ETSI GR ENI 008 V2.1.1 (2021-03)

Experiential Networked Intelligence (ENI); InTent Aware Network Autonomicity (ITANA)

Reference

DGR/ENI-0013_Net_Autonomicity

Keywords

artificial intelligence, management

Contents

Intellectual Property Rights

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Experiential Networked Intelligence (ENI).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document will discuss various design options, in terms of a set of new stand-alone and/or nested Functional Blocks, for using intent with the ENI System Architecture. This includes accepting, translating and validating intent statements, determining how intent affects the goals and operation of the ENI system, and how it is used by business users, application developers and network administrators.

The present document provides a set of recommendations and conclusions whose main scope is ETSI GS ENI 005 [i.2] Release 2. However, the contents of the present document also affect other GSs & GRs (e.g. ETSI GS ENI 001 [i.1], ETSI GS ENI 002 [i.4] and ETSI GR ENI 004 [i.3]).

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI GS ENI 001 (V3.1.1): "Experiential Networked Intelligence (ENI); ENI use cases".

[i.2] ETSI GS ENI 005 (V2.1.1): "Experiential Networked Intelligence (ENI); System Architecture".

[i.3] ETSI GR ENI 004 (V3.1.1): "Experiential Networked Intelligence (ENI); Terminology for Main Concepts in ENI".

[i.4] ETSI GS ENI 002 (V3.1.1): "Experiential Networked Intelligence (ENI); ENI requirements".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

compiler: computer program that translates computer code written in one programming language (the source language) into another language (the target language) (see ETSI GS ENI 005 [i.2])

NOTE: The term "compiler" is primarily used for programs that translate source code from a high-level programming language to a lower level language.

intent creator: set of authorized entities that is able to create an Intent Policy including the User, APP, OSS, BSS and Orchestrator

NOTE: The Intent Creator submits the Intent Policy to the ENI system through dedicated External Reference Points.

intent policy target: entity whose behaviour is affected by the Intent Policy

NOTE: For the purposes of the present document, this is either the Assisted System (or its Designated Entity) or the ENI System itself.

interpreter: computer program that directly executes instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program (see ETSI GS ENI 005 [i.2])

policyMetadata: data that describes and prescribes policy characteristics and behaviour (see ETSI GS ENI 005 [i.2])

NOTE: Examples of policyMetadata include a time period that this intent policy is valid, as well as version information, including a minimum version that will be used.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
APP	Application (Functional Block)
BS	Base Station
BSS	Business Support Systems
CRUD	Create Read Update Delete
DPI	Deep Packet Inspection
DSL	Domain-Specific Language
EMS	Element Management System
ENI	Experiential Networked Intelligence
EPC	Evolved Packet Core
FB	Functional Block
IDMS	Intent-Driven Management System
IMS	IP Multimedia Subsystem
IP	Internet Protocol
ITANA	InTent Aware Network Autonomicity
NFV	Network Functions Virtualisation
NR	New Radio
OAM	Operating and Maintenance
OPEX	OPeration EXpediture
OSS	Operations Support System
RAN	Radio Access Network
SDO	Standard Developing Organization
SFC	Service Function Chain
SLA	Service Level Agreement
SON	Self-Organizing Network
UID	Unique IDentifier
VNF	Virtual Network Function

4 Introduction

Intent Policy is introduced and defined in clause 6.3.9.3.2 of ETSI GS ENI 005 [i.2], which uses a restricted natural language (e.g. external DSL) to express the goals of the policy, without describing how to accomplish the goals. The Intent Policy concept could be applied in various network domains, such as a wireless network and a data centre.

Intent policies enable different users of the ENI System to author policies in a form that a particular constituency of users understands. For example, business users are able to write Intent Policies using business terms without having to use a language that they do not normally use, such as a general-purpose programming language like Java or Python.

The Policy Continuum (see clause 6.3.9.6.2 of ETSI GS ENI 005 [i.2]) is used to formally differentiate between the needs of different constituencies in defining and expressing policy. Each constituency defines a common set of concepts and terminology. Hence, an Intent Policy Rule for one constituency may be different in structure and content than an Intent Policy Rule for a different constituency. More importantly, since an Intent statement does not specify how it should be implemented, every intent statement needs to be translated to a more concrete form for validation and processing. For example, Business needs are generated from many types of users, such as business users and application developers. Hence, the ENI system uses the Policy Continuum to represent the different constituencies served and to translate an intent policy at a higher level of abstraction to a policy at a lower level of translation. Such a translation is performed one or more times (e.g. to go from a business view to a system view to an administrator view).

5 Impact of Intent Policies on the System Architecture

5.1 Architecture Enhancement for Intent Policies

5.1.1 Scope of Intent Policies

There are two different ways that Intent Policies can be used:

1) An External Entity (e.g. an Operator) sends an Intent Policy to the ENI System that affects the behaviour of the Assisted System (or its Designated Entity).

2) An External Entity sends a Policy (of any type) to the ENI System that affects the behaviour of the ENI System.

In each case, the External Entity sends an Intent Policy. The ENI System will translate the Intent Policy, process it, and then produces a set of recommendations and/or commands that realize the Intent Policy. This is done by the Policy Management Functional Block of the ENI System. The difference is the target of the Intent Policy:

1) If the target is the *Assisted System* (or its Designated Entity), then the set of recommendations and/or commands produced are sent to the Denormalisation and Output Generation Functional Blocks, where they are denormalised and formatted. They are then sent to the API Broker, which transmits them to the Assisted System (or its Designated Entity).

2) If the target is the *ENI System*, then the set of recommendations and/or commands produced are sent to the set of affected Functional Blocks of the ENI System.

5.1.2 Adding an Intent Translation Functional Block

A new Functional Block named Intent Translation is introduced to support the use of an Intent Policy process that includes intent translation, intent assurance and the lifecycle management of Intent Policy.

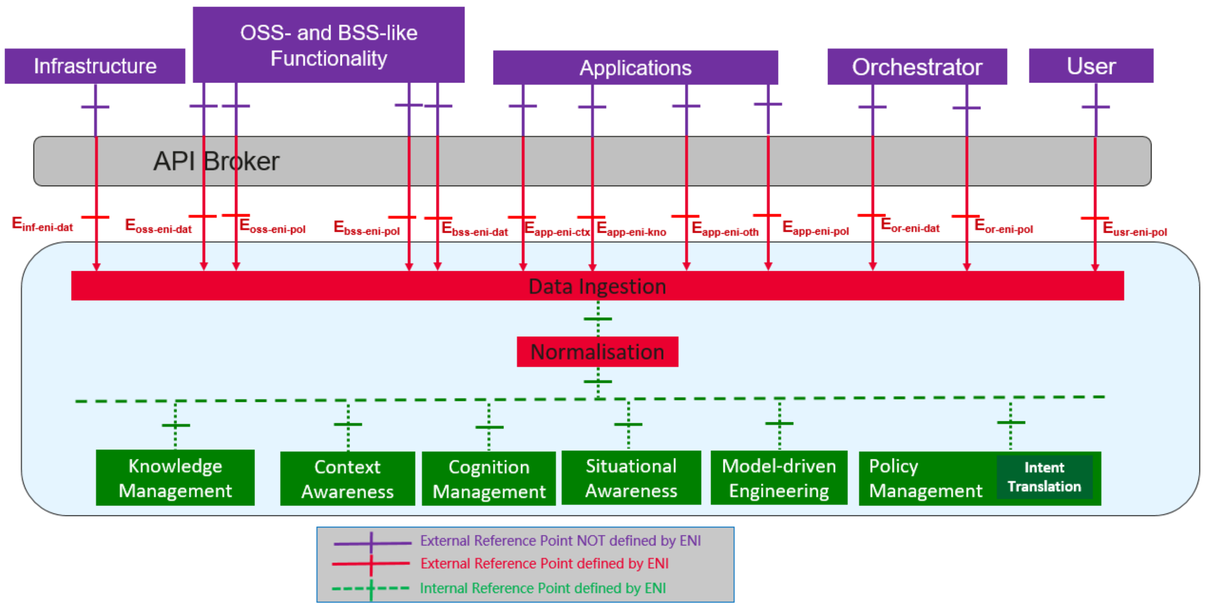

Figure 5-1: Architecture Enhancement for Intent Policy with its Input Reference Points

In this context, the following definitions and considerations apply:

- **Intent Translation Functional Block:** a Functional Block recommended to be added within the Policy Management Functional Block that takes part in the process of Intent Translation. It performs lexical analysis, syntactic analysis, semantic analysis and augmentation, either parsing or compiling and, optionally, interpretation of the Intent Policy.

