ETSI GR ENI 015 V4.1.1 (2024-05)

Experiential Networked Intelligence (ENI); Processing and Management of Intent Policy

Disclaimer

The present document has been produced and approved by the Experiential Networked Intelligence (ENI) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGR/ENI-0025_Int_Policy_proces

Keywords
artificial intelligence, network

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

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 describes the following topics:

- Enhanced procedures for processing Intent Policy, e.g.:
  - detail the Procedures of intent policy processing;
  - conflict detection and resolution between different Intent Policies.
- Knowledge management for Intent Policy, including:
  - how to use a Knowledge Graph to manage Intent policies;
  - how to use a Knowledge Graph for managing Intent policy knowledge;
  - procedures for lifecycle management of intent knowledge, e.g. import, update, delete, and query of the intent knowledge.
- Typical use cases and requirements which can reduce the management complexity for Intent Users, e.g.:
  - use cases for Business Users/Operational Users/Technical Users that are all users of Intent.

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 005 (V3.1.1): "Experiential Networked Intelligence (ENI); System Architecture".

[i.2] ETSI GR ENI 008 (V2.1.1): "Experiential Networked Intelligence (ENI); InTent Aware Network Autonomicity (ITANA)".

[i.3] ETSI GR ENI 016 (V2.1.1): "Experiential Networked Intelligence (ENI); Functional Concepts for Modular System Operation".

[i.4] ETSI GS ENI 030 (V4.1.1): "Experiential Networked Intelligence (ENI); Transformer Architecture for Policy Translation".

[i.5] IEEE Access 8 (2020): "Knowledge graph completion: A review" 192435-192456, Chen Zhe et al.

[i.6] TM Forum IG1253 (V1.2.0): "Intent in Autonomous Networks".

[i.7] MEF 95: "MEF Policy Driven Orchestration", Strassner, J, editor, July 2021.

[i.8] ETSI GS ENI 019 (V3.1.1): "Experiential Networked Intelligence (ENI); Representing, Inferring, and Proving Knowledge in ENI".

[i.9] Stanford definition of "Propositional Logic".

[i.10] ETSI GS ENI 033 (V4.1.1): "Experiential Networked Intelligence (ENI); Definition, Requirements and Procedure of Intent Policy Multi-Stage Translating".

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

bag-of-words model: model which represents text data as a collection of words, regardless of the order of words

NOTE: The bag-of-words model can determine the important words in the text by counting the number of occurrences of each word in the text data.

formal logic: study of propositions, statements, or assertively used sentences and of deductive arguments. Formal logic is the basis of deductive reasoning, which is used to derive a conclusion from a set of premises with certainty

NOTE: Formal logic can also be used to prove the correctness of algorithms by reasoning formally or mathematically about the algorithm.

graph database: database that uses graph structures to store and navigate relationships (i.e. edges) between entities (i.e. nodes) in a graph

information extraction: process of identifying and extracting words and phrases with potential semantic information from text data

knowledge fusion: integration of knowledge from different sources, forms, and structures

knowledge graph: directed cyclic graph that embeds semantics in the nodes and edges using a formal logic (RDF as a minimum; types of Description Logics, such as or an OWL 2 Profile, are preferred)

named entity recognition: identification of entities with specific meanings in data such as text or images, such as person names, organizational structures, place names, etc.

part of speech tagging: process of determining the part of speech of words to better understand the text

quality improvement: process of improving the accuracy, timeliness, completeness, reliability, and robustness of knowledge graph through a series of technical means and algorithms

relationship extraction: process of extracting relationships between entities from raw data

NOTE: Relationship extraction includes character pattern based extraction methods, grammar pattern based extraction methods, and semantic based extraction methods.

state machine: abstract mathematical model used to describe the patterns of transitions and behaviours of an object between various states

topic model: model which is used to identify topics or keywords in text data, such as Latent Semantic Analysis (LSA) based on probabilistic topic model

word embedding: technique used in natural language processing and machine learning to represent words as low dimensional vectors that can be used for tasks such as text classification and sentiment analysis

NOTE: Word embedding can be obtained through neural network learning or manually compiled.

3.2 Symbols

Void.

3.3 Abbreviations

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

ACLs	Access Control Lists
API	Application Programming Interface
BSS	Business Support Systems
CPL	Cloud Private Line
CSRUD	Create-Store-Read-Update-Delete
DL	Description Logic
DSL	Domain Specific Language
GDB	Graph Database
IDS	Intrusion Detection Systems
IPFB	Intent Parser Functional Block
KG	Knowledge Graph
LSA	Latent Semantic Analysis
NAS	Network Attached Storage
NER	Named Entity Recognition
NLP	Natural Language Processing
O&M	Operation and Maintenance
OPEX	Operational Expenditure
OSS	Operations Support System
OTN	Optical Transport Network
OWL	Web Ontology Language
PMFB	Policy Management Functional Block
QoS	Quality of Service
RAN	Radio Access Network
RDF	Resource Description Framework
SPO	Subject-Predication-Object
UHD	Ultra High Definition
URI	Uniform Resource Identifier
VR	Virtual Reality
W3C	World Wide Web Consortium
XR	Extended Reality

4 Background and Overview

4.1 ENI Purpose

Operational Expenditure (OPEX) for network management is one of the operators' biggest concerns. The use of intent in ENI can help operators effectively reduce OPEX. Intent enables different constituencies to express intent policies using a language that is natural to themselves. Hence, the associated business benefit is to understand business needs and use them to determine offered services and resources. This means that offered services can dynamically adapt to changing user needs, business goals, and environment conditions, which will greatly improve the efficiency of network management and reduces the cost of network Operation and Maintenance (O&M).

4.2 Intent Work in ENI

Intent Policy expresses the goals to be accomplished by using a restricted natural language (e.g. an external DSL), without focusing on how to achieve those goals. The definition and introduction about intent policy can be found in clause 6.3.9.3 of ETSI GS ENI 005 [i.1], and a previous study in ETSI GR ENI 008 [i.2] further discussed the architecture of intent policy (in clause 5.1.2), life cycle management (in clause 5.3) and the translation process of intent policy (in clause 5.2.3), as well as related use cases (in clause 6).

4.3 Introduction to Knowledge

Knowledge is defined in clause 6.3.4 of ETSI GS ENI 005 [i.1] and further detailed in clause 6.3.4 of ETSI GS ENI 005 [i.1]. Briefly, knowledge analyses data and information, understands its meaning, and can predict what has happened, is happening, or what is possible to happen in the future. Knowledge management is an essential part in intent aware network, which is responsible for managing the life cycle of the knowledge in the Data and Knowledge Repositories, including storage, assessment, use, sharing, and refinement of knowledge assets. ENI develops knowledge and wisdom from observed, measured, and inferred data and information, as described in ETSI GR ENI 016 [i.3].

5 Procedures of Intent Policy Processing

5.1 Introduction

In ETSI GR ENI 008 [i.2] and ETSI GS ENI 005 [i.1], the procedures of intent policy translation are shown in different detail. In general, the intent policy is created by the intent creator and sent to the ENI System. The ENI System translates and processes the received intent policy and notifies the intent creator of any errors. If there are no errors, then the intent policy is executed on the selected component(s) of the Assisted System.

Considering the complexity of user's goals, the intent policy can be separated into multiple intent policies to simplify the general procedures of intent policy processing, such as translation and named entity recognition. The processed intent policy has three main lifecycle phases (see clause 6.4 of the present document): Translation, Deployment, and Execution.

The benefits of this process are as follows:

1) Ensure the realization of the intent creator's goals by performing two functions automatically:

   a) Translating the intent creator's policy from natural language to a form that can be verified by the ENI System. Subsequent translations could be performed to make the translated policy implementable by the Assisted System. If the translation fails, then errors found will be sent to the intent creator, and processing stops until the intent creator resubmits an edited intent policy.

   b) Testing any new intent policies to ensure that they can be executed correctly. If the testing results are not able to meet the goals defined by the intent creator, then information describing this will be sent to the intent creator, and processing stops until the intent creator resubmits an edited intent policy.

2) Automate the entire process (all processes are triggered automatically, especially automatic testing), while still providing flexibility to the intent creator. For example, the intent creator can have control on each state change of the intent policy lifecycle (see clause 6.4).

3) Support the flexible use of intent policies that are managed by a knowledge process, such as the Knowledge Management Functional Block of ETSI GS ENI 005 [i.1] (see clause 6.3.4 of ETSI GS ENI 005 [i.1]).

NOTE: A group of intent policies could also be managed as a unit (e.g. a high level intent with an associated set of detailed policies on each resource).

