ETSI GS ENI 005 V1.1.1 (2019-09)

Experiential Networked Intelligence (ENI);
System Architecture

Reference

DGS/ENI-005

Keywords

management, network, policy management

Contents

Intellectual Property Rights 8
Foreword....8
Modal verbs terminology....8
Executive summary 8
Introduction 8
1 Scope 9
2 References 9
2.1 Normative references 9
2.2 Informative references....10
3 Definition of terms, symbols and abbreviations....11
3.1 Terms....11
3.2 Symbols....13
3.3 Abbreviations 13
4 Overview of System Architecture (informative)....13
4.1 Introduction 13
4.2 Motivation for ENI....13
4.3 Benefits of ENI....14
4.4 High-Level Description of the ENI System Architecture....15
4.4.1 Overall Description....15
4.4.2 The Assisted System....15
4.4.2.1 Introduction....15
4.4.2.2 Class 1: An Assisted System that has No AI-based Capabilities 16
4.4.2.3 Class 2: An Assisted System with AI that is Not in the Control Loop....17
4.4.2.4 Class 3: An Assisted System with AI Capabilities in its Control Loop 18
4.4.2.4.1 Introduction 18
4.4.2.4.2 Class 3 Options....18
4.4.2.5 Summary of Interaction between the Assisted System and ENI 19
4.4.3 Mode of Operation....20
4.4.3.1 Allowed Modes of Operation....20
4.4.3.2 Setting the Mode of Operation....20
4.4.3.3 Interaction with the Assisted System 21
4.4.3.4 Selecting a Mode of Operation for a Class of Decisions....21
4.4.3.5 Communication of Mode of Operation 21
4.4.3.6 Normal Operation of the Selected Mode of Operation....21
4.4.3.7 Exception Handling for the Selected Mode of Operation 22
4.4.4 Communication....22
4.4.4.1 Overview....22
4.4.4.2 Discovery 22
4.4.4.3 Direct Configuration 22
4.4.4.4 Negotiation....22
4.4.4.5 Switching the Mode of Operation 23
4.4.4.5.1 Overview 23
4.4.4.5.2 Case 1: ENI Indirectly Instructs the Assisted System to Switch Modes....24
4.4.4.5.3 Case 2: ENI Directly Instructs the Assisted System to Switch Modes....24
4.4.5 Functional Concepts 24
4.4.5.1 Introduction....24
4.4.5.2 Functional Block Definitions 24
4.4.5.3 State....24
4.4.5.4 Control Loop Operation 25
4.4.5.5 Inferencing 25
4.4.5.6 Data, Information, Knowledge, and Wisdom....25
4.4.5.6.1 Introduction 25
4.4.5.6.2 Data 27

4.4.5.6.3 Information ... 27
4.4.5.6.4 Knowledge ... 27
4.4.5.6.5 Wisdom ... 27
4.4.5.7 Measured vs. Inferred Knowledge ... 27
4.4.6 ENI Reference Points ... 28
4.4.6.1 Definition of an ENI Reference Point ... 28
4.4.6.2 Definition of an ENI External Reference Point ... 28
4.4.6.3 Definition of an ENI Internal Reference Point ... 28
4.4.7 ENI Interfaces ... 28
4.4.7.1 Definition of an ENI Interface ... 28
4.4.7.2 Definition of an ENI Hardware Interface ... 28
4.4.7.3 Definition of an ENI Software Interface ... 28
4.4.7.4 Definition of an ENI Application Programming Interface ... 28
4.4.7.5 Comparison of ENI Software Interfaces with ENI APIs ... 29
4.5 Functional Architecture ... 29
4.5.1 Functional Block Diagram of the ENI System ... 29
4.5.2 API Broker ... 31
4.5.3 ENI System Functional Blocks ... 31
4.5.3.1 Introduction ... 31
4.5.3.2 Input Processing ... 31
4.5.3.2.1 Overview ... 31
4.5.3.2.2 Data Ingestion Functional Block ... 31
4.5.3.2.3 Normalization Functional Block ... 32
4.5.3.3 Analysis ... 32
4.5.3.3.1 Knowledge Management and Processing ... 32
4.5.3.4 Situation-based, Model-driven, Policy Generation ... 33
4.5.3.4.1 Overview ... 33
4.5.3.4.2 Situation Awareness Functional Block ... 33
4.5.3.4.3 Model Driven Engineering Functional Block ... 33
4.5.3.5 Output Generation ... 34
4.5.3.5.1 Overview ... 34
4.5.3.5.2 Denormalization Functional Block ... 34
4.5.3.5.3 Output Generation Functional Block ... 35
4.5.4 Decision-Making ... 35
4.5.4.1 Overview ... 35
4.5.4.2 Decision-Making using Hindsight ... 35
4.5.4.3 Decision-Making using Deterministic Processing ... 35
4.5.4.4 Decision-Making using Predictive Processing ... 35
4.5.4.5 Decision-Making using Cognitive Processing ... 36

5 ENI Architectural Requirements ... 36
5.1 Introduction ... 36
5.2 Functional Architectural Requirements for ENI Operation ... 36
5.3 Architectural Requirements for Mode of Operation ... 39
5.4 Non-Functional Architectural Requirements for ENI Operation ... 40
5.5 Reference Point Requirements ... 40

6 ENI Reference Architectural Framework ... 41
6.1 Introduction ... 41
6.2 Design Principles of the ENI System architecture ... 41
6.2.1 Overview ... 41
6.2.2 Nesting of Functional Blocks ... 42
6.2.3 Negotiation ... 43
6.2.3.1 Introduction ... 43
6.2.3.2 Distributive Negotiation ... 43
6.2.3.3 Integrative Negotiation ... 43
6.2.3.4 Functional Model: an Informative Example ... 44
6.3 Architectural Functional Blocks of the ENI System ... 44
6.3.1 Introduction ... 44
6.3.2 Data Ingestion Functional Block ... 45
6.3.2.1 Introduction ... 45
6.3.2.2 Motivation ... 45