- **Process of Intent Translation:** the procedure to translate the Intent Policy to the desired format executed internally by the ENI System or transformed into recommendations/commands sent to the Assisted System. This process is discussed in more detail in clause 5.2.

- **Process of Intent Assurance:** the procedure to maintain and monitor the execution of an Intent Policy, including detecting and reporting any conflicts and failures. The use of statistics and possibly analytics to summarize these aspects of Policy will ensure that the requirements of the Intent Creator (e.g. User/APP/OSS-/ and BSS-like Functionality), see clause 5.2.2 regarding its definition, are satisfied. This process is discussed in more detail in clause 5.3.

- **Operational management of Intent Policy:** Create Read Update Delete (CRUD) commands are operations that affect the state of Intent Policies. Intent policies are sent from an External Entity or the Intent Creator (see clause 5.4 for more information) to the ENI System. CRUD Intent Policies are used to manage the Assisted System (or its Designated Entity) or the ENI System itself.

- **Intent knowledge:** knowledge that is used in the process of Intent Translation. It contains the relevant policyMetadata (e.g. a time period that this intent policy is valid, as well as version information, including a minimum version that can be used) and the generated new knowledge (e.g. word and phrase processing and substitution to translate the original Intent Policy into a form understood by the ENI System for a specific domain). Metadata and knowledge are stored in the model repository and knowledge repository defined in ETSI GS ENI 005 [i.2] within the Repository Management Functional Block, respectively.

NOTE: The original form of an Intent Policy uses a restricted natural language. This is likely to contain both ambiguities in the meaning of the Intent Policy as well as terms familiar to the Intent Creator that need to be translated to a form that can be understood by the entities affected by this Intent Policy. This is done by the ENI System. Recommendations and/or commands are produced by the Model-Driven Engineering FB, which are then packaged as ENI Policies by the Policy Management FB. If the target of the Intent Policy was the ENI System itself, then no further processing is required, and the ENI System will execute those recommendations and/or commands. If the target of the Intent Policy was the Assisted System (or its Designated Entity), then the transformed Intent Policy is denormalised and formatted, and then sent to the API Broker. The API Broker sends the Intent Policy to the Assisted System (or its Designated Entity).

5.1.3 Reference Points for Intent Policies Implementation

The internal reference points involved in the Intent Policies process are shown in Figure 5-2.

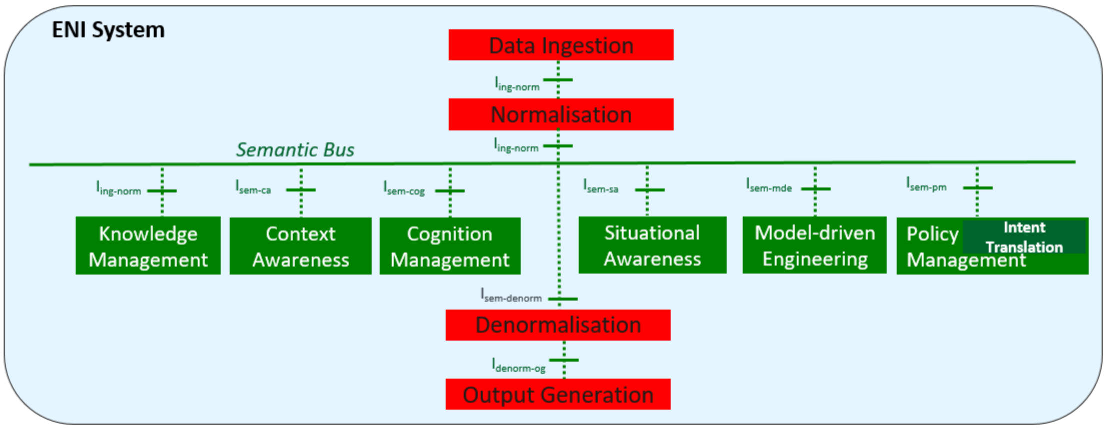

Figure 5-2: Overview of the ENI Internal Reference Points

Table 5-1, depicted further down in clause 5.2.3.2, provides brief descriptions of the Intent Policy and associated information and/or metadata exchanged during the translation process.

5.2 Translation of Intent Policies

5.2.1 Introduction

Intent Policy is first translated to an intermediate form (a.k.a intermediate representation), which is typically a set of data structures (and possibly code) that is used to further analyse and transform the original entry. For example, human-readable text could be transformed into a graph. This type of processing is performed by the Intent Translation Functional Block, as shown in Figure 5-3. The translation process possibly will use more than one intermediate forms (e.g. if more than one intent abstraction level needs to be processed).

The second step is to further analyse the Intent Policy, both syntactically and semantically. At this stage, the Intent Policy is modified to correct errors, remove ambiguities, and transformed into a form that is more conducive for changing the intermediate form into recommendations and/or commands that the Policy Intent Target is able to understand. At this point, the Intent Policy is ready to be executed. The data processing may then continue recursively with Functional Blocks interacting with each other.

In the existing operation of a network, Intent Policy translation is done by experienced operational staff. The use of this knowledge in an ENI system for Intent Policy translation is for further study.

5.2.2 Intent Policy Translation Functional Block Architecture

The Intent Translation Functional Block is a part of the Policy Management Functional Block, and is responsible for translating and transforming the original Intent Policy submitted to a set of recommendations and/or commands that can be applied to the Intent Policy Target. Figure 5-3 shows the Functional Blocks that make up the Intent Translation Functional Block, as shown inside the dotted light blue rectangle. The outer green rectangle shows the Policy Management Function Block (which contains the Intent Translation Functional Block). The outer dashed rectangle denotes the ENI System.

The Intent Translation Functional Block includes the Intent Parser, Syntax Analyser, Semantic Analyser, Local Conflict Resolution, Intent Abstraction Translator and Intent Policy Compiler. The Policy Decision Engine, Policy Execution Engine, and Policy Verification Engine are embedded Functional Blocks that perform runtime processing of ENI Policies within the Policy Management Functional Block. The following parts of this clause will introduce the roles and tasks allocated to each Functional Block, and then a detailed interaction of flows with other ENI Functional Blocks will be introduced in clause 5.2.3.

Figure 5-3 shows the involved Functional Blocks to translate Intent Policies. The Functional Blocks interactions inside the black-dotted box illustrate the process to translate Intent Policies inside the Policy Management Function Block. The blue box illustrates the import Functional Blocks that are required for the processing of Intent Policies, which consists of the nested Intent Translation Functional Block.