5.2 Intent Policy Processing Operation

The original Intent Policy, which is written in a restricted natural language, is sent to the ENI System. It is ingested by the API Broker and sent to the Policy Management Functional Block (PMFB). The PMFB checks to see if the ingested Intent Policy has been seen before (e.g. by matching the text of the newly ingested Intent Policy with the text of a previously stored Intent Policy). If the original Intent Policy matches a previous Intent Policy that has been successfully parsed, verified, and stored, then the ENI System checks to see if anything has changed in the ingested Intent Policy (e.g. parameter values). If nothing has changed, then the PMFB retrieves the previously translated representation of the matched Intent Policy and sends it to the API Broker. The API Broker transforms the translated Intent Policy to a form (i.e. a set of objects and/or code) that the Assisted System can understand. The translated Intent Policy is then sent to the Assisted System. If a change in the ingested Intent Policy is detected, then the PMFB will first verify the proper operation of the Intent Policy by testing it before sending it to the Assisted System. If this test fails, then all errors and warnings are collected and sent back to the Intent Creator. If it passes, then the PMFB sends the verified Intent Policy to the API Broker as described above.

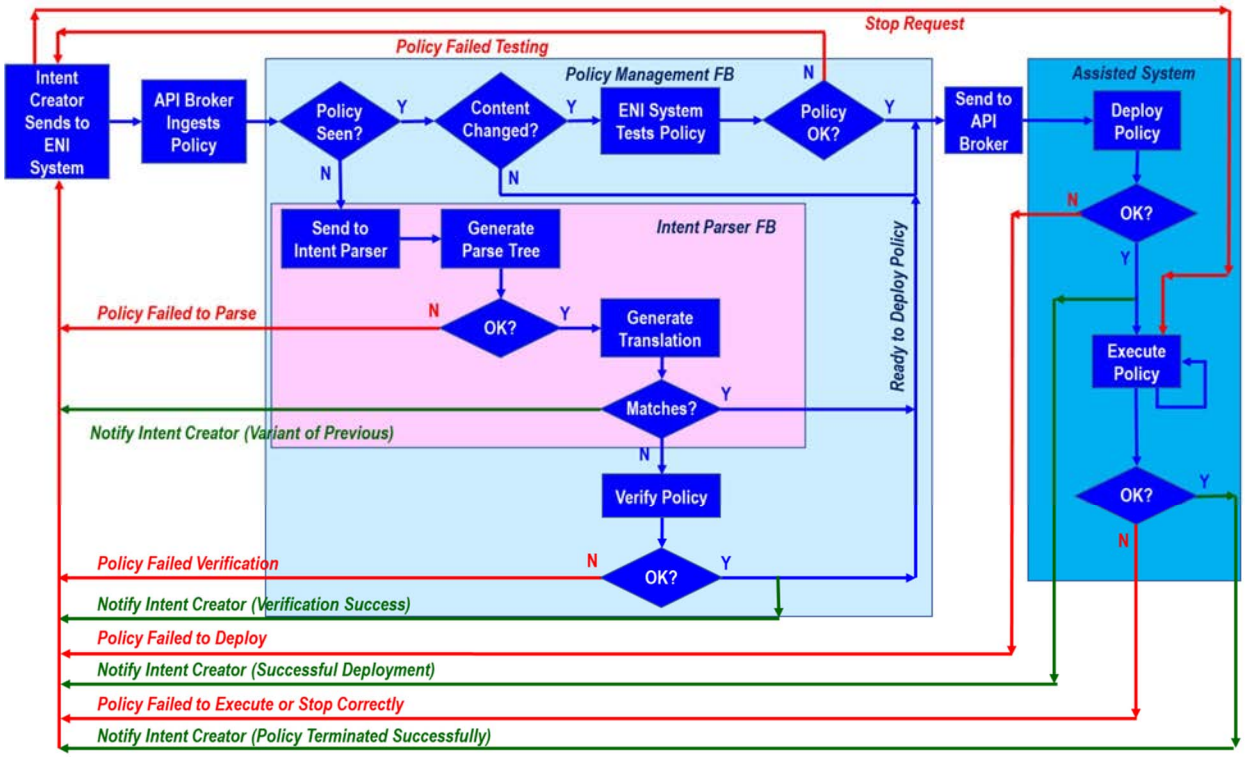

Figure 5.1: Detailed Procedures of Intent Policy Processing

Alternatively, if the original Intent Policy is not found, then it needs to be parsed, optionally compiled and translated, and subsequently verified. This is shown in the rest of Figure 5.1, starting with the "Send to Intent Parser" block. The steps shown in Figure 5.1 are summarized as follows:

1) The input Intent Policy is parsed by the Intent Parser, which is part of the Intent Parser Functional Block (IPFB). The parsing process is described in detail in ETSI GS ENI 005 [i.1], translator architecture in ETSI GS ENI 030 [i.4] and ETSI GR ENI 008 [i.2]:

   a) If the Intent Parser fails to generate a parse tree, then all errors and warnings are collected and sent to the Intent Creator for correction.

   b) Otherwise, the parse tree is generated and sent to the Intent Translation function, which is another Functional Block that resides in the IPFB.

2) The Intent Translation function checks to see if the Intent Policy has been previously used. Conceptually, the Translation function is matching the newly translated object form of the ingested Intent Policy with the object form of the stored Intent Policy, The purpose of this check is to catch grammatical differences that resolve to the same Policy. For example, if Intent Policy 1 says "Give Ray Gold Service" and Intent Policy 2 says "Give Gold Service to the user whose IP Address is 10.10.10.1", and the user of that IP address is Ray, then Intent Policy 1 and Intent Policy 2 are the same. The translation process is specified in ETSI GS ENI 005 [i.1], translator architecture in ETSI GS ENI 030 [i.4] and described as intent translation in ETSI GR ENI 008 [i.2]:

   a) If it has, then the Intent Translation function notifies the Intent Creator and the Intent Policy is ready to be deployed by the Assisted System (step 4).

   b) If not, then the Intent Translation function passes the translated Intent Policy to the PMFB to verify the Intent Policy (step 3).

3) The Intent Policy is verified by testing it (either by simulation or in a closed sandbox). In either case, testing consists of a set of test inputs, execution conditions and expected results prepared for a specific objective, which is used to verify whether the goals of the Intent Policy are met:

   a) If the Intent Policy fails verification, then all errors and warnings are collected and sent to the Intent Creator for correction.

   b) Otherwise, the Intent Policy is ready to be deployed by the Assisted System.

4) The Intent Policy is deployed when either the Intent Creator or the ENI System give the Intent Policy Processing system permission to do so. (The ENI System allows either the Intent Creator to have direct control over the deployment of the Intent Policy, or alternatively, to tell the ENI System to automatically deploy it when ready; this latter is a configurable setting):

   a) If the Intent Policy fails to be deployed, then all errors and warnings are collected and sent to the Intent Creator for correction.

   b) Otherwise, the Intent Policy is ready to be executed.

5) The Intent Policy is executed when either the Intent Creator or the ENI System give the Intent Policy Processing system permission to do so. (The ENI System allows either the Intent Creator to have direct control over the execution of the intent policy, or alternatively, to tell the ENI System to automatically execute it when ready; this latter is a configurable setting):

   a) If the Intent Policy fails to execute, then all errors and warnings are collected and sent to the Intent Creator for correction.

   b) Otherwise, the Intent Policy executes.

The Intent Policy will continue to execute until either the Intent Creator or the ENI System stop its execution.

The above process does not include policy conflict detection and remediation. This is detailed in clause 5.3.

5.3 Conflict detection and resolution

5.3.1 Overview

Conflicts of policies are a well-known challenge to any policy management. The ENI System usually receives intent policies from multiple sources (OSS, BSS, application, end-user, orchestrator, etc.). The ENI System will detect whether there is a conflict between the new Intent Policy and other Intent Policies in Policy Management Functional Block. If the conflict is detected, then the ENI System and Intent Creator try to resolve the conflict. Conflict detection and resolution between different Intent Policies will be discussed in this clause.

NOTE: The present document will only discuss conflict detection and resolution between intent policies. Conflict detection and resolution between intent policies and other types of policies will be done in a future version of the present document.

5.3.2 Conflict detection

When a new intent policy comes, the ENI System needs to consider the possible conflict between it and any existing intent policies that are currently running. If any conflict is detected, the conflict message will be sent to the intent creator, and the knowledge management function block records the conflict information for further reasoning and analysis. The policy conflict is defined in ETSI GS ENI 005 [i.1] as follows: Policy conflict means two policies that, when executed, cause contradictory and otherwise incompatible results within a given execution time window.

From the perspective of detection, there are two levels of conflicts (refer to ETSI GS ENI 005 [i.1] and TM Forum IG1253 [i.6]):

Direct conflict

Direct conflict means that Intent Policies are expressing explicitly opposing or incompatible requirement with each other. The most important thing in this case is to confirm the execution domains and time.

The first example is the following:

EXAMPLE 1: Intent Policy A: All network links are encrypted.
Intent Policy B: All network links are not encrypted.
As written, and with no other information, Intent Policy A and Intent Policy B conflict. However, for example, as long as the user groups do not overlap, there is no explicit contradiction. This can for example be a non-overlapping scope such as requiring encrypted links for one user group and no encryption for another user group, so they can coexist.