6.3.2.3 Function of the Data Ingestion Functional Block....46
6.3.2.3.1 Introduction....46
6.3.2.3.2 Data Filtering....46
6.3.2.3.3 Data Correlation....46
6.3.2.3.4 Data Cleansing....47
6.3.2.3.5 Data Anonymization and Pseudonymization....47
6.3.2.3.6 Data Augmentation....47
6.3.2.3.7 Data Labelling....47
6.3.2.4 Operation of the Data Ingestion Functional Block....47
6.3.3 Data Normalization Functional Block....48
6.3.3.1 Introduction....48
6.3.3.2 Motivation....49
6.3.3.3 Function of the Data Normalization Functional Block....49
6.3.3.4 Operation of the Data Normalization Functional Block....49
6.3.3.4.1 Introduction....49
6.3.3.4.2 Database Design Analogy....49
6.3.3.4.3 Normalization for Machine Learning....50
6.3.3.4.4 Applying Normalization to ENI....51
6.3.4 Knowledge Management Functional Block....52
6.3.4.1 Introduction....52
6.3.4.2 Inferencing....52
6.3.4.3 Motivation....52
6.3.4.4 Function of the Knowledge Management Functional Block....53
6.3.4.5 Operation of the Knowledge Management Functional Block....53
6.3.4.5.1 Introduction....53
6.3.4.5.2 Observe Functionality....54
6.3.4.5.3 Orient Functionality....54
6.3.4.5.4 Decide Functionality....55
6.3.4.5.5 Act Functionality....55
6.3.5 Context-Aware Management Functional Block....55
6.3.5.1 Introduction....55
6.3.5.2 Motivation....56
6.3.5.3 Function of Context Awareness....56
6.3.5.4 Operation of the Context Awareness Functional Block....56
6.3.6 Cognition Framework Functional Block....58
6.3.6.1 Introduction....58
6.3.6.2 Motivation....59
6.3.6.3 Function of the Cognition Framework Functional Block....59
6.3.6.4 Operation of the Cognition Framework Functional Block (informative)....59
6.3.6.4.1 Introduction....59
6.3.6.4.2 The Symbolic Approach....60
6.3.6.4.3 The Connectionist Approach....61
6.3.7 Situational Awareness Functional Block....62
6.3.7.1 Introduction....62
6.3.7.2 Motivation....62
6.3.7.3 Function of Situational Awareness....62
6.3.7.4 Operation of the Situational Awareness Functional Block....62
6.3.7.5 Difference between Context Awareness and Situational Awareness....65
6.3.8 Model Driven Engineering Functional Block....65
6.3.8.1 Introduction....65
6.3.8.2 Motivation....65
6.3.8.3 Function of the Model Driven Engineering Functional Block....66
6.3.8.4 Operation of the Model Driven Engineering Functional Block....66
6.3.9 Policy Management Functional Block....68
6.3.9.1 Introduction....68
6.3.9.2 Motivation....68
6.3.9.3 Function of the Policy Management Functional Block....69
6.3.9.4 Operation of the Policy Management Functional Block....70
6.3.10 Denormalization Functional Block....72
6.3.10.1 Introduction....72
6.3.10.2 Motivation....73
6.3.10.3 Function of the Denormalization Functional Block....73

6.3.10.4 Operation of the Denormalization Functional Block 73
6.3.11 Output Generation Functional Block 74
6.3.11.1 Introduction 74
6.3.11.2 Motivation 75
6.3.11.3 Function of the Output Generation Functional Block 75
6.3.11.4 Operation of the Output Generation Functional Block 75
6.4 API Broker 76

7 Reference Points 76
7.1 Introduction 76
7.2 Reference Point Overview 76
7.3 Reference Point Definitions 79
7.3.1 Reference Point $E_{oss-eni-dat}$ 79
7.3.2 Reference Point $E_{oss-eni-cmd}$ 79
7.3.3 Reference Point $E_{app-eni-ctx}$ 79
7.3.4 Reference Point $E_{app-eni-oth}$ 80
7.3.5 Reference Point $E_{app-eni-kno}$ 80
7.3.6 Reference Point $E_{bss-eni-dat}$ 80
7.3.7 Reference Point $E_{bss-eni-cmd}$ 81
7.3.8 Reference Point $E_{usr-eni-pol}$ 81
7.3.9 Reference Point $E_{or-eni-dat}$ 81
7.3.10 Reference Point $E_{or-eni-cmd}$ 81
7.3.11 Reference Point $E_{inf-eni-dat}$ 82
7.3.12 Reference Point $E_{inf-eni-cmd}$ 82

8 Interacting with Other Standardized Architectures 82
8.1 Introduction 82
8.2 Generic Architecture 83
8.3 Generic SDO Interaction Architecture 84
8.3.1 Introduction 84
8.4 Interaction with NFV MANO 84
8.4.1 High Level description of the NFV MANO - ENI Interaction 84
8.4.2 Initial proposals for interaction scenarios 86
8.4.2.1 Introduction 86
8.4.2.2 Scenario 1: Passive Notification to NFV MANO 86
8.4.2.3 Scenario 2: Active Data Analysis for NFV MANO 86
8.4.2.4 Scenario 3: Active Assistance to the NFV MANO System 86
8.4.2.5 Scenario 4: Active Assistance to the Assisted System 86