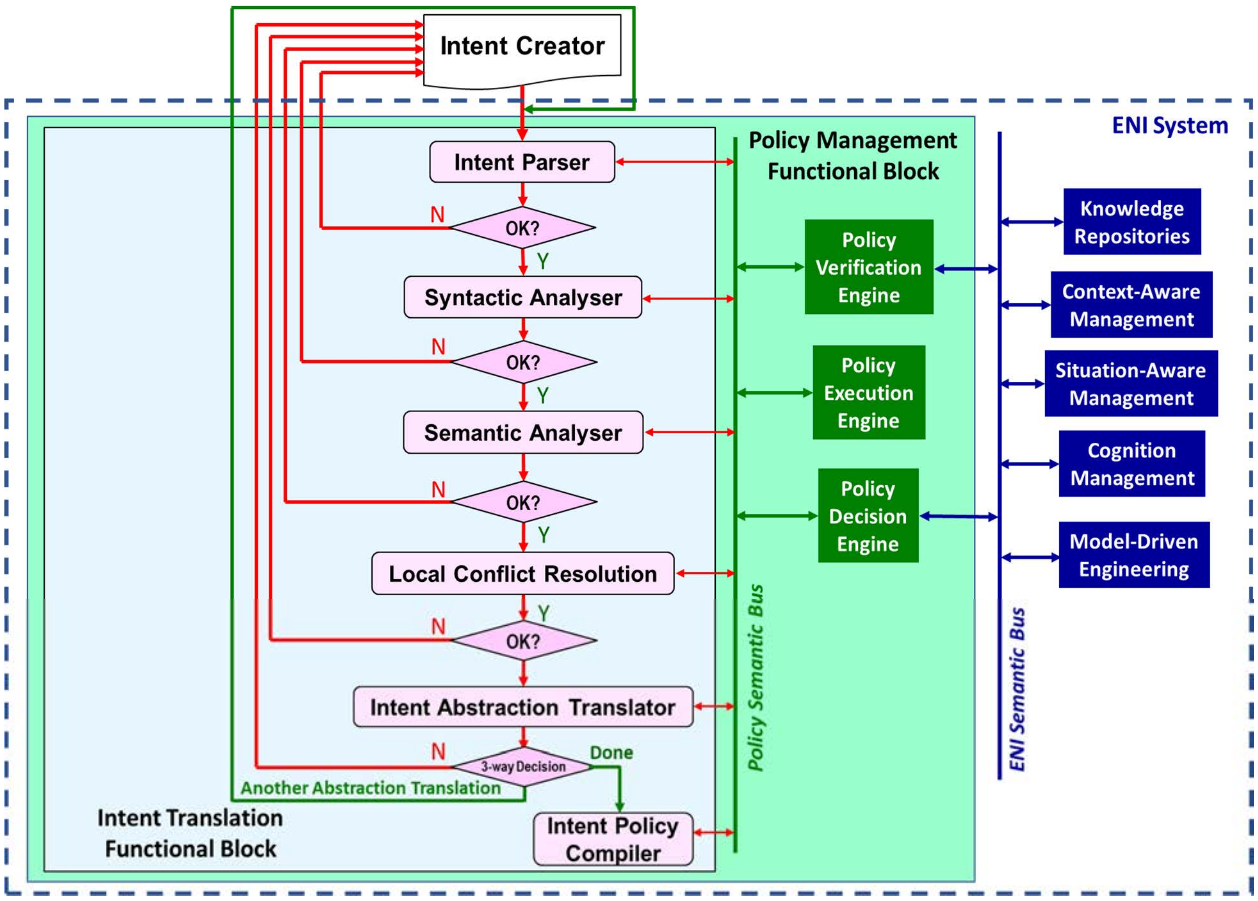

Figure 5-3: Translation of Intent Policies

Intent Creator: This is a set of authorized entities including the User/APP/OSS/BSS and Orchestrator external entities that is able to create an Intent Policy. The Intent Creator submits the Intent Policy to the ENI System through dedicated External Reference Points.

Intent Parser: The Intent Parser is responsible for the initial parsing of the submitted Intent Policy. This typically consists of lexical analysis, token generation, and syntactic analysis (though other functions can also be included). Lexical analysis is the first phase, and takes an input (pre-processed to be a sentence), compares it to a grammar, and then breaks the sentence into sets of characters. The next phase generates tokens from the set of characters that are defined using the grammar rules of the Intent Policy language (e.g. the identification of nouns and verbs). The final phase is syntactical analysis, which ensures that the analysed sentence is grammatically correct. For example, the sentence "Send Host Computer to the Notifications" contains legal tokens, but is grammatically incorrect (it should be "Send Notifications to the Host Computer"). The output of the Syntactical Analyser is either a Concrete or an Abstract Syntax Tree. If any error happens during the parsing process, appropriate error messages are sent to the Intent Creator, and the Knowledge Management Functional Block records the errors for further analysis.

Semantic Analyser: The purpose of the Semantic Analyser is to provide meaning to the output of the Syntax Analyser. Examples include datatype checking, array bounds checking, proper declaration of variables, and scope resolution. For example, the expression "int x = "aValue" will pass lexical and syntactic analysis, as it is structurally correct. However, it will generate a semantic analysis error, since the datatype of the variable does not match the datatype of the value. The Semantic Analyser in ENI generates gists and keywords to provide additional knowledge to other Functional Blocks. If any error happens during this process, error messages are sent to the Intent Creator, and the Knowledge Management Functional Block records the errors for further analysis.

Local Conflict Resolution: This module interacts with the Knowledge Repository and checks if the current Intent Policy conflicts with any existing Policies. For the purposes of the present document, a *policy conflict* is defined as two policies that, when executed, cause contradictory and otherwise incompatible results within a given policy execution time window. For example, if Intent Policy 1 sets the value of an attribute named numErrors to "2" at 08:00:00, and intent Policy 2 then sets the value of the numErrors attribute to 3 at 08:00:01, that is a policy conflict. If any conflict is detected during this process, conflict messages and error information are sent to the Intent Creator, and Knowledge Management Functional Block records the errors for further analysis.

NOTE 1: While the above definition of a policy conflict applies to imperative policies, there are additional problems for intent policies. This is for further study.

Intent Abstraction Translator: The Intent Abstraction Translator retrieves knowledge from the Knowledge Management FB to identify the abstraction level of the input Intent Policy. It then defines the target output abstraction level, as well as the types of output configuration parameters, based on the semantics of the Intent Policy, the gist and keywords, and the input abstraction level. Other information, including Intent Policy metadata, could also be used to help to identify the abstraction levels. The abstraction level refers to the different views of Policy Continuum [i.2]. For instance, an end user authors an Intent Policy to improve video streaming quality without supplying any technical details The Intent Abstraction Translator identifies the abstraction level of the input Intent Policy as the business view level. Similarly, in order to be enforced, the target output abstraction level needs to be at the Instance view level. It is up to the Intent Translation Functional Block to decide if one or more translations are required between the input and output abstraction levels. Once the current abstraction level has successfully completed, this module then determines if another abstraction level translation is required, or if it is finished. If the former, control returns to the Intent parser with new instructions; if the latter, the completed Intent Policy is sent to the Intent Policy Compiler.

NOTE 2: An alternative would be for the Intent Translation Functional Block to simply map this intent to a known SLA, such as Gold or Platinum Service.

Intent Policy Compiler: The Intent Policy Complier compiles the finalized Intent Policy output into a form that can be processed by the Model Driven Engineering Functional Block.

All other Functional Blocks are defined in ETSI GS ENI 005 [i.2] where the policy continuum is specified.

5.2.3 Procedure for Translating DSL Intent Policies

5.2.3.1 Introduction

This clause describes the procedure for intent policy translation when the intent policy is expressed by a DSL, and it is based on the architecture described in clause 5.1.

5.2.3.2 Translation Overview

Figure 5-4 is a simplified message sequence diagram that illustrates the steps between the visible actors (i.e. the entity that authors the policy, the ENI System Functional Blocks, and the Assisted System, thus excluding the ENI System Internal Functional Blocks as well as the ENI System Internal Interfaces). These steps help to describe the Intent Translation process. Figure 5-5, depicted further down, consists of a more detailed message flow sequence diagram where interaction amongst all involved ENI Functional Blocks is shown.

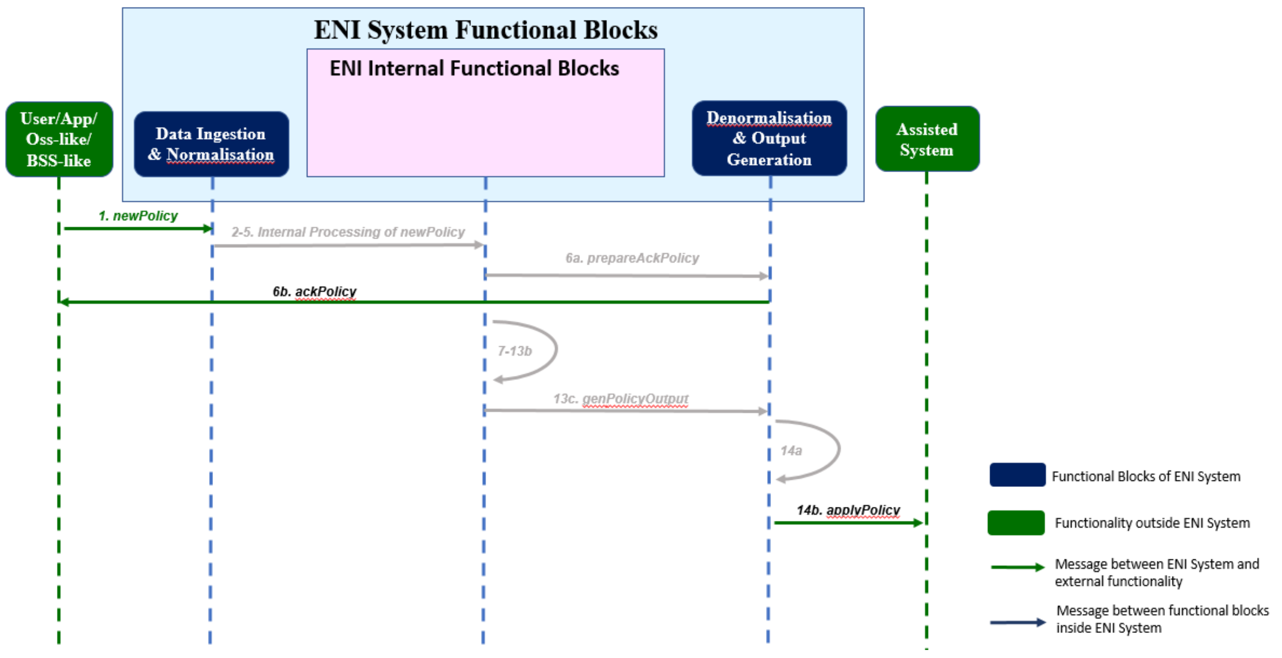
ENI System Functional Blocks
ENI Internal Functional Blocks
User/App/ Oss-like/ BSS-like
Data Ingestion & Normalisation
Denormalisation & Output Generation
Assisted System
1. newPolicy
2-5. Internal Processing of newPolicy
6a. prepareAckPolicy
6b. ackPolicy
7-13b
13c. genPolicyOutput
14a
14b. applyPolicy
Functional Blocks of ENI System
Functionality outside ENI System
Message between ENI System and external functionality
Message between functional blocks inside ENI System

