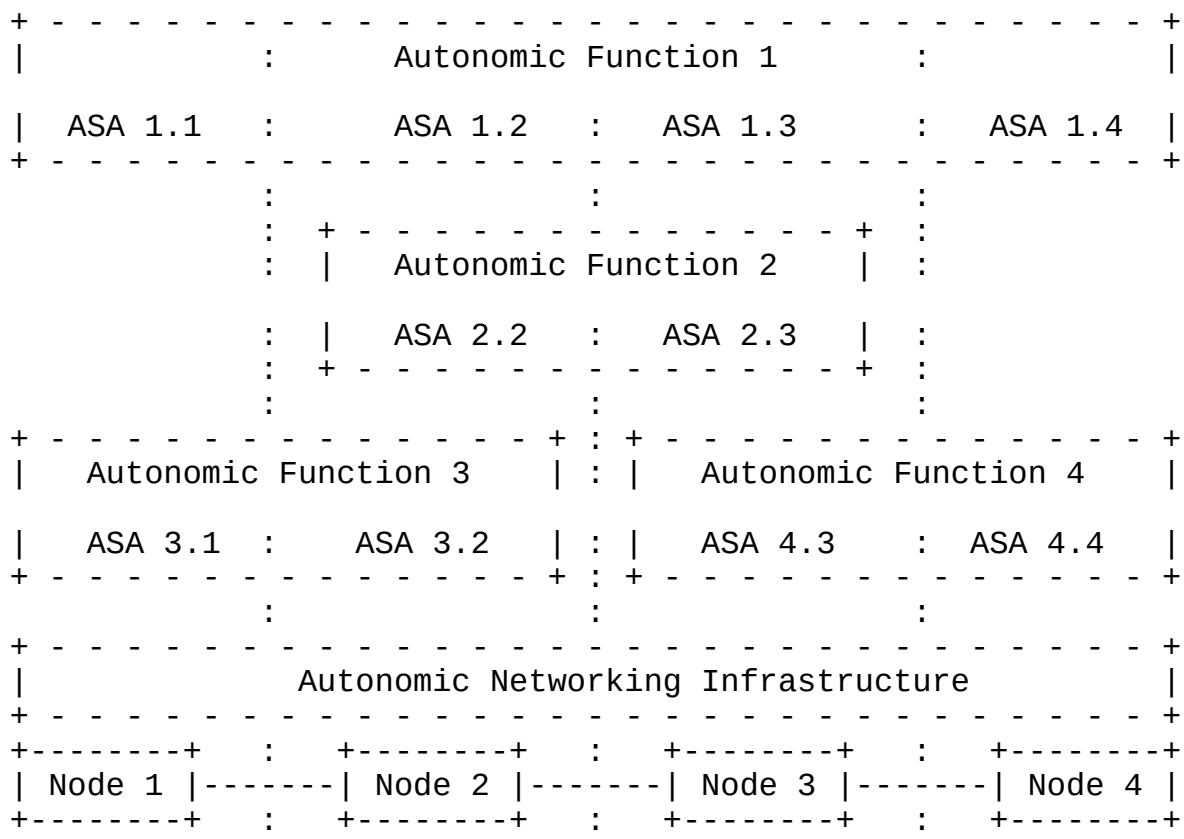

Figure 1: High level view of an Autonomic Network

Figure 1 depicts 4 Nodes, 4 Autonomic Functions and 10 Autonomic Service Agents. Let’s list assumptions with regards of these elements.

Starting with nodes,

each may be either an Unconstrained Autonomic Node, a Constrained Autonomic Node (or even a legacy one?),

they may well be of different models (or having different software versions),

they may well be of different equipment vendors,

they may well be of different technologies (some may be IP routers, some may be Ethernet switches or OTN switches...).