9 Areas for Future Study 87
9.1 Introduction 87
9.2 Future Study for the Assisted System 87
9.3 Models 87
9.3.1 Future Study for the Information Model 87
9.3.2 Future Study for Data Models 88
9.4 Input and Output Processing 88
9.4.1 Future Study for Data Ingestion and Normalization 88
9.4.2 Future Study for Denormalization and Output Generation 88
9.5 Knowledge Processing 88
9.5.1 Future Study for Knowledge Management 88
9.5.2 Future Study for Context Management 89
9.5.3 Future Study for Cognition Management 89
9.5.4 Future Study for Situation Awareness 89
9.6 Future Study for MDE Processing 89
9.7 Future Study for Policy Management and Processing 89
9.8 Future Study for Reference Points 90
9.9 Future Study for Interfaces 90
9.10 Future Study for APIs and the API Broker 90
9.11 Future Study for New ENI Functional Blocks 91
9.12 Future Study for Interaction with Other Organizations 91
9.13 Future Study for Security 91

Annex A (informative): SDO and Open Source Interactions 93

A.1 Integration with Other SDOs and Open Source Communities ... 93
A.1.1 Introduction ... 93
A.1.2 Interaction with MEF-LSO ... 93
A.1.3 Interaction with BBF CloudCO ... 93
A.2 Interaction with Open Source Communities ... 95
Annex B (informative): ENI Architectural Evolution ... 96
B.1 ENI Architecture Evolution Motivation ... 96
B.2 ENI Architecture Evolution Proposal ... 96
B.3 Proposed Definition of the ENI Phases ... 96
Annex C (informative): Authors & contributors ... 97
History ... 98

Intellectual Property Rights

Foreword

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

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**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.

Executive summary

The present document specifies a high-level functional abstraction of the ENI System Architecture in terms of Functional Blocks and External Reference Points. This includes describing how different classes of systems interact with ENI. Processes, models, and detailed information are beyond the scope of the present document.

Introduction

The present document defines a high-level functional abstraction of the ENI System Architecture. The organization of the present document is as follows. Clause 1 defines the scope of the present document. Clauses 2 and 3 provide normative and informative references and definition of terms, respectively. Clause 4 provides an informative overview of the ENI System Architecture, including its motivation, benefits, important concepts, and an overview of its Functional Blocks. Clause 5 lists requirements of the ENI System Architecture. Clause 6 defines important design principles of the ENI System Architecture, and then specifies the different Functional Blocks that make up the ENI System Architecture. Clause 7 specifies the External Reference Points of the ENI System Architecture. Clause 8 describes how ENI interacts with other SDO Systems and clause 9 delineates a list of future study items.

1 Scope

The present document specifies the functional architecture of an ENI System, which is a high-level decomposition of an ENI System into its major components, along with a characterization of the externally visible behaviour (e.g. as defined by a set of reference points) of the components. This includes:

- defining the functionality and behaviour of a system that satisfy the ENI Requirements (ETSI GS ENI 002 [6]);
- defining a functional architecture, in terms of Functional Blocks, that addresses the goals specified by the ENI Use Cases (ETSI GS ENI 001 [5]);
- defining Reference Points used by the above Functional Blocks for all communication with systems and entities that are external to the ENI System;
- proposing a progression plan towards full support of the proposed ENI System and intermediary level of compliance (e.g. support of some architecture components or a subset of the Reference Points).

2 References

2.1 Normative 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.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

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 necessary for the application of the present document.

[1] OMG formal/2015-06-03: "OMG Systems Modeling Language".

[2] BBF TR-384: "Cloud Central Office Reference Architectural Framework", January 2018, G. Karagiannis, D. Hai.

[3] ETSI GS NFV-MAN 001: "Network Functions Virtualisation (NFV); Management and Orchestration".

[4] IETF RFC 4949: "Internet Security Glossary, Version 2", Shirey, R., August 2007.

[5] ETSI GS ENI 001 (V2.1.1): "Experiential Networked Intelligence (ENI); ENI use cases".

[6] ETSI GS ENI 002 (V2.1.1): "Experiential Networked Intelligence (ENI); ENI requirements".

[7] MEF 78: "MEF Technical Specification: MEF Core Model", Strassner, J., editor, January 2019.

[8] Strassner, J., Agoulmine, N., Lehtihet, E.: "FOCALE - A Novel Autonomic Networking Architecture", ITSSA Journal 3(1), 64-79, 2007.

[9] Boyd, J. R.: "The Essence of Winning and Losing", June, 1995.

[10] Strassner. J.C.: "Knowledge Representation, Processing, and Governance in the FOCALE Autonomic Architecture", book chapter, 2011, Elsevier.

[11] NIST: "Role-Based Access Control, Second Edition".

NOTE: Available at https://us.artechhouse.com/Role-Based-Access-Control-Second-Edition-P1386.aspx.

[12] NIST: "Attribute Based Access Control".

NOTE: Available at https://csrc.nist.gov/publications/detail/book/2017/attribute-based-access-control.

[13] IETF RFC 8446: "The Transport Layer Security (TLS) Protocol Version 1.3", August 2018.

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] Strassner, J.: "Policy-Based Network Management", Morgan Kaufman, ISBN 978-1558608597, September 2003.

[i.2] Strassner, J., de Souza, J.N., Raymer, D., Samudrala, S., Davy, S., Barrett, K.: "The Design of a Novel Context-Aware Policy Model to Support Machine-Based Learning and Reasoning", Journal of Cluster Computing, Vol 12, Issue 1, pages 17-43, March, 2009.

[i.3] ETSI GR ENI 003 (V1.1.1): "Experiential Networked Intelligence (ENI); Context-Aware Policy Management Gap Analysis".

[i.4] Strassner, J., Betser, J., Ewart, R., Belz, F.: "A Semantic Architecture for Enhanced Cyber Situational Awareness", Secure and Resilient Cyber Architectures Conference, MITRE, 2010.