Figure 5-4: Simplified Message Sequence Diagram of the Intent Policy Translation Process excluding Internal Interfaces

The steps are shown from left-to-right, and proceed top-to-bottom, in Figure 5-4. The related activities performed in each task are the same as those performed by the corresponding Functional Blocks of Figure 5-5. However, the text in this clause is abbreviated, as it is an overview. These steps **assume that the target of the Intent Policy is the Assisted System (if the target is the ENI System, then the flow is different)**, and are described as follows:

Step 1. The Intent Creator sends a "**1. newPolicy**" message to the Data Ingestion and Normalisation FB **using an appropriate External Reference Point**. This message consists of three parameters, called policyRefPoint, policyContent and policyMetadata. The policyRefPoint parameter records the specific External Reference Point through which this Policy was received. The policyContent parameter defines the policy, and the policyMetadata parameter contains optional descriptive and/or prescriptive metadata.

Steps 2-5. The input policy is normalised, enhanced, and given a PolicyID by the Internal Functional Block of the ENI System. These steps are neither seen by the entity that authored the Intent Policy nor by the Assisted System.

Step 6a. A response is prepared to the Intent Creator, and sent to the Denormalisation and Output Generation FB. This step is neither seen by the entity that authored the Intent Policy nor by the Assisted System.

Step 6b. An acknowledgement of the Intent Policy is sent to the entity that authored the Intent Policy (shown as "**6b. ackPolicy**") **by using an appropriate External Reference Point**.

Step 7-13b. The Intent Policy is compiled and augmented with semantic, context, and situation aware information. The terms in the Intent Policy are then replaced with appropriate information that can be understood by the entities affected by this Policy, and transformed to an output Policy of the desired type. These steps are neither seen by the entity that authored the Intent Policy nor by the Assisted System.

Step 13c. The final Intent Policy is sent in a "**genPolicyOutput**" message **via the Semantic Bus** to the Denormalisation and Output Generation FB. This step is neither seen by the entity that authored the Intent Policy nor by the Assisted System.

Step 14a. The Denormalisation and Output Generation FB denormalises and reformats the Intent Policy (if necessary).

Step 14b. The Denormalisation and Output Generation FB sends the Intent Policy to the appropriate target entities in the Assisted System (shown as "**14b. applyPolicy**") **via an appropriate External Reference Point.**

5.2.3.3 Translation Detailed Description

5.2.3.3.1 Interactions of Intent Policy via External Reference Points

Intent Translation FB is a new functional block, which takes part in the process of Intent Translation and is a part of the Policy Management Functional Block (see clause 5.1.1).

Table 5-1 provides brief descriptions of how an Intent Policy interacts with the External Reference Points of an ENI System. For each External Entity (e.g. User, BSS, OSS, APP or Orchestrator) the information and metadata that can be sent to or received from the ENI system during the processing of the Intent Policy is described using the appropriate External Reference Points (which are defined in ETSI GS ENI 005 [i.2]).

Table 5-1: Information and Data Exchanged for an Intent Policy via an External Reference Point

Name	Publishes to External Reference Point	Receives from External Reference Point
OSS-like Functionality	Intent Policies and associated information and/or metadata sent from the OSS-like Functionality to the ENI System that control behaviour of the Intent Policy Target using $E_{oss\text{-}eni\text{-}pol}$. This can also include notifications and acknowledgements sent from the OSS-like Functionality.	Information, policies, and metadata sent by the ENI System when processing Intent Policy. This can also include notifications and acknowledgements sent to the OSS-like Functionality.
Application	Intent Policies and associated information and/or metadata sent from the Application to the ENI System that control behaviour of the Intent Policy Target using $E_{app\text{-}eni\text{-}pol}$. This can also include notifications and acknowledgements sent from the Application.	Information, policies, and metadata sent by the ENI System when processing Intent Policy. This can also include notifications and acknowledgements sent to the Application.
BSS-like Functionality	Intent Policies and associated information and/or metadata sent from the BSS-like Functionality to the ENI System that control behaviour of the Intent Policy Target using $E_{bss\text{-}eni\text{-}pol}$. This can also include notifications and acknowledgements sent from the BSS-like Functionality.	Information, policies, and metadata sent by the ENI System when processing Intent Policy. This can also include notifications and acknowledgements sent to the BSS-like Functionality.
User	Intent Policies and associated information and/or metadata sent from the User to the ENI System that control behaviour of the Intent Policy Target using $E_{usr\text{-}eni\text{-}pol}$. This can also include notifications and acknowledgements sent from the User.	Information, policies, and metadata sent acknowledged or notified by the ENI System when processing Intent Policy. This can also include notifications and acknowledgements sent to the User.
Orchestrator	Intent Policies and associated information and/or metadata sent from the Orchestrator to the ENI System that control behaviour of the Intent Policy Target using $E_{or\text{-}eni\text{-}pol}$. This can also include notifications and acknowledgements sent from the Orchestrator.	Information, policies, and metadata sent acknowledged or notified by the ENI System when processing Intent Policy. This can also include notifications and acknowledgements sent to the Orchestrator.

5.2.3.3.2 Flux Diagram of the Intent Policy Translation Process and steps between actors

Figure 5-5 illustrates the steps between all actors (i.e. the entity that authors the policy, the ENI System Functional Blocks, and the Assisted System). These steps are part of a detailed message sequence diagram, depicted below, that helps to describe the Intent Translation process.

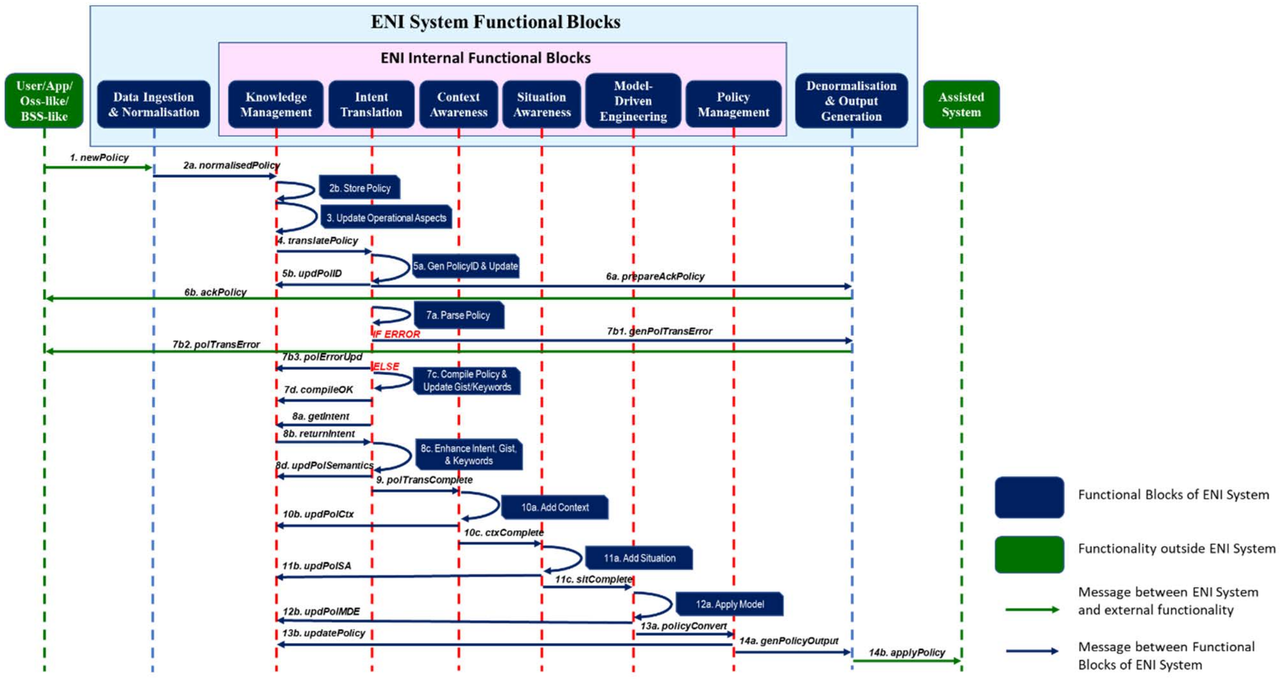

Figure 5-5: Detailed Message Sequence Diagram of the Intent Policy Translation Process including Internal Interfaces