Pursuing with Autonomic Functions,

they may well have different objectives (one could target automatic configuration of OSPF-TE, while another one is optimizing traffic load), but they may well have identical objectives as two could optimize energy consumption (possibly on different areas as function 3 and function 4),

each may be composed of no more than one ASA (either because the function is responsible for a single node or because the function relies on a centralized implementation),

each may well be composed of different sort of ASAs, meaning the software is different (either because their version number is different, or because the software provider is different, or because their respective nodes/equipments differ or because the role of each agent is different),

[Observation] Depending on the implementation the same piece of software may fulfill different roles or each role may come from a different from a different piece of code,

each has reached a given organization, meaning an organized set of ASAs in charge of a set of nodes ()whether formalized or not), this organization may either come from the piece of software itself (e.g. embedding a self-organization process) or come from directions of the network operator (e.g. through intents/policies, or through deployment instructions)

each may work internally in a peer to peer fashion (where every agents have the same prerogatives) or in hierarchical fashion (where some agents have some prerogatives over other) [this option is a good example of role differences],

each having its scope of work in terms of objective to reach and area/space/part of the network to manage.

Completing with individual Autonomic Service Agents, those are pieces of software:

embedded inside the node/equipment OS (hence present since the bootstrap or OS update of the equipment),

running in a machine different than the node (this could be a node controller or any other host or virtual machine)

[Observation] In the latter case, the ASA would likely require external credentials to interact with the node,

directly monitoring and configuring the equipment (likely requires the ASA to be embedded) or through a management interface of the equipment (e.g. SNMP, TL1, Q3, NetConf) or through an equipment controller (e.g. SDN paradigm) or through a network manager (e.g. using the north interface of the manager)

either activated at start-up or as the result of a management action,

may be installed (either inside the equipment or on a different machine) when requested by an operator from a software origin (e.g. a repository in the ACP, a media)

provided by the same vendor as the equipment it manages or by any third party (like another equipment vendor, a management software vendor, an open-source initiative or the operator software team),

sharing a technical objective with the other ASAs of the Autonomic Function they belong, (or at least a similar one)?

can it contains multiple technical objective?

must the technical objective be intrinsic or can it be set by a managing entity (a network operator or a management system)?

The last three points being largely questionable are marked as questions.

The figure below illustrates how an ASA interacts with a node that the ASA manages. The left side depicts external interactions, through exchange of commands towards interfaces either to the node OS (e.g. via SNMP or NetConf), or to the controller (e.g. (G)MPLS, SDN, ...), or to the NMS. The right side depicts the case of the ASA embedded inside the Node OS.

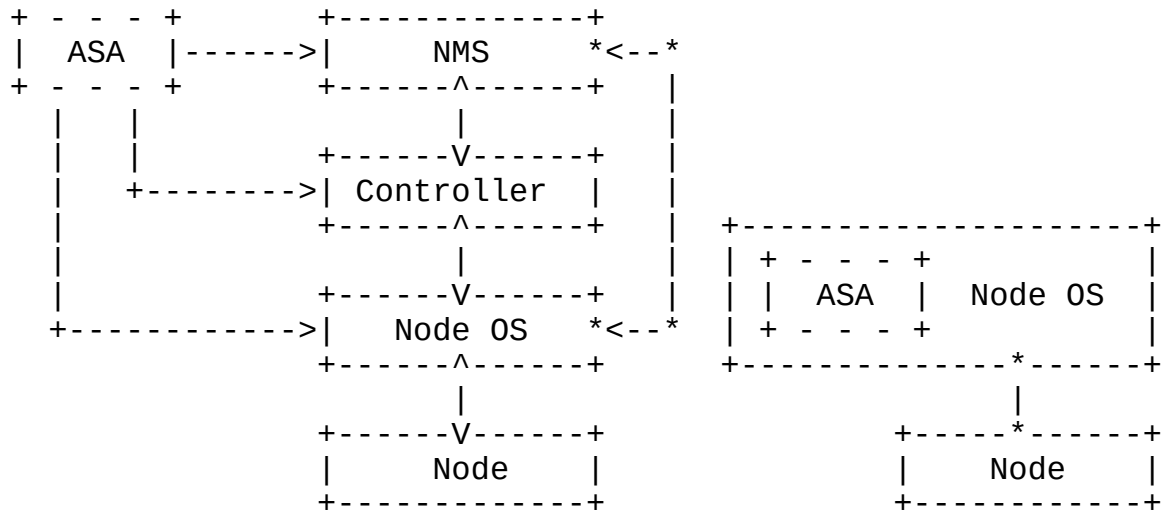

Figure 2: Interaction possibilities between ASA and Resources

2.3. Operator's requirements with regards to autonomic functions

Regarding the operators, at this point we can try to list few requirements they may have with regards with the Autonomic Functions and their management...