Another example:

EXAMPLE 2: Intent Policy C: Set the value of an attribute named numErrors to "2" during 08:00:00 to 08:30:00.
Intent Policy D: Set the value of an attribute named numErrors to "3" during 08:45:00 to 09:00:00.
In this case, Intent Policy C and Intent Policy D do not conflict because their execution times do not overlap, while they will conflict when the execution time overlaps.

Indirect conflict

In many cases, conflicts can not be obviously seen from the Intent Policy itself, but originate from conflicting actions being proposed in their handling process, which is generally caused by common infrastructure and limited resources.

EXAMPLE 3: Intent Policy E: Increase RAN coverage delivered to users.
Intent Policy F: Increase throughput delivered to users.
On the surface, they do not involve the same indicator and there seems no conflict. However, both Intent Policies imply opposing reconfiguration and actions of RAN cells. This is like a seesaw. When the fulfilment of one Intent Policy is improved, the other will be degraded.

EXAMPLE 4: Intent Policy G: Allocate 10 M bandwidth to user A.
Intent Policy H: Allocate 5 M bandwidth to user B.
When there is only 10 M available bandwidth in the network, the two Intent Policy cannot be satisfied at the same time, so conflicts occur.

Based on above, Figure 5.2 shows the procedure of conflict detection, which includes:

1) Translate the new Intent Policy;

2) Detect whether the execution domain of new Intent Policy overlaps with other Intent Policies.

3) If the answer of Step 2 is yes, then detect whether the execution time of new Intent Policy overlaps with other Intent Policies.

4) If the answer of Step 3 is yes, then determine whether execution of the new Intent Policy conflicts with the actions of other active Intent Policies.

NOTE: The detection content in step 4 is not limited to configuration parameters and network resources. It can be multiple aspects that may conflict during the execution process of the Intent Policy.

5) If the answer of Step 4 is yes, the new Intent Policy is judged to conflict with the existing policy, and conflict resolution is needed.

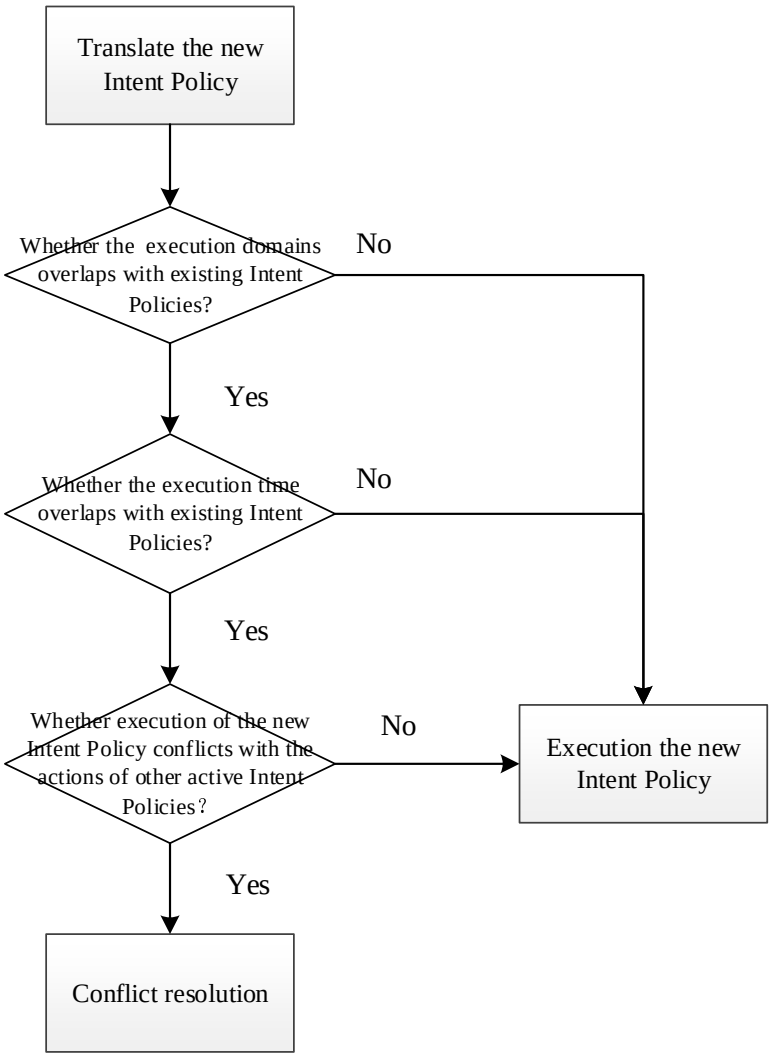

Figure 5.2: The procedure of conflict detection

5.3.3 Conflict resolution

The ENI System needs to resolve conflicting Intent Policies, helping users to properly choose between Intent Policy alternatives that may have different ramifications.

In some conditions, conflict can be easily resolved (for example, the requirement of one Intent Policy includes another). In this case, it is supposed that one Intent Policy requires a maximum delay of 10 ms and another one for the same target requires a maximum delay of 5 ms. As written, the policies conflict, because they apply to all applications, and the system does not know whether to set the maximum delay to 5 ms or 10 ms. There are at least two different solutions to resolving this conflict. The first specifies which applications receive the 5 ms delay and which receive the 10 ms delay. The second is to simply choose the 5 ms delay as long as all applications can support this more stringent requirement.

For more complicated situations, priority is an efficient way to resolve conflicts as described in MEF 95 [i.7]. This indicates that an ENI System should be able to prioritize actions and therefore find the set of operational states that is globally preferential even if it means to degrade some Intent Policies.

Factors affecting priority setting may include:

- Task type. For example, network failure, business configuration, engineering change, security events, etc.
- Impact on network. The tasks that have a great impact on the stability of the network will be processed first, and the tasks that have a small impact on the stability will be processed when they are idle.
- Task deadline. Prioritize tasks with deadlines approaching.

A proper scheduling algorithm should be adopted and used for priority setting and conflict resolution. The specific scheduling algorithm is beyond the scope of the present document.

6 Knowledge management for Intent Policy

6.1 Introduction to Knowledge Graphs

6.1.1 Definition

A Knowledge Graph is a directed cyclic graph that embeds semantics in the nodes and edges using a formal logic (RDF as a minimum; types of Description Logics, such as an OWL 2 Profile, are preferred).

6.1.2 Motivation for Using Knowledge Graphs in ENI

6.1.2.1 Motivation

Knowledge graph technology is a typical cross-field technology. It is integrated with artificial intelligence, mathematics, graphics, database, and other disciplines. Based on graph theory and the probability graph model in [i.5], it can identify, discover and infer the complex relationships between concepts and how they relate to collected data. In terms of using a knowledge graph for intent policy management, it is necessary to transform intent policies constructed using a restricted natural language into a structured knowledge representation, and then select Intent Policies in the knowledge graph to analyse for various management purposes. This process can help an ENI System understand the meaning of the ingested intent policy by using procedures such as named entity recognition and removing any ambiguities found in the translation process. This in turn can help translate abstract intent policies to more concrete intent policies that can be understood by the target(s) of the intent policy. The following provides examples of how a Knowledge Graph can be used to find intent conflicts:

1) Ingested intent policies can be examined in terms of their nodes and relationships to find conflicts. In addition, Metadata could be generated to automatically categorize ingested intent policies according to their nodes and relationships.

2) Knowledge Graphs (KGs) are a way of structuring information in a graph form by representing entities (e.g. customers, services, and places) as nodes, and relationships between those entities (e.g. a customer has a service located at his home) as edges. This inherent capability enables KGs to travel relationships much more efficiently than other technologies, such as directories or relational databases. When a new intent policy is ingested, the KG can verify whether there is a conflict relationship between the new intent policy and existing active intent policies. A KG can help locate and verify the cause of the conflict by using its formal reasoning capabilities. This information can then be supplied to other Functional Blocks in the Policy Management Functional Block to provide detailed information to the ENI System and to the intent creator for further conflict resolution. Conflict detection and resolution capabilities vary, and are dependent on the particular description logic used.

3) An intent policy can be separated into multiple constituent intent policies, and multiple intent policies can be combined into a composite intent policy to realize flexibility and promote reuse. All original intent policies, including any policies that are created by separating a complex intent policy into simpler intent policies, are stored as objects. This enables them to be reused to create new, more powerful, intent policies. The objects can take the form of triples in the Knowledge Graph and ENI Policy Rule objects as defined in ETSI GS ENI 019 [i.8].

NOTE: The decomposition method can be based on the semantics of the intent policy. For example, a single ingested intent policy could be translated into a hierarchical set of intent policies to remove ambiguity; this would also promote reusability, since every intent policy can be saved as an object. It could also be based on behaviour (e.g. action execution order or different operation domains or different target objects).

6.1.2.2 Requirement

The intent policy KG has the characteristics of large scale, complex knowledge structure, diverse sources, high dynamics and timeliness, and deeper reasoning mechanisms. These characteristics give rise to the following requirements for an ENI System:

1) It is expected to be able to use the ENI Information Model, ETSI GS ENI 019 [i.8], to aid in constructing the KG. For example, classes and relationships defined in the ENI Information Model could correspond to named entities and edges in a KG, respectively.