[i.5] Gamma, E., Helm, R. Johnson, R., Vlissides, J.: "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Nov, 1994. ISBN 978-0201633610.

[i.6] Bäumer, D. Riehle, W. Siberski, M. Wulf: "The Role Object Pattern", Proceedings of the 1997 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '97), ACM Press, 1997, Page 218-228.

[i.7] Rowley, J.: "The wisdom hierarchy: representations of the DIKW hierarchy", Journal of Information and Communication Science, 33(2): 163-180.

[i.8] Chin, K.O., Ganb, K.S., Alfred, R., Anthony, P, and Lukose, D.: "Agent Architecture: An Overview", Transactions on Science and Technology, vol 1, No 1, pp 18-35, 2014.

[i.9] Shehory, O and Sturm, A., editors: "Agent-Oriented Software Engineering", Springer, 2014.

[i.10] Martin, R. C.: "Agile Software Development, Principles, Patterns, and Practices", Prentice Hall, 2003 ISBN 978-0135974445.

[i.11] Ritter, F.E., Tehranchi, F., Oury, J.D.: "ACT-R: A Cognitive Architecture for Modeling Cognition", Wiley Interdisciplinary Reviews, Cognitive Science 10(4): e1488.

NOTE: Available at http://act-r.psy.cmu.edu.

[i.12] IETF RFC 8328: "Policy-Based Management Framework for the Simplified Use of Policy Abstractions (SUPA)", Liu, W., Xie, C., Strassner, J., Karagiannis, G., Klyus, M., Bi, J., Cheng, Y., and D. Zhang .

NOTE: Available at https://www.rfc-editor.org/info/rfc8328.

[i.13] Rothenberg, J.: "The Nature of Modelling", Artificial Intelligence, Simulation, and Modeling, John Wiley and Sons, Inc., 1989, pp. 75-92.

NOTE: Available at https://www.rand.org/content/dam/rand/pubs/notes/2007/N3027.pdf.

[i.14] Recommendation ITU-T 9594-1: "Information Technology - Open Systems Interconnection - The Directory: Overview of Concepts, Models, and Services".

[i.15] Recommendation ITU-T 9594-7: "Information Technology - Open Systems Interconnection - The Directory: Selected Object Classes".

[i.16] ETSI GR ENI 003 (V1.1.1): "Experiential Networked Intelligence (ENI); Context-Aware Policy Management Gap Analysis".

[i.17] MEF Technical Specification: "Policy-Driven Orchestration", Call for Comments v0.7, August 2017, Strassner, J., editor.

NOTE: Available at https://wiki.mef.net/download/attachments/59846378/MEF%20PDO%20CfCB1.zip?api=v2.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

agent: computational process that implements the autonomous, communicating functionality of an application

- **software agent:** software that acts on behalf of a user or another program
- **software autonomous agent:** software agent that acts on behalf of the entity that owns it without any communication from the owning entity
- **software intelligent agent:** software agent that reasons about its environment and take the best set of actions to satisfy a set of goals

NOTE: This has the connotation of containing AI mechanisms to provide the reasoning and decision-making capabilities.

- **software multi-agent:** set of software agents that are physically separate that work together to satisfy a set of goals

API: set of communication protocols, code, and tools that enable one set of software components to interact with either a human or a different set of software components

NOTE: This is also known as an Application Programming Interface.

API Broker: software entity that mediates between two systems with different APIs, enabling the two different systems to communicate transparently with each other

architecture: set of rules and methods that describe the functionality, organization, and implementation of a system

- **cognitive architecture:** system that learns, reasons, and makes decisions in a manner resembling that of a human mind

NOTE: Specifically, the learning, reasoning, and decision-making is performed using software that makes hypotheses and proves or disproves them using non-imperative mechanisms that typically involve constructing new knowledge dynamically during the decision-making process.

- **deliberative architecture:** symbolic world model that enables problem-solving components to be built using a sense-plan-act paradigm
- **hybrid architecture:** system made up of reactive and deliberative components that are combined into a hierarchy of interacting layers, where each layer reasons at a different level of abstraction
- **reactive architecture:** system that is aware of changes that affect its computations and adjusts accordingly

NOTE: The adjustment is made by reacting to an event in real-time without centralized control. The availability of new information drives program logic execution.

- **software architecture:** high-level structure and organization of a software-based system. this includes the objects, their properties and methods, and relationships between objects

assisted system: system that the ENI system is providing recommendations and/or management commands to is referred to as the "assisted system"

cognition: process of understanding data and information and producing new data, information, and knowledge

context: collection of measured and inferred knowledge that describe the environment in which an entity exists or has existed

decision making: set of processes that result in the selection of a set of actions to take from among several alternative possible actions

designated entity: operator, nms, ems, controller, or orchestrator acting on behalf of the assisted system

NOTE: The Designated Entity is a trusted entity [4].

design pattern: general, reusable solution in a given context to a commonly occurring software problem

NOTE: This type of design pattern is not an architecture and not even a finished design; rather, it describes how to build the elements of a solution that commonly occurs. It may be thought of as a reusable template.

- **design pattern, architecture:** general, reusable solution in a given context to a commonly occurring problem in the design of the software architecture of a system
- **design pattern, software:** general, reusable solution in a given context to a commonly occurring problem in the design of a software system

formal: study of (typically linguistic) meaning of an object by constructing formal mathematical models of that object and its attributes and relationships

knowledge: analysis of data and information, resulting in an understanding of what the data and information mean

NOTE: Knowledge represents a set of patterns that are used to explain, as well as predict, what has happened, is happening, or is possible to happen in the future; it is based on acquisition of data, information, and skills through experience and education.