The steps are shown from left-to-right, and proceed top-to-bottom, in Figure 5-5. They are described as follows:

Step 1 Submitting a New Policy

The Intent Creator sends a "**1. newPolicy**" message to the Data Ingestion and Normalisation FB **using an appropriate External Reference Point** (i.e. $E_{oss-eni-pol}$, $E_{bss-eni-pol}$, $E_{app-eni-pol}$, $E_{or-eni-pol}$, or $E_{usr-eni-pol}$). This message consists of three parameters, called policyRefPoint, policyContent and policyMetadata. The policyRefPoint parameter records the specific External Reference Point through which this Policy was received, along with other related information, such as time and date. The policyContent parameter defines the content of the Intent Policy submitted by the Policy Intent Creator, and the policyMetadata parameter contains optional descriptive and/or prescriptive metadata. Examples of descriptive metadata include a description of best current practices and additional purpose information or keywords; examples of prescriptive metadata include a time period that this intent policy is valid, as well as version information, including a minimum version that can be used.

Step 2 Normalising the Intent Policy Input

a) The Data Ingestion and Normalisation FB sends the normalised information in "2a. normalisedPolicy" message to the Knowledge Management FB via the Semantic Bus (see Figure 6-3 in ETSI GS ENI 005 [i.2]).

b) The Knowledge Management FB stores the normalised information in its Policy Repository (shown as "2b").

Step 3 Knowledge Management Internal Process (all 4 steps below are shown as a single step **"3"** for simplicity)

a) The Knowledge Management FB then examines the **policyRefPoint data** of the **normalisedPolicy** message to determine which external entity has sent this Policy.

b) Once this is determined, it consults a list of agreements between the ENI System and that external entity describing the operational procedures for processing Policies (e.g. if negotiation is allowed or required for certain parameters).

c) It then updates the received Policy instance with this information.

d) The Knowledge Management FB also updates the Policy Repository.

Step 4 Begin Translation of the Intent Policy

If the received Policy is an Intent Policy, then the Knowledge Management FB sends a "**4. translatePolicy**" message to the Intent Translation FB **via the Semantic Bus**. Otherwise, the Knowledge Management FB follows the procedures for a non-Intent Policy.

NOTE: Procedures for I policies that are not Intent Policy are outside the scope of the present document.

Step 5 Intent Translation Internal Process

a) The Intent Translation FB generates a policy identifier (policyID) for the received Intent Policy (shown as message "**5a**").

b) The Intent Translation FB updates the Policy Repository by sending a "**5b. updPolID**" message **via the Semantic Bus**.

Step 6 Inform the Entity that Authored the Intent Policy

a) The Intent Translation FB then sends a "**6a. prepareAckPolicy**" message to the Denormalisation and Output Generation FB **via the Semantic Bus**.

b) The Denormalisation and Output Generation FB then sends a **"6b. ackPolicy"** message to the Intent Creator via an appropriate External Reference Point; this informs that entity that its Policy has been received and the ENI System is starting to process it.

Step 7 Intent Translation FB Attempts to Parse the Intent Policy

a) A parser lexical analyser processes the input Policy and attempts to generate tokens. The tokens are then processed by a syntactic analyser. This process may build an abstract syntax tree (or an equivalent data structure). This is shown as "**7a**".

b) **IF ERROR:**

  i) If at any time during the overall parsing and syntactic analysis processes an error is produced the Intent Translation FB creates a **"policyTranslationError"** data structure containing at least three parameters (the **type of error**, any keywords that have been generated, and as much **context and other relevant information** as possible (e.g. line number and word(s) that are not recognized or invalid in this context). A "**genPolTransError**" message is sent to the Denormalisation and Output Generation FB (shown as "**7b1**") **via the Semantic Bus**.

  ii) A "**polTransError**" message is sent to the entity that authored the Policy (shown as "**7b2**") **via an appropriate External Reference Point**.

  iii) A "**polErrorUpd**" message is sent to the Policy Repository in the Knowledge Management FB (shown as "**7b3**") **via the Semantic Bus**. This enables the information to be used for further grammar analysis and adjustment. The translation process is then terminated.

c) **ELSE** (no error found, compile is considered as successful):

  i) The syntax tree is then processed by a semantic analyser. The gist and/or keywords are updated (shown as "**7c**").

  ii) The compiled policy, along with its gist and/or keywords, is stored in the Knowledge Management FB after sending a "**7d. compileOK**" message **via the Semantic Bus**.

Step 8 Semantically Augment the Intent Policy

a) The Intent Translation FB retrieves applicable intent knowledge from the Knowledge Management FB in a "**getIntent**" message sent to the Knowledge Management FB **via the Semantic Bus**. Note that this phase of the process could imply multiple exchanges (e.g. a retrieved datum then requires additional data to be retrieved; this is **not** shown in the figure to keep it simple).

b) All requested information is sent to the Intent Translation FB in a "**returnIntent**" message sent **via the Semantic Bus**. Again, note that this could imply multiple exchanges as explained in the last step.

c) The Intent Translation FB combines the retrieved information with the syntax tree and, using semantic analysis, derives additional information about this intent policy (shown as "**8c**"). Additional information may include information that further clarifies the intent objective, the names of network status parameters and identifiers of network elements related to the intent policy, etc. A gist and/or additional keywords is then generated.

d) An "**updPolSemantics**" message is sent **via the Semantic Bus** to the Knowledge Management FB to update it with this information and knowledge.

Step 9 Successful Translation of the Intent Policy is Sent

At this point, the Intent Policy has been translated from its raw form to a new (internal to ENI) form that enables further processing. The Intent Translation FB sends a "**polTransComplete**" message to the Context Awareness FB **via the Semantic Bus**.

Step 10 Addition of Context Awareness Information to the Intent Policy

a) The translated Policy is then examined by the Context Management FB, which adds any additional relevant context information to the Policy (shown as "**10a**").

b) The Knowledge Management FB is updated with this information by a "**10b. updPolCtx**" message sent **via the Semantic Bus**.

c) The Context Awareness FB sends a "**ctxComplete**" message to the Situation Awareness FB **via the Semantic Bus**.

Step 11 Addition of Situation Awareness Information to the Intent Policy

a) The translated Policy is then examined by the Situation Awareness FB, which adds any relevant information to the Policy (shown as "**11a**").

b) The Knowledge Management FB is updated with this information by a "**11b. updPolSA**" message sent **via the Semantic Bus**.

c) The Situation Awareness FB sends a "**sitComplete**" message to the Model-Driven Engineering FB **via the Semantic Bus**.

Step 12 Generation of the Intent Policy

a) The translated Policy is then sent to the Model-Driven Engineering FB, which transforms the Policy by replacing terms used in the Intent Policy with appropriate more information that can be understood by the entities affected by this Policy; this also transforms this Intent Policy to a form ready for the Policy Management FB (shown as "**12a**").

b) The Knowledge Management FB is updated with this information by a "**12b. updPolMDE**" message sent **via the Semantic Bus**.

c) The Model-Driven Engineering FB sends a "**mdeComplete**" message **via the Semantic Bus** to the Policy Management FB.

Step 13 Format the Intent Policy

a) The Policy Management FB takes the transformed Intent Policy and converts it into a Policy of the desired type (shown as "**13a**").

b) The Knowledge Management FB is updated with this information by a "**updPolFinal**" message sent by the Policy Management FB **via the Semantic Bus**.

c) The final Intent Policy is sent, by the Policy Management FB, to the Denormalisation and Output Generation FB in a "**genPolicyOutput**" message **via the Semantic Bus**.