Being capable to set those functions a scope of work in term of area of duty,

Being capable to monitor the actions taken by the autonomic functions, and which are its results (performance with regards to the function KPIs)

Being capable to halt/suspend the execution of an Autonomic function (either because the function is untrusted, or because an operation on the network is to be conducted without interference from the autonomic functions, etc...)

Being capable to configure the autonomic functions by adjusting the parameters of the function (e.g. a Traffic Engineering autonomic function may achieve a trade-off between congestion avoidance and electrical power consumption, hence this function may be more or less aggressive on the link load ratio, and the network operator certainly has his word to say in setting this cursor.

3. Installation phase

Before being able to instantiate and run ASAs, the operator must first provision the infrastructure with the sets of ASA software corresponding to its needs and objectives. The provisioning of the infrastructure is realized in the installation phase and consists in installing (or checking the availability of) the pieces of software of the different ASA classes in a set of Installation Hosts.

As mentioned in the Problem statement section, an Autonomic Function Agent (ASA) controls resources of one or multiple Network Elements (NE), e.g. the interfaces of a router for a Load Balancing ASA. An ASA is a software, thus an ASA need first to be installed and to execute on a host machine in order to control resources.

There are 3 properties applicable to the installation of ASAs:

The dynamic installation property allows installing an ASA on demand, on any hosts compatible with the ASA.

The decoupling property allows controlling resources of a NE from a remote ASA, i.e. an ASA installed on a host machine different from the resources' NE.

The multiplicity property allows controlling multiple sets of resources from a single ASA.

These three properties are very important in the context of the installation phase as their variations condition how the ASA class could be installed on the infrastructure.

3.1. Operator's goal

In the installation phase, the operator's goal is to install ASA classes on Installation Hosts such that, at the moment of instantiation, the corresponding ASAs can control the sets of target resources. The complexity of the installation phase come from the number of possible configurations for the matching between the ASA classes capabilities (e.g. what types of resources it can control, what types of hosts it can be installed on...), the Installation Hosts capabilities (e.g. support dynamic installation, location and reachability...) and the operator's needs (what deployment schemes are favored, functionality coverage vs. cost trade-off...).

For example, in the coupled mode, the ASA host machine and the network element are the same. The ASA is installed on the network element and control the resources via interfaces and mechanisms internal to the network element. An ASA MUST be installed on the

network element of every resource controlled by the ASA. The identification of the resources controlled by an ASA is straightforward: the resources are the ones of the network element.

In the decoupled mode, the ASA host machine is different from the network element. The ASA is installed on the host machine and control the resources via interfaces and mechanisms external to the network element. An ASA can be installed on an arbitrary set of candidate Installation hosts, which can be defined explicitly by the network operator or according to a cost function. A key benefit of the decoupled mode is to allow an easier introduction of autonomic functions on existing (legacy) infrastructure. The decoupled mode also allows de-correlating the installation requirements (compatible host machines) from the infrastructure evolution (NEs addition and removal, change of NE technology/version...).

The operator’s goal may be defined as a special type of intent, called the Installation phase intent. The details of the content and format of this proposed intent are left open and for further study.

3.2. Installation phase inputs and outputs

Inputs are:

[ASA class of type_x] that specifies which classes ASAs to install,

[Installation_target_Infrastructure] that specifies the candidate Installation Hosts,

[ASA class placement function, e.g. under which criteria/constraints as defined by the operator]
that specifies how the installation phase shall meet the operator’s needs and objectives for the provision of the infrastructure. In the coupled mode, the placement function is not necessary, whereas in the decoupled mode, the placement function is mandatory, even though it can be as simple as an explicit list of Installation hosts.

The main output of the installation phase is an up-to-date directory of installed ASAs which corresponds to [list of ASA classes] installed on [list of installation Hosts]. This output is also useful for the coordination function and corresponds to the static interaction map.

The condition to validate in order to pass to next phase is to ensure that [list of ASA classes] are well installed on [list of installation Hosts]. The state of the ASA at the end of the installation phase is: installed. (not instantiated). The following

commands or messages are foreseen: install(list of ASA classes, Installation_target_Infrastructure, ASA class placement function), and un-install (list of ASA classes).

4. Instantiation phase

Once the ASAs are installed on the appropriate hosts in the network, these ASA may start to operate. From the operator viewpoint, an operating ASA means the ASA manages the network resources as per the objectives given. At the ASA local level, operating means executing their control loop/algorithm.