2) An ENI System is expected to provide the ability of processing semi-structured and unstructured data into a structured knowledge representation suitable for expressing a KG. This will likely be done by a combination of the data ingestion and normalization Functional Blocks.

3) An ENI System is expected to provide the ability to construct, store and update KGs in a knowledge repository. This will likely be done by using the Repository Management Functional Block in the Knowledge Management Functional Block.

4) An ENI System is expected to provide intelligent search and match abilities for querying KGs. This will be done in the Intent Policy Processing and Management Functional Block, or equivalent Functional Blocks, which reside in the Policy Management Functional Block.

5) An ENI System is expected to provide knowledge reasoning and knowledge fusion. This will likely be done using a combination of the Knowledge Management and Cognition Management Functional Blocks. It is likely to be supported by other Functional Blocks as well.

6.1.2.3 Location of the Knowledge Graph

It is recommended that the Knowledge Graph processing and management be located in a Functional Block in order to be consistent with the existing ENI System architecture. Figure 6.1 shows two possible deployment options.

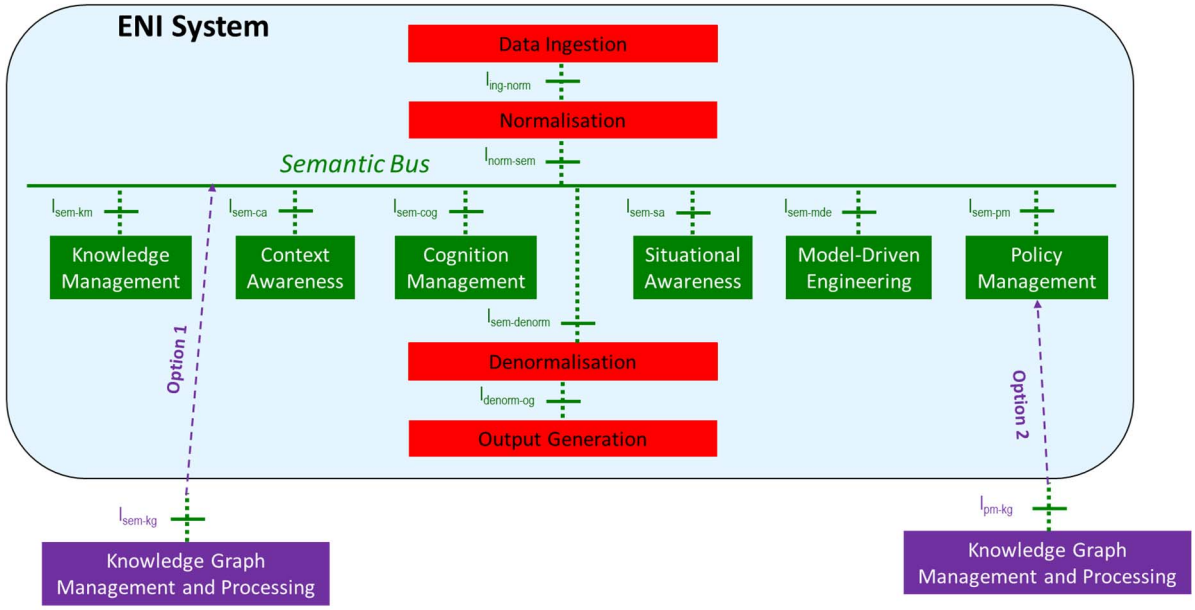

Figure 6.1: Two Deployment Options for a Knowledge Graph Functional Block for ENI

The option on the left of Figure 6.1 creates a new top-level internal Functional Block. This option is best suited if the Knowledge Graph Functional Block will be used generically, and not specifically for ENI Policies. The option on the right of Figure 6.1 creates a new second-level internal Functional Block within the Policy Management Functional Block. This option is best suited if the Knowledge Graph Functional Block will be used specifically for ENI Policies.

Both options enable the Knowledge Graph Functional Block to take advantage of the Semantic Bus, and hence, it can communicate with the other internal Functional Blocks. Both options also prevent direct access of the Knowledge Graph from external consumers (i.e. all communication first goes through the API Broker, then the Data Ingestion and Normalization Functional Blocks, and then put onto the Semantic Bus).

6.2 Constructing Knowledge Graphs in the ENI System

6.2.1 Introduction

A Knowledge Graph is made up of three main components: nodes, edges, and labels. These components provide a mechanism to explicitly define the semantics of an Intent Policy. In addition, a Knowledge Graph has a syntax associated with it, which is the well-formedness of the underlying language (e.g. RDF or OWL) that is used to represent its entities and relations. In addition, a knowledge graph explicitly defines the semantic relationships between the various parts of an Intent Policy. Technologies related to knowledge representation and reasoning, information extraction, natural language processing, data mining and machine learning are involved in the construction of knowledge graph. One of the possible construction processes of a knowledge graph can be summarized as: process data obtained from different sources, determine the knowledge representation model, extract entities and relationships from the processed data, and then generate knowledge, verify knowledge and store knowledge for the use of ENI System.

The construction procedure of Knowledge Graphs in the ENI System includes data processing, knowledge representation, knowledge generation, knowledge verification and knowledge graph storage. The detailed procedures are described in the following clauses, as shown in Figure 6.2.

NOTE: Figure 6.2 is based on the selection of option 1 in Figure 6.1.

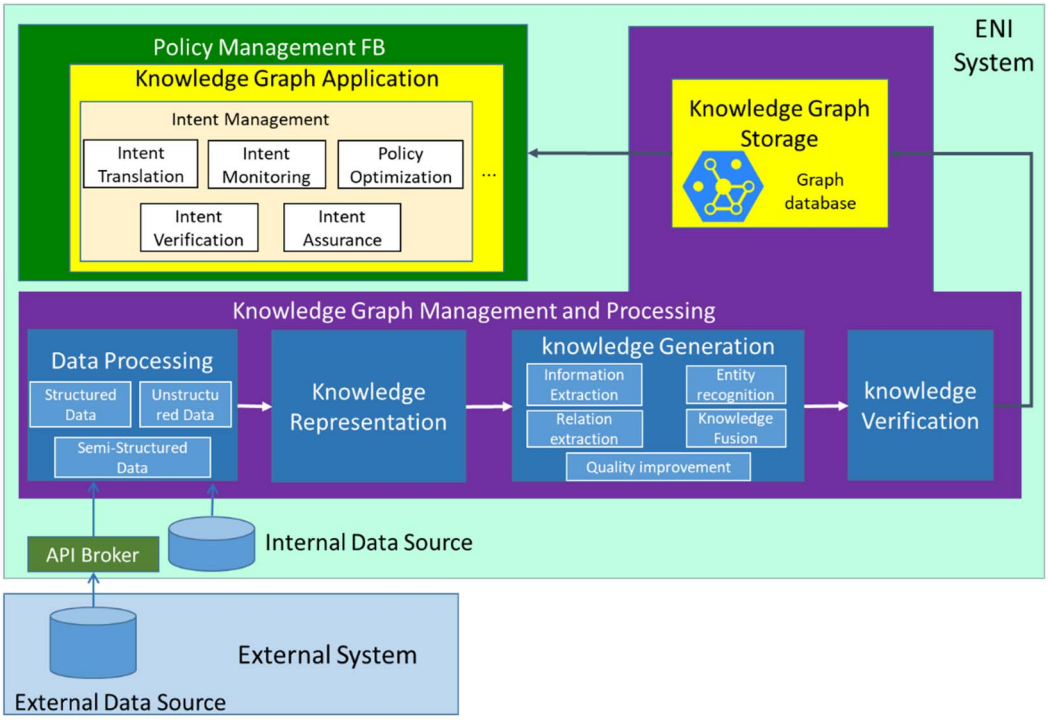

Figure 6.2: The construction procedure of Knowledge Graphs in the ENI System

6.2.2 Construction Procedures

6.2.2.1 Data Processing

In this step, data is pre-processed (data cleaning, data filtering, etc.) and then convert to a unified format for further processing.

The data sources required for building knowledge graphs in ENI System include:

- Internal data source inside ENI System, including structured data, semi-structured data, unstructured data;
- External data source from external system, including structured data, semi-structured data, unstructured data. The data is firstly sent to ENI API Broker for format conversion, and then sent to the Data Ingestion Functional Block.

6.2.2.2 Knowledge Representation

The common knowledge representation methods of knowledge graph mainly use symbolic representation methods. The symbolic representation methods include description logic, semantic network, production rules and framework system. Resource Description Framework (RDF), Web ontology language (OWL) and other similar knowledge representation methods commonly used in symbolic representation methods are recommended for the knowledge representation methods in the present document. OWL has a semantic data exchange standard and specification promoted by World Wide Web Consortium (W3C®). OWL has a strict semantic logic foundation, supports reasoning, and is compatible with the less complex representation language in RDF. When RDF triples (SPO) cannot meet the requirements of semantic expression, OWL, as a complete ontology language, can provide more powerful semantic expression components. These are stable and mature standards with broad application in knowledge management scenarios.