Step 14 Send the Intent Policy to the Assisted System

a) The Denormalisation and Output Generation FB denormalises and reformats the Intent Policy (if necessary; this is shown as "**14a**").

b) The Denormalisation and Output Generation FB sends a "**14b. applyPolicy**" message **via an appropriate External Reference Point** containing the Intent Policy to the appropriate target entities in the Assisted System.

5.3 Lifecycle Management of Intent Policy

5.3.1 General

An Intent Policy has its own distinct lifecycle. It is created, translated and deployed. The Intent Policy may be changed by the Intent Creator, and as being enabled, disabled, or removed. There are two types of major events that occur during the lifecycle of an Intent Policy:

1) those performed externally (e.g. by the Intent Creator or the Assisted System);

2) those performed by the ENI System.

5.3.2 State of Intent Policy

It is beneficial to define a set of Intent Policy states to better exhibit different stages of the Intent Policy lifecycle and to associate the actions of the ENI System with each stage. External and internal lifecycle management operations may trigger transitions between Intent Policy states.

Defining the state machine for the lifecycle of the Intent Policy can help the ENI System take the appropriate action for each state:

- **Design:** the status of the design of this Intent Policy. States include {not_started, in_process; complete, in_modification}.
- **Deployment:** the status of the deployment of this Intent Policy. States include {not_deployed, ready_to_be_deployed; deployed}.
- **Execution:** the status of the execution of this Intent Policy. States include {not_started, in_process; executed_without_errors, execution_aborted, execution_failed, execution_conflict_and_rollback, exeution_conflict_and_error, execution_timeout}.
- **Admin:** the administrative status of this Intent Policy. States include {enabled, disabled, in_test}.

NOTE: Conflicts with other type of policies are not addressed in this release.

5.3.3 Operations of State Management

The lifecycle management operations from the Intent Creator and the Assisted System (or its Designated Entity) will include:

- **Create Intent Policy:** a new Intent Policy is sent to the ENI System.
- **Enable Intent Policy:** the ENI System is requested to enable an Intent Policy.
- **Disable Intent Policy:** the ENI System is requested to disable an Intent Policy.
- **Execute Intent Policy:** the ENI System is requested to execute an existing Intent Policy.
- **Update Intent Policy:** the ENI System is requested to change the content of an existing Intent Policy.
- **Delete Intent Policy:** the ENI System is requested to remove an Intent Policy.

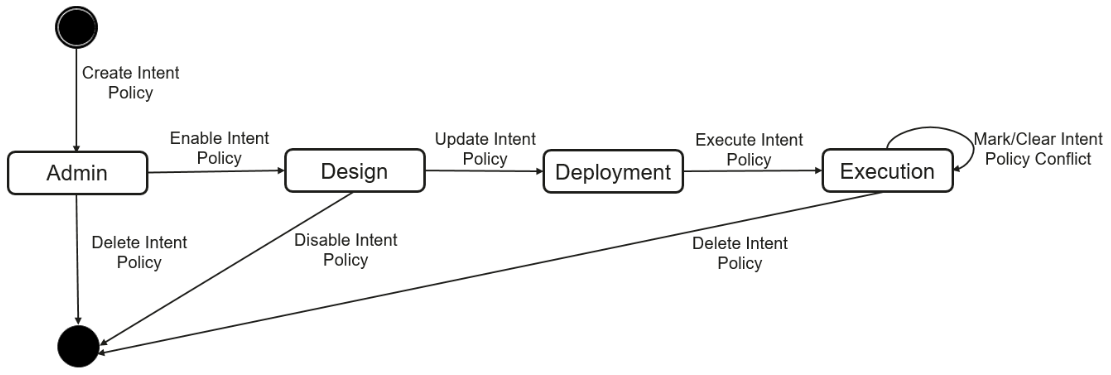

Figure 5-6: Intent policy state management the lifecycle management operations from the ENI System may include

- **Enable Intent Policy:** the ENI System enables an Intent Policy.
- **Disable Intent Policy:** the ENI System disables an Intent Policy (e.g. when the policy fails to execute properly).
- **Update Intent Policy:** the ENI System changes the content of an existing Intent Policy (including adding metadata).
- **Execute Intent Policy:** the ENI System executes an existing Intent Policy.
- **Mark Intent Policy Conflict:** the ENI System marks an Intent Policy as conflicting with another Intent Policy.
- **Clear Intent Policy Conflict:** the ENI System removes the conflicting metadata for an Intent Policy.
- **Delete Intent Policy: the ENI System is requested to remove an Intent Policy.**

5.4 Absorb environment and vendor difference for intent-enabled autonomous system

Challenges exist when implementing an intent-enabled system for a multi-vendor environment or migrating a system to a different vendor environment. Indeed, the difference in environments may lead to re-analysis and redesign of intent-enabled system, thus resulting in an increase in the complexity software development. There are 2 main challenges to apply an intent-enabled system to different vendor environments:

1) for the intent-enabled system, how to collect the needed environment information which are in various format that are produced by different environment to gain context awareness and situational awareness;

2) for the intent-enabled system, how to produce recommendations and/or commands different environments are able to recognize.

NOTE: As an example environment may be considered is an intent-enabled cloud management system where the cloud user uses intent to specify goals concerning cloud application performance and so on, and the intent translation module translates the intent into recommendations and commands. The intent-enabled cloud management system needs to collect various cloud environment information (e.g. log data)from the cloud environment, as well as produces recommendations and/or commands that the cloud environment is able to recognize If the cloud environment differs (e.g. the cloud environment X is implemented on Openstack whereas cloud environment Y is implemented on VMware), it is desired that the intent-enabled cloud management system is able to be applied to both environments, i.e. to collect environment information as well as produce recommendations and/or commands as aforementioned without system modification.

For the first challenge, there are 3 approaches to overcome this challenge:

- **Approach 1.** Standard open API between the intent-enabled system and the environment. The intent-enabled system specifies a standard API with pre-defined data model on the basis of the environment information that are needed, and environment vendors follow the API specification to feed the information to the intent-enabled system. The merit of this approach is that the intent-enabled system is able to utilize the environment information from different environment with the least analysis, development cost, while the shortcoming is that the effect of this approach is highly dependent on the number of vendors applying the open API.
- **Approach 2.** Semantic environment information. In this approach, the environment vendor uses metadata to describe the types, specification, etc. of the environment information, and the intent-enabled system retrieves the environment information together with the metadata from the environment. The intent-enabled system parses these environment information utilizing the types, specification information described by the metadata. The merit of this approach is that it offers more flexibility for environment vendors at the same time improves the intent-enabled system' compatibility to different vendor environments. The shortcoming of this approach is that it may be necessary to implement a parse function to abstract the environment information on the basis of the semantic description.
- **Approach 3.** The combination of the approach 1 and approach 2.

For the second challenge, implementing Model Driven Engineering FB and Output Generation FB is the recommended approach to solve the challenge.

6 Use Cases of Intent Awareness

6.1 Introduction

This clause presents the use cases that demonstrates how to use Intent Policies when improving the operator experience.

In order to demonstrate the flexibility and compatibility of InTent Aware Network Autonomicity (ITANA), besides the use cases defined in ETSI GS ENI 001 [i.1], other use cases related to intent from ISG NFV are shown and that their intent can be fulfilled by ENI System. Moreover, some of the use cases in ETSI GS ENI 001 [i.1] are revisited to present how Intent Policies are used.

6.2 VoLTE Service Experience Optimization

6.2.1 Overview

Clause 5.4.1 of ETSI GS ENI 001 [i.1] presents Use Case#3-1 about context-aware VoLTE service experience optimization.

The VoLTE and Vo5G scenarios are prone to changes in context, which in turn require the wireless networks to be adaptively configured to ensure that SLAs and SLOs are not violated. Though the requirements of voice service remain unchanged (e.g. low drop rate and high availability), the network configuration has to be changed accordingly due to the changing network traffic and radio coverage. The network operator expresses the desired characteristics of the VoLTE and Vo5G service using a set of Intent Policies, and the ENI System will translate the set of Intent Policies and determine the appropriate configuration based on current network performance data.