- **inferred knowledge:** knowledge that was created based on reasoning, using evidence provided
- **measured knowledge:** knowledge that has resulted from the analysis of data and information that was measured or reported
- **propositional knowledge:** knowledge of a proposition, along with a set of conditions that are individually necessary and jointly sufficient to prove (or disprove) the proposition

location: physical geographic location (e.g. a geocode or a bounding polygon) of an entity (e.g. a server)

NOTE: Contrast this with Placement.

Model-Driven Behaviour (MDB): approach in which the behaviour of components, modules of systems are managed using MDE. For ENI, this applies to Functional Blocks, not components within a Functional Block

Model-Driven Engineering (MDE): approach in which models are central to all phases of the development and implementation processes

negotiation: set of communications that is intended to reach a beneficial outcome for a set of conflicting issues

- **distributive negotiation:** zero-sum game, in which each participant assumes that there is a fixed amount of value to be divided between the (winning) bidders
- **integrative negotiation:** win-win (or non-zero-sum) game, in which all collaborating participants receive optimal value

placement: logical placement of an entity (e.g. a virtual machine) on or in another entity (e.g. a server).

NOTE: Contrast this with Location.

repository: centralized location of a set of storage devices that enable different functional blocks to store and retrieve information

- **active repository:** repository that pre- and/or post-processes information that is stored or retrieved

NOTE: It may contain dedicated (typically internal) Reference Points that provide the loading, activation, deactivation, and unloading of specialized functions that change the pre- and/or post-processing functionality according to the needs of the application.

- **passive repository:** repository that stores or retrieves information without pre- or post-processing

SDO system: part of an Assisted System that is defined by another SDO

NOTE: Examples include NFV MANO and MEF LSO.

semantics: study of the meaning of something (e.g. a sentence or a relationship in a model)

telemetry: process of recording and transmitting data to receiving equipment for monitoring purposes

NOTE: The process is typically automated, and the data transfer may include wireless, cellular, optical, and other mechanisms.

3.2 Symbols

Void.

3.3 Abbreviations

Void.

4 Overview of System Architecture (informative)

4.1 Introduction

This clause provides an informative introduction to the ENI System Architecture. Clauses 4.2 and 4.3 describe the motivation and benefits of ENI. Clause 4.4 provides a high-level description of the ENI System Architecture, including how it interacts with the Assisted System (and/or its Designated Entity), different types of Assisted Systems, the mode of operation that the Assisted System (or its Designated Entity) can choose, direct and indirect communication between the ENI System and the Assisted System (or its Designated Entity), and important concepts used in this System Architecture. Clause 4.5 describes the functional architecture of the ENI System in terms of Functional Blocks. It ends with a discussion of how decision-making is done in the ENI System.

NOTE: See clause 9 for further applicable future work.

4.2 Motivation for ENI

Current network management provisioning and monitoring functions are time-consuming and error-prone. This is due to the proliferation of different technologies, as well as different implementations from different vendors. In addition, users are demanding more complex services (e.g. context-aware, personalized services). Hence, operators are concerned about the increasing complexity of integration of different platforms in their network and operational environment. These human-machine interaction challenges increase the time to market of innovative and advanced services. Moreover, there is no efficient and extensible standards-based mechanism to provide contextually-aware services (e.g. services that adapt to changes in user needs, business goals, or environmental conditions).

These and other factors contribute to a very high OPerational EXpenditure (OPEX) for network operation and management. Operators need to optimize the use of networked resources (e.g. through the automation of their network configuration and monitoring processes to reduce this OPEX). More importantly, operators need to improve the use and maintenance of their networks.

Indeed, operators are concerned about decreasing the number of the operator's personnel dedicated to managing the infrastructure (e.g. how many people are required to control a given number of network devices, or particular structures, such as SD-WANs). This requires improved automation and real-time control loops. Operators are also keenly interested in improving infrastructure efficiencies. For example, it is likely that 5G and IoT are expected to have complex interactions between different sets of network devices. Concurrently, user needs, business goals, and the environment frequently change. Thus, network intelligence is needed to detect these contextual changes, determine which groups of devices and services affect each other, and manage the resulting services while maintaining Service Level Agreements (SLAs).

To accomplish these targets, the associated challenges are stated as:

a) automating complex human-dependent decision-making processes (e.g. managing and optimizing network and system configuration processes) by providing system and network intelligence tools and services;

b) determining which services are offered, and which services are in danger of not meeting their Service-Level Agreement (SLA)s, as a function of changing context;

c) defining how best to visualize how network services are provided and managed to improve network maintenance and operation;

d) providing an experiential architecture (i.e. an architecture that uses Artificial Intelligence (AI) and other mechanisms to improve its understanding of the environment, and hence the operator experience, over time);

e) improving operator's personnel efficiency through improved management and automation, while providing increased visibility and a simplified interface between the operator and the networks and networked applications; this reduces errors and makes human-directed commands more efficient and intuitive; and

f) improving infrastructure utilization and agility (response to real-time changes) while maintaining SLAs. The deployed networks and systems likely need to be aware of the needs of Services and Applications, and handle environmental changes in an automated way without human involvement.

4.3 Benefits of ENI

ENI defines a Functional Block architecture that helps address the six challenges of clause 4.2. It specifies Functional Blocks and Reference Points for providing a model-based, policy-driven, context-aware system that provides recommendations and/or commands to Assisted Systems. This communication is to be done directly or indirectly via a Designated Entity acting on the behalf of the Assisted System. A Designated Entity is an NMS, EMS, controller, or current or future management and orchestration systems. ENI is expected to enable the Assisted System to perform more accurate and efficient decision making.

ENI provides the following important benefits:

1) to measure and quantify the operation and performance of the resources and services of an operator;

2) to enable personalized services to be provided to customers;

3) to learn from the operation of the network, and decisions made by the operator;

4) to automate the operator's complex human-dependent decision-making processes by translating changing user needs, business goals, and environmental conditions into closed-loop configuration and monitoring;

5) to enable the optimization and adjustment of resources and services managed by the operator, as well as associated tools and applications needed by the operator to conduct business.