NOTE: Other knowledge representation methods (e.g. artificial neural networks) can also be used as long as they express intent as knowledge graphs in a way that is compatible with the proposed concepts in the present document.

6.2.2.3 Knowledge Generation

Knowledge generation includes vocabulary extraction, entity recognition, relation extraction, knowledge fusion and quality improvement.

Information extraction

Information extraction refers to the process of identifying and extracting words and phrases with potential semantic information from text data. Typical methods used in information extraction include:

1) **Bag-of-words model:** Represents text data as a collection of words, regardless of the order of words. The bag-of-words model can determine the important words in the text by counting the number of occurrences of each word in the text data.

2) **Word Embedding:** Representing words as low dimensional vectors that can be used for tasks such as text classification and sentiment analysis. Word embedding can be obtained through neural network learning or manually compiled.

3) **Topic model:** Identify topics or keywords in text data, such as Latent Semantic Analysis (LSA) based on probabilistic topic model.

4) **Part of Speech Tagging:** Determine the part of speech of words to better understand the text.

Entity recognition

Named Entity Recognition (NER) refers to the identification of entities with specific meanings in data such as text or images, such as person names, organizational structures, place names, etc.

Relation extraction

Relationship extraction is the process of extracting relationships between entities from raw data. This includes character pattern based extraction methods, grammar pattern based extraction methods, and semantic based extraction methods.

Knowledge fusion

Knowledge fusion refers to the integration of knowledge from different sources, forms, and structures. By using conflict detection, consistency checking, knowledge reasoning, and other methods to determine the correctness of knowledge. The verified knowledge is aligned and linked to entities to form a more complete, accurate, and reliable knowledge base.

Quality improvement

The quality improvement of knowledge graph refers to improving the accuracy, timeliness, completeness, reliability, and robustness of knowledge graph through a series of technical means and algorithms.

6.2.2.4 Knowledge Verification

Knowledge validation is the final check on the quality of the knowledge graph. This can be done through crowdsourcing verification, expert evaluation, cross validation, user feedback, and other methods. The verified knowledge graphs will be stored in the Knowledge Management Functional Block for further use.

Formal logic is the study of propositions, statements, or assertively used sentences and of deductive arguments. Formal logic is the basis of deductive reasoning, which is used to derive a conclusion from a set of premises with certainty. Formal logic can also be used to prove the correctness of algorithms by reasoning formally or mathematically about the algorithm.

The following lists prominent types of formal logic, from simplest to most robust.

1) Propositional Logic, also known as sentential logic, is a formal language that uses entire propositions, statements, or sentences to form more complicated propositions, statements, or sentences [i.9].

2) Predicate Logic is a formal language in which propositions are expressed in terms of predicates, variables, and quantifiers. A predicate is a statement or mathematical assertion that contains variables, sometimes referred to as predicate variables, and may be true or false depending on those variables' value or values. It is an extension to propositional logic, in which the notions of truth values, logical connectives, etc. still apply but propositional letters (which used to be atomic elements), will be replaced by a newer notion of proposition involving predicates and quantifiers.

3) First-order logic restricts predicates to individual objects and does not allow for quantification is only allowed over individual objects, and not over predicates or functions. It also restricts predicates to individual objects and does not allow for quantification over predicates or functions. This makes first-order logic less expressive but more computationally tractable than higher-order logic.

4) Higher-order logics (e.g. second-order logic) are distinguished from first-order logic by additional quantifiers and sometimes stronger semantics. For example, second-order logic allows quantification over sets of individuals, and third-order logic allows quantification over sets of sets. It also allows categorical axiomatizations of the natural numbers and real numbers, which are impossible with first-order logic. However, higher-order logic does not admit an effective, sound, and complete proof calculus.

NOTE: The present document will not consider higher-order logics.

5) Description logics (DLs) are a family of formal knowledge representation languages. Many DLs are more expressive than propositional logic but less expressive than first-order logic. In contrast to the latter, the core reasoning problems for DLs are usually decidable, and efficient decision procedures have been designed and implemented for these problems. The Web Ontology Language (OWL) and its profiles are based on DLs.

6) Deductive logic is a process of reasoning from one or more general statements (premises) to reach a logically certain conclusion. It involves drawing conclusions logically from other things that are already known.

7) Inductive logic is a process of reasoning based on an observation. Inductive reasoning seeks to establish general rules based on specific observations Inductive logic can only suggest that a conclusion is highly probable based on the premises observed.

8) Abductive logic is a process of reasoning that seeks the simplest and most likely conclusion from a set of observations. It also can only suggest that a conclusion is highly probable based on the premises observed.

AI programs are able to use deductive, inductive, and abductive logic. AI programs can also use description logics if specially designed to do so. Verified Knowledge graphs (as described in Knowledge Verification) inherently uses description logics as described in step 5) above.

6.2.2.5 Knowledge Graph Storage

Verified knowledge graphs are stored in the graph database (Knowledge Management Function Block) for the ENI System to query and use. The Knowledge Management Functional Block stores the constructed knowledge graphs, and responds to requests for knowledge application and knowledge sharing from other Functional Blocks.

The Graph Database (GDB) is a database that uses graph structures to store and navigate relationships (i.e. edges) between entities (i.e. nodes) in a graph. A graph database stores nodes and relationships instead of tables, or documents. An edge consists of a start node, end node, type, and direction, and an edge can have properties that describe different types of relationships, actions, ownership, and other concepts. There is no limit to the number and kind of relationships a node can have.

There are two popular models of graph databases: property graphs and RDF graphs. Property graphs model relationships between entities, and enable analytics to be performed based on these relationships. RDF graphs do the same, but in addition, conform to a set of W3C® standards. An RDF model represents a statement using three elements: two vertices connected by an edge reflecting the subject, predicate and object of a sentence (i.e. an RDF triple). Every vertex and edge is identified by a unique URI, or Unique Resource Identifier. The RDF model provides a way to publish data in a standard format with well-defined semantics, enabling information exchange.

This facilitates executing semantic queries with nodes, edges, and properties. Querying relationships is fast because they are directly stored in the database. Relationships can be intuitively visualized using graph databases, making them useful for heavily inter-connected data.

6.3 Using Knowledge Graphs to Manage Intent policies

6.3.1 Intent Translation

The translation of an Intent Policy involves taking a source language (e.g. an Intent policy authored in either a restricted language or a DSL) and transforming it into a target language (e.g. a form that the ENI System can understand). Once this is done, appropriate low-level network configurations can be created.

Firstly, intent creators express their intent or expected goals for the network in a restricted language or a DSL. The intent policies may be expressed in a high-level and abstract way, e.g. "ensuring high availability", "optimizing network performance", or "implementing security policies". A knowledge graph helps understand these Intents and transform them into the form that ENI System can understand. For example, a knowledge graph can refine and understand the Intent of "ensuring high network availability" as a series of indicators related to network availability: Average Minutes of Service, Instantaneous Failure Rate, Load Balancer Accuracy, Network Latency, etc.

The next step is to translate the Intent Policy into low-level network configurations. This step involves mapping the high-level intents to specific device configurations, protocols, and settings. A knowledge graph leverages its structured representation to derive the necessary configurations for network devices. For example, when the Intent Policy is to prioritize voice traffic over other data traffic to ensure optimal call quality, the knowledge graph can represent this Intent by specifying policies for traffic classification, Quality of Service (QoS) settings, and Access Control Lists (ACLs). It can then guide the generation of device-specific configurations and send them (via the Semantic Bus) to be translated into low-level ENI Policies. These ENI Policies will then be sent to the Assisted System via the API Broker.

NOTE: ETSI GS ENI 030 [i.4] describes how to use the transformer architecture to convert input policies into ENI policies. The detailed procedure of how to use knowledge graph to multi-stage translate Intent Policy will be provided in ETSI GS ENI 033 [i.10], which is not yet published.

6.3.2 Intent Verification

The translated Intent Policy will be verified to ensure that it is executed as expected. In this step, the expected results will be compared with the actual results. The ENI System can use knowledge graphs to verify Intent by checking any discrepancies or deviations between the expected and actual network behaviour. For example, if the Intent Policy is to execute a particular security policy, the knowledge graph can be used to check if the configured Access Control Lists (ACLs) and firewall rules align with the policy.

6.3.3 Intent Monitoring

The ENI System continuously monitors the network status after the deployment and execution of Intent policies to ensure the realization of Intent Policies. The ENI System collects data from network devices and systems, and updates the knowledge graphs. Once the changes in network state or user's Intent is detected that causes the original Intent Policy unable to be met, the Intent creator will be notified in a timely manner.