6.2.2 Motivation

A network operator wants to ensure the average drop rate (e.g. the percentage of RAN nodes with poor drop rates (e.g. above 0,3 %)) of VoLTE and Vo5G services in busy hours in a specified area (e.g. London) are less than 0,5 %. At the same time, the operator also requires a certain level of availability (i.e. ensure that Customers can connect to and use VoLTE and Vo5G greater) than 99,5 %.

6.2.3 Operational communications

Figure 6-1 shows a simplified functional block diagram of how an ENI system can be used to enable an operator to use Intent Policies in a wireless domain.

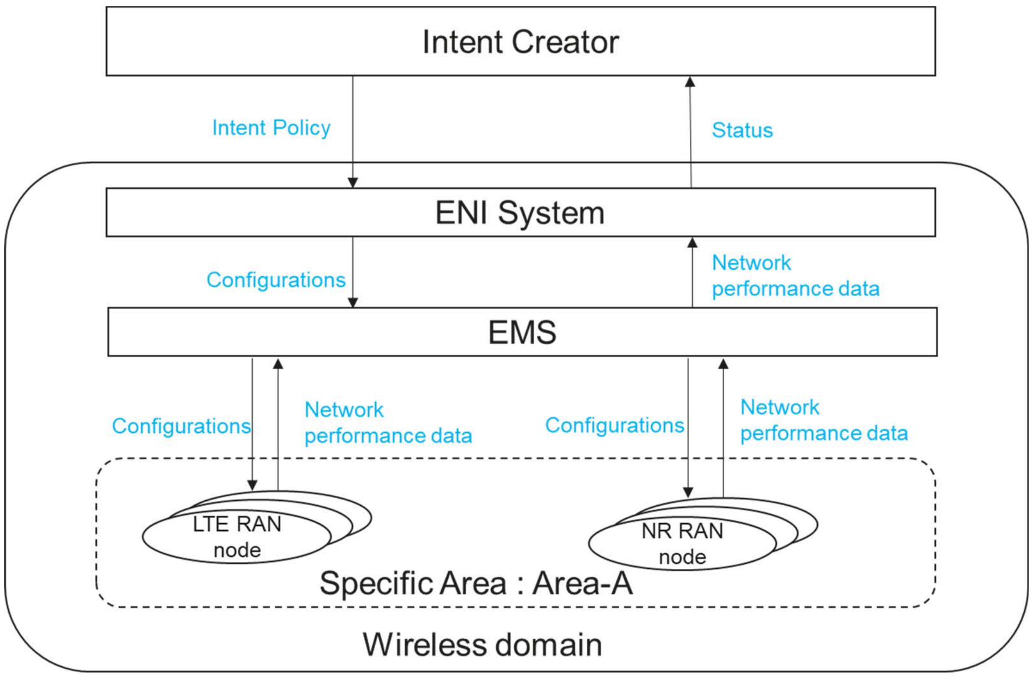

Figure 6-1: An example of VoLTE and Vo5G drop rate and accessibility assurance

The network operator sends the Intent Policy to the ENI System. The Intent Policy expresses the high-level abstract goals of the Operator:

"Ensure that Premium Voice service is always available in the London vicinity during working hours".

An ENI System will perform a set of translations to transform the above intent into a form that can be used to configure affected devices. For example, the above intent is transformed to: "In Area-A and during busy hours, keep the percentage of RAN nodes with drop rate of VoLTE and Vo5G between 0 % and 0,5 %, and the availability of VoLTE and Vo5G services larger than 99,5 %".The ENI System will validate the received intent policy, and then translate it into the operational flow description that is understandable to the ENI System for this particular wireless network domain (e.g. clause 5.4.1.2.6 in ETSI GS ENI 001 [i.1]).

After that, the ENI System tells the EMS to collect VoLTE and Vo5G service information from LTE RAN nodes and New Radio (NR) RAN nodes located in Area-A using the External Reference Point e.g. $E_{oss\text{-}eni\text{-}dat}$.

The ENI System then determines the required configuration according to the context of the current network performance data (e.g. changing reference signal power or policy for handover), and then sends the set of recommended commands to the EMS through e.g. $E_{oss\text{-}eni\text{-}cmd}$. The EMS then sends the set of commands to the LTE and NR systems to configure the RAN nodes.

An Intent Policy once invoked is in effect until it is revoked. The Intent policy will obtain network performance data to determine if it is being satisfied, and report the status to the Operator through an appropriate interface e.g. $E_{oss\text{-}eni\text{-}cmd}$. The status (satisfied or not) is then forwarded be to the operator. The ENI System will give recommendations to the EMS to perform the configuration adjustment to ensure the intent policy is satisfied as needed.

6.3 Use Cases in NFV Domains

6.3.1 Overview

Network Functions Virtualisation (NFV) has been widely used in data centre networks, examples include virtual firewall, virtual load balancer, and virtual Deep Packet Inspection (DPI). Meanwhile, implementation of NFV in telecommunication network has been progressing, examples include virtual IMS (IP Multimedia Subsystem), virtual Evolved Packet Core (EPC) and 5G core. Intent Policy can be used to express NFV Operator's need in a more abstract manner. NFV domain intent may define goals of NFV management and orchestration of NFV service fulfilment phase and NFV service assurance phase.

6.3.2 Use of Intent for NFV Service Fulfilment Tasks

NFV domain intent may define the NFV service fulfilment goals, including but not restricted to functionality-related goals, reliability-related goals and performance-related goals. Intent Translation FB in ENI system is responsible for translating the intent to NFV implementation details including Service Function Chain (SFC) and instance flavour of each instance in the SFC (e.g. the number of vCPUs, memory and disk, etc. to be allocated to instances in the SFC) to fulfil the above goals. For instance, in the case where a NFV operator defines abstract NFV service fulfilment goal using an intent "To realize packet filtering function in L2 and L3, with gold level of reliability, and the filtering function is able to process 10 000 packets per second", the intent is translated into implementation details as follows. The functionality-related intent part "to realize packet filtering function in L2 and L3" is translated into a SFC with a L2 Firewall VNF instance and a L3 firewall VNF instance, thus meet the functionality-related goal; the reliability-related intent part "with gold level of reliability" is translated to a replica of the above SFC to keep the service available even if a fault happens in the original SFC, thus meet the reliability-related goal specified as gold level; the performance-related intent part "the filtering function is able to process 10 000 packets per second" is translated into the flavour of each instance or the number of vCPUs, memory, etc. allocated to each instance in the SFCs thus meet the performance-related goal. The implementation details can be formatted into a NFV service template.

6.3.3 Use of Intent for NFV Service Tasks in order to guarantee SLAs

NFV domain intent may define the NFV service task goals to guarantee the SLAs of NFV service. For instance, a NFV operator defines an intent as "monitor the packet loss of NFV service (UID:xxx), keep the packet loss under 1 %" to guarantee the SLA "packet loss under 1 %". In this case, the Intent Translation FB needs to translate the intent into ser service task details as follows. For the task goal "monitor the packet loss of NFV service (UID:xxx)", the Intent Translation FB translates it into actions including launching the corresponding monitoring tool instance for the given NFV service, registering the NFV service UID and the telemetry information that is related to "packet loss", which needs to be monitored. For the task goal "keep the packet loss under 1 %", the Intent Translation FB translates it into actions based on the results of monitored telemetry. For example, if the packet loss goes over 1 %, the ENI System analyses the cause and takes corresponding actions such as scaling, etc. to restore the service level as specified by the intent. In this example, the Intent Translation FB only translates the NFV intent into a form that the ENI System can understand; all subsequent work is done by other FBs of the ENI System.

6.3.4 Actors and Roles

NFV operator: defines the NFV fulfilment goals and SLA-related NFV service task goals using intent.

ENI System: responsible for translating the fulfilment goals and SLA-related NFV service task goals to NFV implementation and service task details to guarantee SLA.

NFV resource orchestrator/controller: NFV resource orchestrator/controller is responsible for managing the NFV resources, including orchestration, routing, monitoring, alarm handling, etc. Note that at least one external reference point of the NFV resource orchestrator/controller is needed for the ENI system to connect to.

6.3.5 Operational communications

The flows are detailed in the following steps:

1) The NFV operator defines NFV fulfilment goals and SLA-related NFV service task goals using intents, and inputs the intents to the ENI system using $E_{usr\text{-}eni\text{-}pol}$.

2) To meet the fulfilment goal defined by the NFV operator intents, the Intent Translation FB in the ENI System translates the NFV fulfilment-related intent into implementation details as illustrated in clause 6.3.2. The translation details are described in clause 5.2.