But right before that, there are two things to take into consideration. First, there is a difference between 1. having a piece of code available to run on a host and 2. having an agent based on this piece of code running inside the host. Second, in a coupled case, determining which resources are controlled by an ASA is straightforward (the determination is embedded), in a decoupled mode determining this is a bit more complex (hence a starting agent will have to either discover or be taught it).

The instantiation phase of an ASA covers both these aspects: starting the agent piece of code (when this does not start automatically) and determining which resources have to be controlled (when this is not obvious).

4.1. Operator’s goal

Through this phase, the operator wants to control its autonomic network in two things:

1 determine the scope of autonomic functions by instructing which of the network resources have to be managed by which autonomic function (and more precisely which class e.g. 1. version X or version Y or 2. provider A or provider B),

2 determine how the autonomic functions are organized by instructing which ASAs have to interact with which other ASAs (or more precisely which set of network resources have to be handled as an autonomous group by their managing ASAs).

Additionally in this phase, the operator may want to set objectives to autonomic functions, by configuring the ASAs technical objectives.

The operator’s goal can be summarized in an instruction to the ANIMA ecosystem matching the following pattern:

[ASA of type_x instances] ready to control
[Instantiation_target_Infrastructure] with
[Instantiation_target_parameters]

4.2. Instantiation phase inputs and outputs

Inputs are:

[ASA of type_x instances] that specifies which are the ASAs to be targeted (and more precisely which class e.g. 1. version X or version Y or 2. provider A or provider B),

[Instantiation_target_Infrastructure] that specifies which are the resources to be managed by the autonomic function, this can be the whole network or a subset of it like a domain a technology segment or even a specific list of resources,

[Instantiation_target_parameters] that specifies which are the technical objectives to be set to ASAs (e.g. an optimization target)

Outputs are:

[Set of ASAs - Resources relations] describing which resources are managed by which ASA instances, this is not a formal message, but a resulting configuration of a set of ASAs,

4.3. Instantiation phase requirements

The instructions described in section 4.2 could be either:

sent to a targeted ASA In which case, the receiving Agent will have to manage the specified list of [Instantiation_target_Infrastructure], with the [Instantiation_target_parameters].

broadcast to all ASAs In which case, the ASAs would collectively determine from the list which Agent(s) would handle which [Instantiation_target_Infrastructure], with the [Instantiation_target_parameters].

This set of instructions can be materialized through a message that is named an Instance Mandate. Instance Mandates are described in the requirements part of this document, which lists the needed fields of such a message (see Section 6.3 - ASA Instance Mandate).

The conclusion of this instantiation phase is a ready to operate ASA (or interacting set of ASAs), then this (or those) ASA(s) can

describe themselves by depicting which are the resources they manage and what this means in terms of metrics being monitored and in terms of actions that can be executed (like modifying the parameters values). A message conveying such a self description is named an Instance Manifest. Instance Manifests are described in the requirements part of this document, which lists the needed fields of such a message (see Section 6.4 - ASA Instance Manifest).

Though the operator may well use such a self-description "per se", the final goal of such a description is to be shared with other ANIMA entities like:

- the coordination entities (see [I-D.ciavaglia-anima-coordination] - Autonomic Functions Coordination)
- collaborative entities in the purpose of establishing knowledge exchanges (some ASAs may produce knowledge or even monitor metrics that other ASAs cannot make by themselves why those would be useful for their execution) (see knowledge exchange items in Section 5 - Operation phase)

5. Operation phase

Note: This section is to be further developed in future revisions of the document.

During the Operation phase, the operator can:

Activate/Deactivate ASA: meaning enabling those to execute their autonomic loop or not.

Modify ASAs targets: meaning setting them different objectives.

Modify ASAs managed resources: by updating the instance mandate which would specify different set of resources to manage (only applicable to decouples ASAs).

During the Operation phase, running ASAs can interact the one with the other:

in order to exchange knowledge (e.g. an ASA providing traffic predictions to load balancing ASA)

in order to collaboratively reach an objective (e.g. ASAs pertaining to the same autonomic function targeted to manage a network domain, these ASA will collaborate - in the case of a load balancing one, by modifying the links metrics according to the neighboring resources loads)

During the Operation phase, running ASAs are expected to apply coordination schemes

then execute their control loop under coordination supervision/ instructions

6. Autonomic Function Agent specifications

6.1. Life-cycle

Based on the phases described above, this section defines formally the different states experienced by Autonomic Function Agents during their complete life-cycle.