6.3.4 Intent Assurance

When deviations between expected and actual network states are identified, automatic Intent assurance mechanism will be triggered to restore the network to the desired state. Knowledge graph plays a crucial role in determining appropriate remedial measures based on identified issues. It can suggest correcting configurations, re-routing strategies, or adjusting strategies to address network differences. For example, if a network link fails or becomes congested, a knowledge graph can analyse the network topology, determine alternative paths, and recommend rerouting traffic to maintain connectivity and performance. It can also trigger automatic operations to update device configurations, such as disabling or enabling specific interfaces to solve problems.

6.3.5 Intent Optimization

Knowledge graph technology contributes to optimizing network policies and recommending best practices. By incorporating domain-specific knowledge and network expertise, the knowledge graph provides insights on network optimization strategies. For example, when a user expresses the intent to enhance network security, the knowledge graph can suggest security measures such as implementing Access Control Lists (ACLs), firewall rules, and Intrusion Detection Systems (IDS) based on known security vulnerabilities and successful security configurations.

6.4 Lifecycle Management

6.4.1 Lifecycle Management of Intent Policies

6.4.1.1 States of Intent Policy management

Lifecycle management of Intent Policy is defined and briefly introduced in clause 5.3 of ETSI GR ENI 008 [i.2]. An Intent Policy has its own distinct lifecycle. It is created, translated, verified, deployed, run, and retired. The Intent Creator may request Intent Policy to be created. Once an Intent Policy is created, the Intent Creator may then request that it be enabled or disabled when it is deployed or executed. Finally, the Intent Creator may request that the Intent Policy be removed.

In the present document, the states of Intent Policy are further defined as: **Created**, **Translated**, **Deployed**, **Executed**, **Tested**, **Deleted** and **Terminated**. It is beneficial for the ENI System and Intent Creator to define state values, so that they can accurately be aware of the nodes where Intent Policies are processed, and perform appropriate actions. Following are states of Intent Policy and corresponding state values:

- **Created:** the values of the Created state include {initialized, in_process, created, failed}.
- **Translated:** the values of the Translated state include {initialized, in_process, translated, failed_and_error, failed_and_conflict, failed_and_timeout, failed_and_other+}.
- **Deployed:** the values of the Deployed state include {initialized, in_process, deployed, failed}.
- **Executed:** the values of the Executed state include {initialized, in_process, executed, execution_aborted, execution_conflict_and_rollback, execution_conflict_and_error, execution_timeout, failed}.
- **Tested:** the values of the Tested state include {initialized, in_process, tested, testing_aborted, testing_timeout, failed}.
- **Deleted:** the values of the Deleted state include {initialized, in_process, deleted_locally, deleted_globally, failed}.
- **Terminated:** the values of the Terminated state include {Terminated_OK, Terminated_Error}.

6.4.1.2 State machine of Intent Policy processing

The state machine is able to represent the transition between different states of Intent Policy in an intuitive way. A general high-level state machine of Intent Policy processing is shown as follows in Figure 6.3.

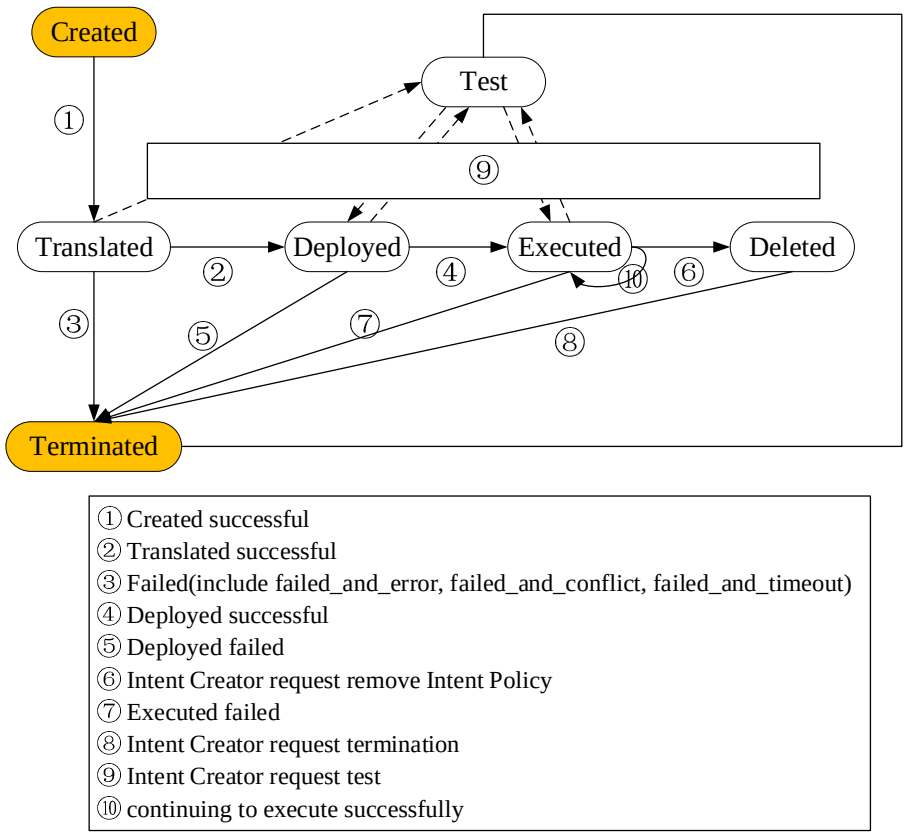

Figure 6.3: State transition of intent policy processing

- **Created:** the Intent Creator submitted Intent Policy to ENI System. The next state is Translated.
- **Translated:** the ENI System translates and transforms the original Intent Policy to a set of recommendations and/or commands that can be executed by the Intent Policy Target. If successful, the Intent Policy will enter the next state (Deployed). If failed, it will be terminated.

NOTE: The failure may be caused by conflict, error, out of scope, etc. ENI System may try to optimize internally (e.g. when encountered with issues) or negotiate with Intent Creator to solve the problems. If ENI System judges the issues cannot be solved within the specified time limit, it will report Intent Creator, who will decide whether to remove the Intent Policy. If the result is that Intent Policy should be removed, the next state is terminated. Deployed state and Executed state are similar, and will not be repeated.

- **Deployed:** the ENI System deploys the Intent Policy. If successful, the Intent Policy will enter the Executed state. If failed, it will be terminated.
- **Executed:** the ENI System executes the Intent Policy. If successful, it will wait for the next instruction of Intent Creator, include suspended (not execute), executed, or deleted. If failed, it will be terminated.
- **Deleted:** the Deleted state has two options:

  1. Deleted locally: deleting a policy from the active working set of policies but still retaining it in a repository.
  2. Deleted globally: removing a policy from both the active working set as well as all repositories. Both options need to get permission from the Intent Creator.

- **Tested:** testing is optional. If the Intent Creator requests the Intent Policy to be tested, the ENI System will attempt to do so. In addition, at any time after the Created state is entered, the Intent Creator is able to ask the ENI System to perform testing. For example, an Intent Policy of delay-sensitive scenarios can be directly deployed and executed after translation, and then tested, adjusted, and/or optimized during execution.

- **Terminated:** the end of the Intent Policy lifecycle. The termination of the Intent Policy was requested, or some other exception case occurred, and the ENI System has stopped executing the Intent Policy. This is the final clean-up action. Note that the Intent Creator can request to terminate an Intent Policy in any state.

6.4.1.3 Lifecycle Management Roles of Intent Policy

The lifecycle management of Intent Policy is initiated with communication between the Intent Creator and the ENI System.

Intent Creator

The Intent Creator is the author of the Intent Policy. The Intent Creator has two options for controlling the lifecycle of the Intent Policy, as follows:

1) The first is the ENI System can only enter the next state or a certain state with the permission of the Intent Creator. This is done by requesting the ENI System to perform an appropriate operation on the Intent Policy. For example, if the Intent Creator requests a policy to be created, then the ENI System will attempt to do so. If successful, the Intent Policy will not advance to the next state until the Intent Creator tells the ENI System to do so. It could also proceed to a certain state, at which point it will remain until either the policy fails or the Intent Creator requests a new state to be transitioned to.

2) The second is that, some transitions can be triggered automatically without the Intent Creator's permission. For example, when the Intent Policy is translated successfully, it is able to automatically enter the next state (deployed). In this case, the Intent Creator only needs to make decisions when the ENI System encounters issues, which greatly reduces the workload of Intent Creator.

ENI System

Once an ENI System accepts an Intent Policy, it is obliged to execute the lifecycle of the Intent Policy as well as possible based on the resources and solutions it has available. It is recommended that the ENI System provide status describing the Intent Policy throughout its lifecycle to the Intent Creator.

6.4.2 Lifecycle Management of Knowledge Used by Policies

6.4.2.1 Previous Work

NOTE: the text that follows uses an Intent Policy as an example.

In ETSI GR ENI 008 [i.2], Intent Knowledge refers to the 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.1] within the Repository Management Functional Block, respectively.

The above definition of Intent Knowledge has certain limitations. In the present document, Intent knowledge (knowledge used by the Intent Policy) is defined as the knowledge used in the whole lifecycle process of Intent Policy processing. It may contain the knowledge generated during Intent Policy translation, testing, conflict detection, as well context knowledge, knowledge from external sources and so on.