The ENI System is based on an experiential architecture. This means that it learns through experience (i.e. through the operation of the systems that it assists in governing). This self-learning principle is key to improving operator experience, and enables the system to evolve over time from proposing to implementing decisions.

The ENI architecture enables the operator to adjust the offering of services in response to contextual changes. It is expected that ENI enables the deployment, administration, and control of migration toward new technologies, such as SDN and NFV.

4.4 High-Level Description of the ENI System Architecture

4.4.1 Overall Description

Carrier and Enterprise network architectures are becoming increasingly complex and diverse. For example, it is possible to split a single application into separate distributed entities. As another example, managers, controllers, and orchestrators are also being distributed, making their interaction critical. Furthermore, the network and applications use new technologies and are programmed in diverse languages. Therefore, it is becoming more and more difficult to realize potential and significant reductions in CapEx and OpEx. A primary goal of ENI is to provide a robust, distributed platform that uses modelling, policy management, and AI to solve these problems.

The ENI System is an innovative, policy-based, model-driven functional entity that improves operator experience. In addition to network automation, the ENI System assists decision-making of humans as well as machines, to enable a more maintainable and reliable system that provides context-aware services that more efficiently meet the needs of the business. For example, the ENI System enables the network to change its behaviour (e.g. the set of services offered) in accordance with changes in context, including business goals, environmental conditions, and the varying needs of end-users.

This is achieved by using policy-driven closed control loops that use emerging technologies, such as big data analysis, analytics, and artificial intelligence mechanisms, to adjust the configuration and monitoring of networks and networked applications. It dynamically updates its acquired knowledge to understand the environment, including the needs of end-users and the goals of the operator, by learning from actions taken under its direction as well as those from other machines and humans (i.e. it is an experiential architecture). It also ensures that automated decisions taken by the ENI System are correct and increase the reliability and stability, and lower the maintenance required, of the network and the applications that it supports. It improves and simplifies the management of network services through their visualization, and enables the discovery of otherwise hidden trends and interdependencies.

Finally, it helps determine which services are appropriate to be offered for a specific context, and which services are in danger of not meeting their Service-Level Agreement (SLA), as a function of changing context. In order to achieve the latter and to assist in monitoring and improving infrastructure efficiencies, it draws knowledge from telemetry shared with it.

It is possible to use the ENI System to advise on balancing the resources or availability of the network elements or function used to provide the services.

NOTE: The goal of this work, at this stage in the industry evolution of AI/ML, is to ensure proper management mechanisms are present in any control loop used within ENI. This enables ENI or a human operator to manage the degree of autonomicity used within the control loop. This applies to passive and active recommendations and commands (e.g. monitoring information and sending reconfiguration commands), which in turn ensures the prevention of undesired outputs.

4.4.2 The Assisted System

4.4.2.1 Introduction

The system that the ENI System is providing recommendations and/or management commands to is referred to as the "Assisted System". Within the current scope of ENI, there are three classes of Assisted Systems. In each case, ENI requires data from the Assisted System. Changes to the Assisted System are not required for any class of Assisted System in order to facilitate the use and rapid adoption of ENI.

ENI typically uses an API Broker to mediate between the ENI System and the Assisted System (regardless of its class). This is because the ENI System and the Assisted System typically have different APIs. The API Broker defines the correct way for one system to request services from the other system without requiring ENI to understand the details of every API of every Assisted System that wants to communicate with it. All communication is done using external Reference Points.

NOTE: APIs, and the API Broker, are for further study in Release 2 of the present document (see clause 9). The Assisted System and/or its Designated Entity do not have to accept the recommendations offered by ENI when in recommendation mode. The Assisted System and/or its Designated Entity accepts the recommendations offered by ENI when in management mode. This includes those decisions that apply to recommendations and commands, respectively, when the Assisted System is in a mixed mode of operation. These different operational modes are defined in clause 4.4.3, and specified as [MOP15] and [MOP16] in clause 5.4.

Clause 4.4.2.5 compares the three classes of Assisted Systems, and summarizes their common characteristics and differences.

4.4.2.2 Class 1: An Assisted System that has No AI-based Capabilities

This type of Assisted System has no AI-based decision-making capabilities. Two options are shown in Figure 4-1.