3) ENI System instructs the NFV resource orchestrator/controller to create the SFCs in accordance to the generated implementation details in step 2 using $E_{or\text{-}eni\text{-}pol}$. The NFV resource orchestrator/controller informs ENI system using $E_{or\text{-}eni\text{-}dat}$ that the NFV fulfilment-related intent execution is completed.

4) To meet the SLA-related NFV service task goals defined by the NFV operator intent, the Intent Translation FB in the ENI System translates intent into task details as described in clause 6.3.3. The translation details are described in clause 5.2.

5) ENI System instructs the NFV resource orchestrator/controller to launch monitoring tool instances, register telemetry to be monitored to the tool, etc. in accordance to the generated service task details in step 4 through $E_{or\text{-}eni\text{-}pol}$. The collected telemetry/alarm data is passed to ENI system using $E_{or\text{-}eni\text{-}dat}$, and if the SLA is not meet, the ENI system analyses the cause, and gives instructions to solve the problem to NFV resource orchestrator/controller using $E_{or\text{-}eni\text{-}pol}$.

6.4 Intent based energy saving for radio networks

6.4.1 Overview

Energy saving is critical for Self-Organizing Network (SON) and currently discussed in SDOs such as 3GPP RAN3 and 3GPP SA5. For operators, energy saving can effectively reduce OPEX. In one of the scenarios where Base Stations are concerned, the typical behaviour of energy saving is to switch off one or multiple particular cells or to reduce power of base station based on traffic prediction. When traffic increases, the cells may be re-activated via certain X2/Xn signalling/procedure or OAM. Requirement of energy saving may be different considering other aspects such as signalling overhead, latency and coverage. From an OPEX point of view, it is ideal to always minimize the energy consumption while maintaining other network KPIs. Energy saving that considers a good trade-off between energy and other network KPIs is therefore useful for operators for efficient network management.

6.4.2 Motivation

Network Operators desire to reduce energy consumption for the network. The Operator sends an Intent Policy defining a goal energy saving, along with requirements about network KPIs, such as latency and coverage that are important to enforce while saving energy, to the ENI System. The ENI System translates the Intent Policy and analyses it to understand the goals of the Operator.

NOTE: The conflict resolution of Intent Policies will be studied in next release of ITANA.

6.4.3 Actors and Roles

The actors and associated roles that are part of the Use Case are indicated below:

- **Operator:** The operator is responsible for generating Intent Policies and sending them to the ENI System. The Operator is also responsible for approving any changes made to those Policies by ENI, as well as communicating with the ENI System as required.
- **ENI System:** The ENI System is responsible for translating the intent into one or more abstractions that the ENI System can further process and produce recommendations and/or commands to send to the Assisted System or its Designated Entity.
- **OSS:** The OSS may be responsible for providing network status and telemetry information to the ENI System.

6.4.4 Operational communications

A high-level functional architecture of the intent-based energy saving Use Case is described in Figure 6-2.

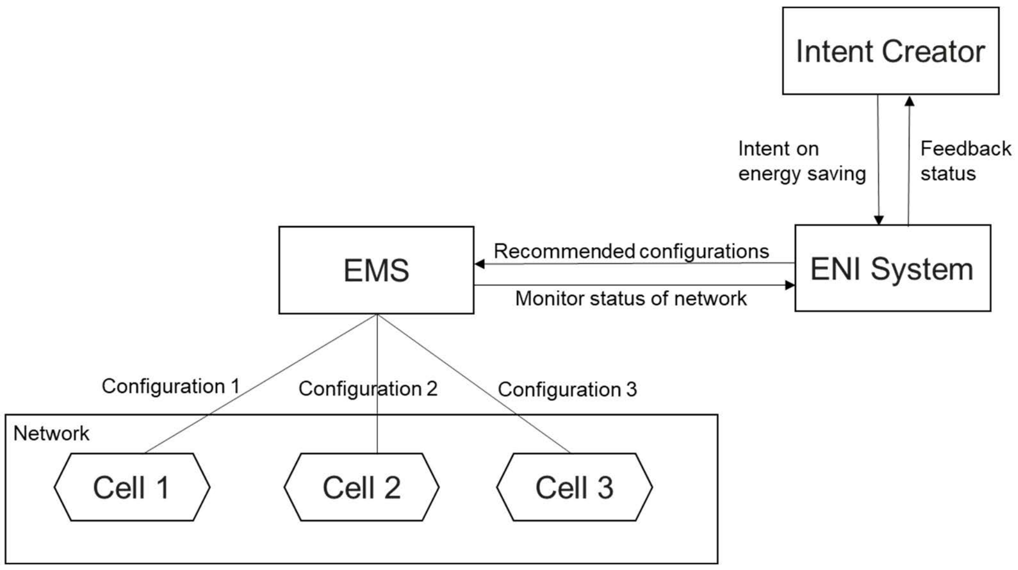

Figure 6-2: Use Case of intent based energy saving

A high-level functional behaviour of intent-based energy saving is described in the following steps:

Step1: Operator sends an energy saving Intent Policy having a goal of, for example, 30 % energy reduction) to the ENI System. Other KPIs (e.g. latency less than 5 ms) that also need to be adhered to when energy is reduced may be sent as well.

Step 2: The ENI system translates the Intent Policy to recommended configurations for different cells according to the different state of cells (e.g. transmission power of cell 1 is high, traffic of cell 2 is high, work time of Cell 3 is long, and cell 4 is Inactive), such as:

- Cell 1 BS transmission power reduction.
- Cell 2 Transfer x % load to other cells.
- Cell 3 Increase sleep time.
- Cell 4 Activate.

During the above procedure, Intent Translation FB may need multiple passes to completely and adequately translate the Intent Policy, and then the rest of the ENI Functional Blocks collaborate to define the recommendations.

Step 3: The OSS determines configurations to be sent to each Cell based on the recommendations and/or commands received from the ENI System for each affected cell of the network

Step 4: ENI system monitors the status of network from OSS and feedbacks the status of the Intent Policy to the Operator. If the Intent Policy is fulfilled, the cell configurations like step 2 are not sent. Otherwise, the cell configurations will be continuously sent until the target is satisfied.

From the point-of-view of the ENI System, it will judge if the desired system goals (i.e. as much of the target energy conservation along with the network KPIs) are able to be fulfilled. If not, the feedback will be reported to the Operator.

NOTE: ENI is a logical system and it may be independently deployed outside of EMS or embedded within EMS.

7 Conclusions and recommendations

Intent Policy is proposed to help express the goals of network services. The use of Intent Policy would simplify operators' handling.

The present document discusses architecture, lifecycle management, and procedures for processing Intent Policy, as well as related use cases.

The study has identified:

- Architecture enhancements to process the Intent Policy, and their connection to the Semantic Bus.
- Procedures related to processing Intent Policy, including an Intent Policy translation procedure that is used to translate the received Intent Policy to the desired commands or recommendations that can be recognized by the Assisted System.
- Lifecycle management of Intent Policy, including states and operations for transition between different states.
- Use cases related to operators. For example, a Context-aware VoLTE Service Experience Optimization is introduced to simplify the management interface by sending a simple Intent Policy instead of several commands.

It is recommended to propose a revision of the present document in the next Release to work on the following aspects:

- Enhanced procedures for processing Intent Policy, e.g.:
  - Conflict detection and resolution between different Intent Policies.
  - Coordination between policies, including Intent Policies from different parts of the infrastructure.
- Knowledge management for Intent Policy, including:
  - How to implement a knowledge repository.
  - Procedures for lifecycle management of intent knowledge, e.g. create, read, update, delete.
- Additional Use cases and requirements that can reduce the management integration complexity, e.g.:
  - Use Cases for operators' business.
  - Use Cases for vertical industries.
  - Use Cases for end-users.

In the future, the content about procedures and knowledge management can be contributed to ETSI GS ENI 005 [i.2], and the use case section can be contributed to ETSI GS ENI 001 [i.1].

Annex A: Change History

Date	Version	Information about changes
2019-06	v0.0.1	Initial early draft with skeleton
2019-07	v0.0.2	Combine ENI(19)010_024r4
2019-07	v0.0.3	CombineENI(19)000215r1, ENI(19)000219
2019-09	v0.0.4	CombineENI(19)000225r2, ENI(19)000223r1, ENI(19)000222r2 &ENI(19)000228
2019-10	v0.0.5	Combine ENI(19)011_035r2

History

Document history		
V2.1.1	March 2021	Publication