The drawing of the life-cycle presented below shows both the states and the events/messages triggering the state changes. For simplification purposes, this sketch does not display the transitory states which correspond to the handling of the messages.

The installation and Instantiation phase will be concluded by ASA reaching respectively Installed and Instantiated states.

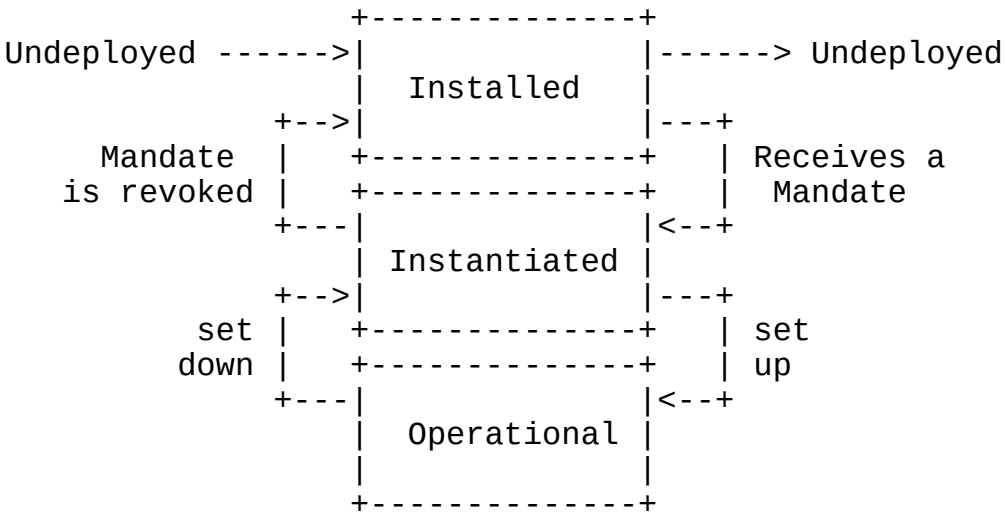

Figure 3: Life cycle of an Autonomic Function Agent

Here are described the successive states of ASA.

Undeployed - In this "state", the Autonomic Function Agent is a mere piece of software, which may not even be available on any host.

Installed - In this state, the Autonomic Function Agent is available on a (/multiple) host(s), and after having shared its ASA class Manifest (which describes in a generic way independently of the deployment how the ASA would work). In this state the ASA is waiting for an ASA Instance Mandate, to determine which resources ti manage (when the ASA is strictly coupled to resources [e.g. part of a Node OS], there is no need to wait for an instance mandate, the target resources being intrinsically known).

Instantiated - In this state the Autonomic Function Agent has the knowledge of which resources it is meant to manage. In this state the ASA is expecting a set Up message in order to start executing its autonomic loop. From this state on the ASA can share an Instance Manifest (which describes how the ASA instance is going to work).

Operational - In this state, ASAs are executing their autonomic loop, hence acting on network, by modifying resources parameters. A set down message will turn back the ASA in an Instantiated state.

The messages are described in the following sections.

6.2. ASA Class Manifest

An ASA class is a piece of software that contains the computer program of an Autonomic Function Agent.

In order to install and instantiate appropriately an autonomic function in its network, the operator needs to know which are the characteristics of this piece of software.

This section details a format for an ASA class manifest, which is (a machine-readable) description of both the autonomic function and the piece of code that executes the function.