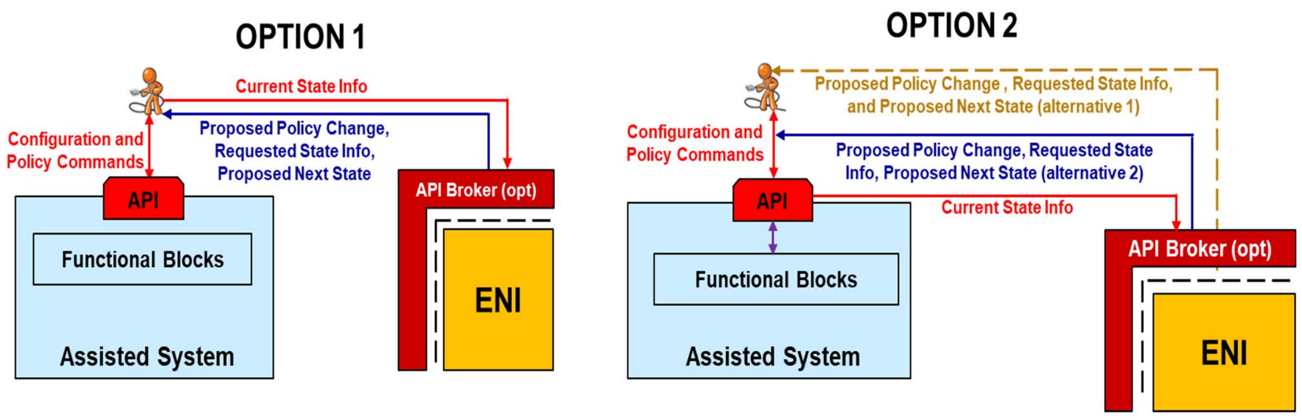

OPTION 1
Current State Info
Configuration and Policy Commands
Proposed Policy Change, Requested State Info, Proposed Next State
API
API Broker (opt)
Functional Blocks
Assisted System
ENI
OPTION 2
Proposed Policy Change , Requested State Info, and Proposed Next State (alternative 1)
Configuration and Policy Commands
Proposed Policy Change, Requested State Info, Proposed Next State (alternative 2)
API
Current State Info
API Broker (opt)
Functional Blocks
Assisted System
ENI

Figure 4-1: Class 1: The Assisted System Has No AI Capabilities

Both variants of a Class 1 Assisted System include the optional use of an API Broker. The difference between the variants is that the first does not expose an API, whereas the second does.

The variant labelled "Option 1" on the left represents the case where the Assisted System is not able to provide an API that is usable by ENI. Hence, in this option, ENI communicates directly with the Designated Entity of the Assisted System (i.e. the operator, an EMS, an NMS, a controller, or an orchestrator). The communication types shown in Figure 4-1 (e.g. "current state info") are exemplary. All communication is defined by the capabilities of the ENI API; hence, additional communication types not shown in Figure 4-1 are also possible.

In contrast, in the variant labelled "Option 2" the Assisted System exposes an API. The communication types (e.g. "current state info") are also exemplary, which means that all communication is defined by the capabilities of the ENI API; hence, additional communication types not shown in Figure 4-1 are also possible. There are two alternatives for communication between ENI and the Assisted System; these are communication via their APIs (shown in the solid blue line) and from the API to the Operator or Designated Entity of the Assisted System (shown in the brown dashed line). Both communication alternatives are able to be used interchangeably. For example, the API of the Assisted System (e.g. the solid blue line) is used to implement some policy changes, while other policy changes are implemented by communicating directly to the operator or Assisted System (e.g. the dashed brown line).

In Option 2, ENI communicates using the APIs, so an API Broker is probably needed in order to accommodate the potential difference between ENI's native APIs and those exposed by the Assisted System. For illustration purposes, that API translation is performed by the API Broker between a standard API provided by ENI and the API of the Assisted System (or its Designated Entity) that is communicating with ENI. This translation is external to the ENI System, because each Assisted System (or Designated Entity) typically has its own unique API. It is desirable to insulate ENI from the specific details of the Assisted System, so that it does not have to change with every change of APIs of the Assisted System. Thus, an API Broker is proposed to function as a gateway between the constant standardized set of APIs provided by ENI and the varying set of APIs that are particular to each individual Assisted System. Alternatively, the ENI System communicates with the Operator or Designated Entity of the Assisted System using the API Broker.

This class of Assisted System has no AI capabilities, so ENI operates as an external system that communicates with the Assisted System through standard Reference Points and APIs defined by ENI. In addition, most Assisted Systems of this class were not designed to accommodate new internal modules, so ENI needs to be incorporated as a discrete external system. The Assisted System and its Designated Entity are likely not able to use the outputs of the ENI System to their full potential. The ENI System is not involved in making decisions in the real-time control loop of the Assisted System. The ENI System sends recommendations and/or management commands to improve decisions that do not involve real-time configuration (e.g. change some statically defined parameters to improve resource utilization). The burden is on ENI (and the API Broker, in the case of APIs) to present its results in a form that the Assisted System is able to understand. Hence, nothing has to change in the Assisted System in order for ENI to be used.

4.4.2.3 Class 2: An Assisted System with AI that is Not in the Control Loop

This type of Assisted System has some AI-based decision-making capabilities. Two options are also possible as shown in Figure 4-2.

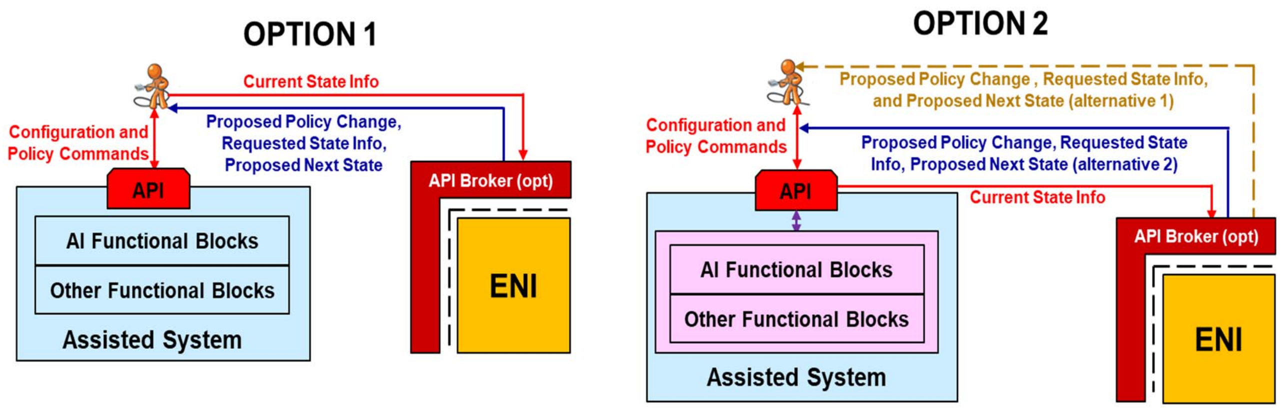

OPTION 1
Current State Info
Configuration and Policy Commands
Proposed Policy Change, Requested State Info, Proposed Next State
API
API Broker (opt)
AI Functional Blocks
Other Functional Blocks
Assisted System
ENI
OPTION 2
Proposed Policy Change , Requested State Info, and Proposed Next State (alternative 1)
Configuration and Policy Commands
Proposed Policy Change, Requested State Info, Proposed Next State (alternative 2)
API
Current State Info
API Broker (opt)
AI Functional Blocks
Other Functional Blocks
Assisted System
ENI

Figure 4-2: Class 2: Assisted System with AI, but Not in the Control Loop

Both variants of a Class 2 Assisted System include the optional use of an API Broker. The difference between the variants is that the first does not expose an API, whereas the second does.