6.4.2.2 Lifecycle Management Roles of Knowledge Used by Policies

Different from Intent Policy, the lifecycle management of knowledge in the present document used by Intent Policy mainly takes place within ENI System (Knowledge Management Functional Block).

The Knowledge Management Functional Block in ENI System is used to store knowledge and generate new knowledge through inferences. Knowledge management works with data, information, knowledge, and wisdom, directing which context and situation information are applied to the raw data, transforming it to information and then knowledge. In addition, this Functional Block defines a formal and consensual representation of knowledge so that a computer system could implement the machine learning algorithms and perform reasoning using the knowledge (see clause 6.3.4 of ETSI GS ENI 005 [i.1]).

6.4.2.3 Lifecycle of Knowledge Used by Policies

Figure 6.4 presents the general lifecycle of knowledge in the present document used by Intent Policy. This lifecycle follows the evolution process of Create-Store-Read-Update-Delete (CSRUD). In this figure:

- **Create:** there are two main sources of knowledge used by Intent Policy: internally generated knowledge and externally absorbed knowledge. Internally generated knowledge refers to the knowledge created by analysing policyMetadata and Intent Policy related information, understanding the content of these data and information and any contextual information that applies, and then creating knowledge that can be used to translate the Intent Policy. Externally absorbed knowledge may include common sense knowledge and domain specific knowledge. Combination, filtering and fusion may be required for knowledge from different sources.
- **Store:** the knowledge used by Intent Policy is stored in knowledge repository.
- **Read:** during Intent Policy processing, knowledge will be retrieved for reasoning and decision-making.
- **Update:** update the existing knowledge repository when new knowledge comes.
- **Delete:** delete useless and invalid knowledge.

Figure 6.4: Lifecycle of Knowledge Used by Intent Policy

7 Use Cases

7.1 Use cases for operators' business

7.1.1 Use Case #1-1: Intent-driven operation for user-centric cloud-network convergence services

7.1.1.1 Use case context

Cloud services are constantly reshaping social production and lifestyles. Home broadband applications (such as cloud games), large enterprises (such as governmental and finance enterprises), and small- and medium-sized enterprises (such as hospitals and e-commerce enterprises) are greatly facilitated by cloud services. Enterprises are using cloud services to reduce O&M costs, and applications are gradually being migrated to the cloud. The multi-cloud strategy and multi-cloud services have become a trend in the industry. One-hop cloud access via Optical Transport Network (OTN) has been widely used in the industry and has become the mainstream choice for enterprises and cloud leased line services due to its various advantages, including high bandwidth, low latency, hard isolation, high reliability, and one-hop cloud access. However, it has always been difficult to manage services on OTN due to network configuration complexity and multi-vendor collaboration challenges. To overcome the difficulty, the system needs to be more intelligent so that a user without relative knowledge can express his/her demands.

7.1.1.2 Description of the Use Case

7.1.1.2.1 Overview

Cloud Private Line (CPL) services connect cloud service users to edge or cloud data centres, and edge or cloud data centres to each other, with deterministic connection performance. They may represent point-to-point, point-to-multipoint, multipoint-to-point, or multipoint-to-multipoint connectivity service topologies, and are implemented typically using connection-oriented technologies. Data centres may be operated by the CPL service customer, by the CPL services provider, by some other service provider(s), or by any combination of these. CPL service traffic consists of machine-to-machine data flows with a range of characteristics. Dynamically mass-customized CPL services are driven operationally by CPL service consumers, either semi-manually (e.g. through a user-facing provisioning portal) or directly by scheduling software systems. Clearly intent is a useful service-driving paradigm to support such capability.

7.1.1.2.2 Motivation

In the cloud-network convergence scenario, users need enough bandwidth as well as the computing resources. To be more specific, the user can express an intent of creating a cloud-network convergence service. This intent is then automatically fulfilled by provisioning the corresponding services and allocating the required resources. The already fulfilled intent can be modified by the user. The Intent-based system monitors the parameters of the cloud-network convergence service (e.g. bandwidth usage), and automatically triggers appropriate closed-loop actions (e.g. increase max bandwidth) in order to fulfil the purpose of the intent policy.

The user's intent, expressed in a natural language, can be translated by the ENI System. The use of the ENI System allows users to achieve their intents only through ordinary descriptions without understanding specific technical details. In this use case, the intent translation process in ENI System is formulated as a Question Answering (QA) problem, in which the key information can be accurately extracted from a dialogue between the user and the ENI System. The ENI System transforms an intent from a simple declarative form to a robust technology-specific form in an automated way.

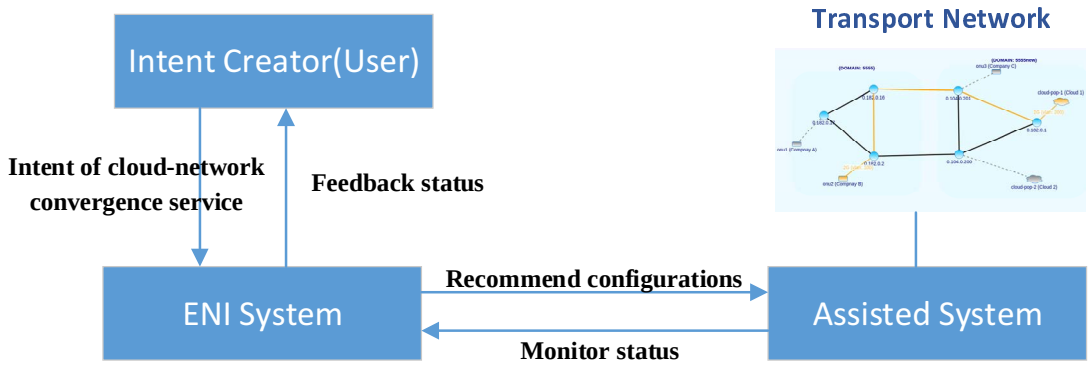

Figure 7.1: A General Architecture of Cloud-network Convergence Services

7.1.1.2.3 Actors and Roles

User: the cloud-network convergence service consumer.

ENI System: receives and translates user's intent and make closed-loop decisions.

Network Operator: manages the network topology and computing resources.

7.1.1.2.4 Initial context configuration

Both topology of transmission network and information of computing resources have been input into the ENI System.

Optimization policy has been input into the ENI System.

The Natural Language Processing (NLP) algorithm is well trained and successfully runs in the ENI System.

7.1.1.2.5 Triggering conditions

The ENI System receives the user's intent. For example, a simple user input describing a customer's need in a natural language may be "I need a connection from company A to Cloud Two, with a bandwidth of 2 Gbps".

7.1.1.2.6 Operational flow of actions

1) The ENI System receives the user's intent. The user expresses, in a natural language, an intent to connect one (or more) enterprises to one (or more) clouds, as well as his/her expectation for the quality of the service.

2) The ENI System translates the user intent into more specific service intent parameters.

3) The ENI System communicates with the user to verify that its translation meets the user needs.

4) The ENI System sends the recommend configurations to the assisted system.

5) The Assisted System receives the recommended configurations to fulfil the user's intent.

6) The user validates whether the demand is met.

NOTE: This second validation checks the runtime interactions between the user's intent and the rest of the services supported by the Assisted System. The previous validation only ensures that the translation of the intent is correct.

7) The ENI System continuously monitors the conditions of the network against the intent specification to ensure compliance with the intent.

8) If the network state cannot meet the user's intent, the ENI System sends new recommended configurations to the Assisted System.

7.1.1.2.7 Post-conditions

The intent translation and intent life-cycle management have been achieved.

The intent assurance is achieved by closed-loop automation.

7.2 Use cases for vertical industry

7.2.1 Use Case #2-1: Intent-Driven Home Intranet Management

7.2.1.1 Use case context

Home router hardware has been upgraded for generations. Currently, the hardware performance of a home router is capable of more complex jobs, other than routing. Meanwhile, the latest router system normally is compatible with complex jobs, such as workload balancing, resource orchestration, etc. However, to accomplish these jobs mentioned above, the user currently should have some computer and network management skills. This prerequisite has become a barrier to making the home Intranet develop further. To overcome this difficulty, the system needs to be more intelligent so that a user without relative knowledge can express his/her demands and the router responds correctly.

With the vigorous development of the Internet of Things and the emergence of various intelligent devices, the concept of 'Smart Home' has gradually become one of the foci of the industry. One of the core ideas of Smart Home is to manage the family Intranet according to user intent policies.

7.2.1.2 Description of the Use Case

7.2.1.2.1 Overview

The access of many smart devices to the home network brings many new services such as Extended Reality (XR), Network Attached Storage (NAS), Ultra High Definition (UHD) video streaming; these new services gradually complicates the structure of the home network and requires more dedicated network management.