Field Name	Type	Description
ID	Struct	A unique identifier made out of at least a Function Name, Version and Provider Name (and Release Date).
Description	Struct	A multi-field description of the function performed by the ASA, it is meant to be read by the operator and can point to URLs, user-guides, feature descriptions.
Installation	3 Booleans	Whether the ASA is dynamically

properties		installable, can be decoupled from the NE and can manage multiple resources from a single instance (see Section 1 - Problem statement).
Possible Hosts	OS...	Lists the OS/Machines on which the ASA can be executed. [Only if ASA is dynamically installable]
Network Segment	NetSegment...	Lists the network segments on which the autonomic function is applicable (e.g. IP backbone versus RAN).
Manageable Equipments	Equipments...	Lists the nodes/resources that this piece of code can manage (e.g. ALU 77x routers, Cisco CRS-x routers, Huawei NEXE routers).
Autonomic Loop Type	Enum	States what is the type of loop MAPE-K and whether this loop can be halted in the course of its execution.
Acquired Inputs	Raw InfoSpec...	Lists the nature of information that an ASA agent may acquire from the managed resource(s) (e.g. the links load).
External Inputs	Raw InfoSpec...	Lists the nature of information that an ASA agent may require/wish from other ASA in the ecosystem that could provide such information/knowledge.
Possible Actions	Raw ActionSpec	Lists the nature of actions that an ASA agent may enforce on ASA the managed resource(s) (e.g. modify the links metrics).
Technical Objectives Description	Technical Objective Spec...	Lists the type of technical objectives that can be handled/received by the ASA (e.g. a max load of links).

Table 1: Fields of ASA class manifest

6.3. ASA Instance Mandate

An ASA instance is the ASA agent: a running piece of software of an ASA class. A software agent is a persistent, goal-oriented computer program that reacts to its environment and interacts with other elements of the network.

In order to install and instantiate appropriately an autonomic function in its network, the operator may specify to ASA instances what they are supposed to do: in term of which resources to manage and which objective to reach.

This section details a format for an ASA Instance Mandate, which is (a machine-readable) set of instructions sent to create autonomic functions out of ASA.

Field Name	Type	Description
ASA class Pattern	Struct	A pattern matching the ID (or part of it) of ASAs being the target of the Mandate. This field makes sense only for broadcast mandates (see end of this section).
Managed Resources	ResourcesId...	The list of resources to be managed by the ASA (e.g. their IP@ or MAC@ or any other relevant ID).
ID of Coord	Interface Id	The interface to the coordination system in charge of this autonomic function.
Reporting Policy	Policy	A policy describing which entities expect report from ASA, and which are the conditions of these reports (e.g. time wise and content wise)

Table 2: Fields of ASA instance mandate

An ASA instance mandate could be either:

sent to a targeted ASA In which case, the receiving Agent will have to manage the specified list of resources,

broadcast to all ASA In which case, the ASAs would collectively determine which agent would handle which resources from the list, and if needed (and feasible) this could also trigger the dynamic installation/instantiation of new agents (ACP should be capable of bearing such scenarios).

6.4. ASA Instance Manifest

Once the ASAs are properly instantiated, the operator and its managing system need to know which are the characteristics of these ASAs.

This section details a format for an ASA instance manifest, which is (a machine-readable) description of either an ASA or a set of ASAs gathered into an autonomic function.

Field Name	Type	Description
ASA Class ID	Struct	A unique identifier made out of at least a Function Name, Version and Provider Name (and Release Date).
ASA Instance ID	Long	A unique Id of the ASA instance (if the ASA instance manifest gathers multiple ASAs working together, this would be a list).
Hosts	Resource ID	ID of the Machines on which the ASA executes.
Managed Resources	ResourcesId...	The list of resources effectively managed by the ASA (e.g. their IP@ or MAC@ or any other relevant ID).
Acquired Inputs	Instance InfoSpec...	Lists information that this ASA agent may acquire from the managed resource(s) (e.g. the links load over links with ID x and y).
External Inputs	Instance InfoSpec...	Lists information that this ASA agent requires from the ecosystem (e.g. the links load prediction over links with ID x and y).
Possible Actions	Instance ActionSpec	Lists actions that this ASA agent may enforce on its managed resource(s) (e.g. modify the links metrics over links x and y).

Table 3: Fields of ASA instance manifest

7. Implication for other ANIMA components

7.1. Additional entities for ANIMA ecosystem

In the previous parts of this document, we have seen successive operations pertaining to the management of autonomic functions. These phases involve different entities such as the ASAs, the ASA Hosts and the ASA Management function. This function serves as the interface between the network operator and its managed infrastructure (i.e. the autonomic network). The ASA management function distributes instructions to the ASAs such as the ASA Instance Mandate, ASA set up/set down commands and also trigger the ASA

installation inside ASA Hosts. This function is likely to be co-located or integrated with the function responsible for the management of the Intents.

In this first version, we do not prescribe any requirements on the way the ASA Management function should be implemented, neither the various deployment options of such a function and neither on the way ACP or GRASP could be extended to interact with this function. We believe these design and specifications work should be first discussed and analyzed by the working group.

7.2. Requirements for GRASP and ACP messages

GRASP and ACP seems to be the best (and currently only) candidates to convey the following messages between the ASA Management function and the ASAs:

ASA Class Manifest

ASA Instance Mandate (and Revoke Mandate)

ASA Instance Manifest

Set Up and Set Down messages

These section concludes with requests to GRASP protocol designers in order to handle the 3 last messages of the list above. These 3 messages form the minimal set of features needed to guarantee some control on the behavior of ASAs to network operators.

A mechanism similar to the bootstrapping one would usefully achieve discovery of pre-installed ASAs, and possibly provide those with a default Instance Mandate.

A mechanism to achieve dynamic installation of ASAs compatible with ACP and GRASP remains to be identified.

In the case of decoupled ASAs, even more for the ones supporting multiplicity, when a Mandate is broadcast (i.e. requesting the Instantiation of an autonomic function to manage a bunch of resources), these ASAs require synchronization to determine which agent(s) will manage which resources. Proper ACP/GRASP messages supporting such a mechanism have to be identified together with protocol authors.

7.2.1. Control when an ASA runs

To control when an ASA runs (and possibly how it runs), the operator needs the capacity to start and stop ASAs. That is why an imperative command type of message is requested from GRASP.

Additionally this type of message could also be used to specify how the ASA is meant to run, e.g. whether its control loop is subdued to some constraints in terms of pace of execution or rhythm of execution (once a second, once a minute, once a day...)

Below a suggestion for GRASP:

In fragmentary CDDL, an Imperative message follows the pattern:

imperative-message = [M_IMPERATIVE, session-id, initiator, objective]

...

7.2.2. Know what an ASA does to the network

To know what an ASA does to the network, the operator needs to have the information of the elements either monitored or modified by the ASA, hence this ASA should disclose those.

The disclosing should take the form of a ASA Instance Manifest (see Section 6.4 - ASA Instance Manifest), which could be conveyed inside a GRASP discovery message, hence the fields of the ASA Instance Manifest would be conveyed inside the objective.

At this stage there are two options:

The whole manifest is conveyed as an objective.

Each field of the manifest is conveyed as an individual objective, more precisely, the acquired inputs would appear as discovery only, and the modifiable parameters would appear as negotiation objective. The unclear part is the expression of requested fields (when the ASA claims being a client for such objective). Could one of the already existing objective options a good match or should a new one be created.

...

7.2.3. Decide which ASA control which equipment

To determine which ASA controls which equipment (or vice-versa which equipments are controlled by which ASAs), the operators needs to be able to instruct ASA before the end of their bootstrap procedure.

These instructions sent to ASA during bootstrapping should take the format of an ASA Instance Mandate (see Section 6.3 - ASA Instance Mandate). This ASA Instance Mandate are sorts of Intents, and as GRASP is meant to handle Intents in a near future, it would be beneficial to already identify which sort of GRASP message are meant to be used by Intent in order to already define those. An option could be to reuse the Imperative messages defined above.

...

8. Acknowledgments

This draft was written using the xml2rfc project.

This draft content builds upon work achieved during UniverSelf FP7 EU project.

9. IANA Considerations

This memo includes no request to IANA.

10. Security Considerations

TBD

11. References

11.1. Normative References

[I-D.ciavaglia-anima-coordination]
Ciavaglia, L. and P. Peloso, "Autonomic Functions Coordination", draft-ciavaglia-anima-coordination-00 (work in progress), July 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

11.2. Informative References

[I-D.behringer-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., Liu, B., Jeff, J., and J. Strassner, "A Reference Model for Autonomic Networking", draft-behringer-anima-reference-model-04 (work in progress), October 2015.

[RFC7575] Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, DOI 10.17487/RFC7575, June 2015, <http://www.rfc-editor.org/info/rfc7575>.

Authors' Addresses

Peloso Pierre
Nokia
Villarceaux
Nozay 91460
FR

Email: pierre.peloso@nokia.com

Laurent Ciavaglia
Nokia
Villarceaux
Nozay 91460
FR

Email: laurent.ciavaglia@nokia.com