Normally, network management is a very complex process for a home user, which takes a considerable amount of time for learning the relative knowledge To accomplish family Intranet management, a user is required to:

- Have basic network knowledge. The user needs to get familiar with common network protocols and packet transmission mechanisms. The user needs to be able to come up with a reasonable management plan.
- Be familiar with Network Management. The user needs to have experience in managing networks so that the user can understand what different network telemetry mean, which kind of data is important to monitor, etc.

7.2.1.2.2 Motivation

The current home network management method is simple. The user should input network policy directly to the network. Thus, there are certain requirements for users' professional skills. However, home users often do not have much experience on network management. Moreover, different from other network management scenarios, since home network users usually do not have much relevant knowledge. Therefore, the user's intention input is usually more obscure than for other use cases. For example, these users describe requirements for the service layer only. To achieve the request, the network needs to orchestrate the resources.

ENI System can translate the intent policy input into a restricted natural language. The introduction of the ENI System in the home network environment can effectively reduce the knowledge requirement threshold for users to manage the network. With the help of an ENI System, users can not only use traditional computer language to operate the network, but also express their expectations for the network or a specific business running on the network by directly using natural language. The ENI System can accurately extract the keywords in the sentences, determine the real needs of users, translate those needs into appropriate network commands, and automatically dispatch them to the corresponding home network devices.

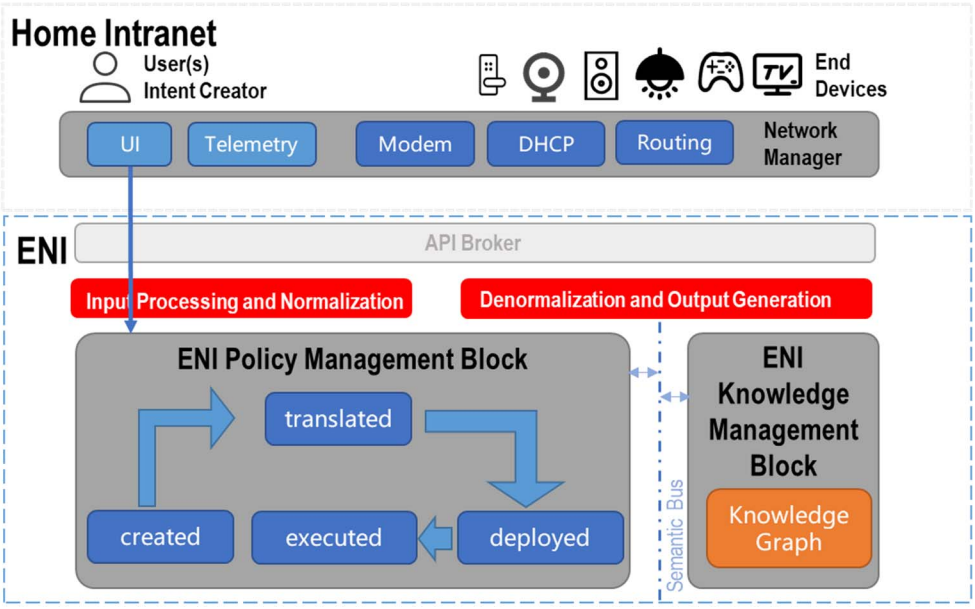

Figure 7.2: A General Architecture of Home Intranet Management

7.2.1.2.3 Actors and Roles

User: the family Intranet user. Assume in this case, the user has not learned any relevant skills.

End devices: intelligent devices connected to the family Intranet. Normally, they are laptops, intelligent TVs, cell phones, VR helmets, etc.

ENI system: the system that receives intent polices from the users, and translates the intent policies into polices.

Network manager: the system that monitors services provided and changes their configurations according to user intent.

Running applications: applications that are currently running on any end device.

Terminated applications: applications that were running on end devices but terminated before the ENI System starts processing a new intent policy.

7.2.1.2.4 Initial context configuration

End Devices are connected to the home Intranet and they are all functional.

The home Intranet runs in a stable state.

ENI constructs a knowledge graph with the following attributes and relations and stores it in the knowledge management Functional Block.

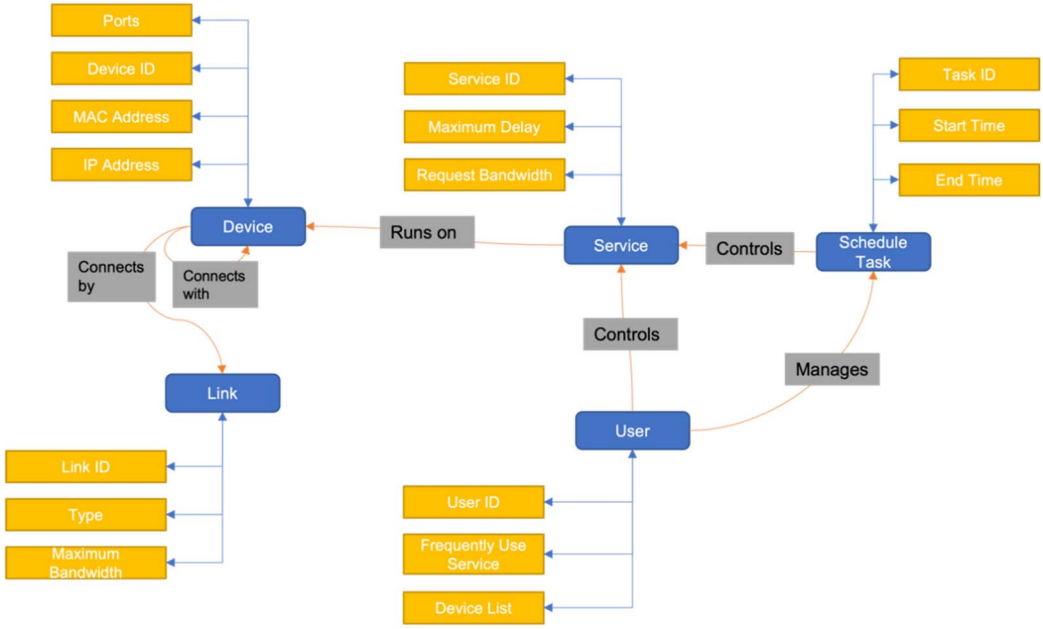

Figure 7.3: Structure of the Home Intranet Management Knowledge Graph

7.2.1.2.5 Triggering conditions

The user inputs an intent policy which implies a state the user expects the network or running business to achieve.

7.2.1.2.6 Operational flow of actions

1) The ENI System receives the new intent policy from users or running applications. Then the ENI System translates the intent to a form that it understands.

2) The ENI System determines if there are any conflicts among new intent policy and processed intent polices. If there are any, jump to step 2a). Otherwise, continue to step 2b).

2a) If the processed Intent Policies were from terminated applications, continue to 2b). Otherwise, jump to 5).

2b) The ENI System translates the new intent policy into suitable recommendations or commands with the assistance of the knowledge graph.

3) The ENI System sends the translated polices to the home router.

4) The home router receives the polices and modifies its resources accordingly.

5) End devices modify their resource consuming strategy to accommodate the change of network.

6) The ENI System writes the information down into a log and reports it to the Network Manager along with recommendations and commands.

7) The user validates whether the application's or user's demand is met.

8) If not, the user may input new intent policies in the form of natural language or a computer programming language to correct the previous intent policy.

Step 7) is optional. If the network state meets the user demand, the step should be skipped.

7.2.1.2.7 Post-conditions

The state of the home Intranet or a business has been changed accordingly.

The user's intent policy has been achieved.

8 Conclusions and recommendations

The present document discusses the detailed procedures of Intent Policy processing, conflict detection and resolution between different Intent Policies, knowledge management for Intent Policies, as well as related use cases.

More specifically, it has identified and described:

- The detailed Intent Policy processing operation, including parsing, translation, testing, deployment and execution.
- The procedure of Intent Policy conflict detection and the solution of conflict resolution.
- The procedure of constructing Knowledge Graphs in the ENI System, and how the ENI System utilizes knowledge graphs to manage Intent Policies.
- Lifecycle management of Intent Policies, including states and the transition between different states.
- Lifecycle management of knowledge used by Policies, including create, store, read, update and delete.
- Use cases related to operators' business and vertical industry. For example, Intent-driven operation for user-centric cloud-network convergence services and Intent-Driven Home Intranet Management.

It is recommended to improve the present document in the next Release to work on the following aspects:

- Enhanced procedures of Intent Policy processing, e.g.:
  - Intent Policy assurance.
  - Enhanced solutions of conflict resolution and related use cases.
- Knowledge management in the ENI System, e.g.:
  - Add a Knowledge Graph Functional Block for the ENI System.
  - Specify the knowledge management in the ENI System, including requirement, detailed knowledge management functionalities, interaction with other Functional Blocks and use cases for knowledge management in the ENI System.
  - Additional scenarios and use cases for using knowledge graphs to manage Intent Policy.
- Enhanced state machine of Intent Policy processing.
- Security considerations need to be detailed and considered.

History

Document history		
V4.1.1	May 2024	Publication