This class of Assisted System uses some AI-based algorithms, but critically, does not use AI in its internal control loop. For example, optimization of the physical server location, virtual machine, and/or Container placement on a server, as well as the geographic location of the server, is likely performed using an AI-algorithm by this class of Assisted System. However, for this class of Assisted System, the ENI System is not involved in making decisions in the real-time control loop of the Assisted System. The ENI System sends recommendations and/or management commands to improve decisions that do not involve real-time configuration (e.g. as indicated above, change some statically defined parameters to improve resource utilization). This class of system works as an extension of class 1.

The architectural details of the Assisted System for class 2 systems are not visible to ENI. The two types of Functional Blocks shown in Figure 4-2 (i.e. "AI Functional Blocks" and "Other Functional Blocks") are meant to illustrate what has been mentioned in the last paragraph.

There are two alternatives for communication between ENI and the Assisted System. For Option 1, ENI communicates, using its API Broker, with the Operator or Designated Entity of the Assisted System. Data ingestion by ENI (e.g. "Current State Info") is performed via the API Broker. Again, as for Class 1, in Option 2 of Class 2, ENI preferentially communicates with the APIs of the Assisted System; however, it is also possible to communicate via the Operator or Designated Entity of the Assisted System (as in Option 1) using its API Broker. Both communication alternatives are able to be used interchangeably. For example, the API of the Assisted System (e.g. the solid blue line) is used for some policy changes, while others use direct communication to the operator or Assisted System (e.g. the dashed brown line).

Again, as for Class 1, a Class 2 Assisted System is not designed to directly accommodate the features provided by ENI. Hence, ENI operates as an external system that communicates with the Assisted System and/or its Designated Entity, as shown in Figure 4-2. The burden is on ENI (and the API Broker, in the case of APIs) to present its results in a form that the Assisted System and its Designated Entity are able to understand. Hence, nothing has to change in the Assisted System, or its Designated Entity, in order for ENI to be used.

4.4.2.4 Class 3: An Assisted System with AI Capabilities in its Control Loop

4.4.2.4.1 Introduction

This type of Assisted System has AI-based decision-making capabilities as part of its internal control loop. Two options are shown in Figure 4-3.

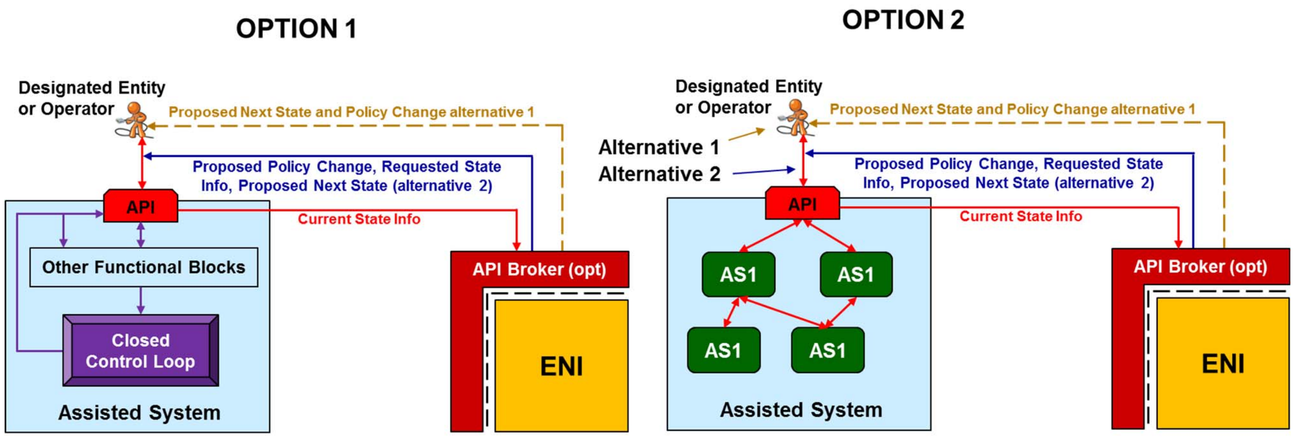

OPTION 1
Designated Entity or Operator
Proposed Next State and Policy Change alternative 1
Proposed Policy Change, Requested State Info, Proposed Next State (alternative 2)
API
Current State Info
Other Functional Blocks
Closed Control Loop
Assisted System
API Broker (opt)
ENI
OPTION 2
Designated Entity or Operator
Proposed Next State and Policy Change alternative 1
Alternative 1
Alternative 2
Proposed Policy Change, Requested State Info, Proposed Next State (alternative 2)
API
Current State Info
AS1
AS1
AS1
AS1
Assisted System
API Broker (opt)
ENI

Figure 4-3: Class 3: Assisted System with AI in the Control Loop

This class of Assisted System optionally has other AI capabilities. The ENI System is involved in making decisions for any function performed by the Assisted System, as long as it has permission to do so; significantly, this includes real-time decisions. This is a fundamental difference between this class of Assisted System and those of Classes 1 and 2.

4.4.2.4.2 Class 3 Options

4.4.2.4.2.1 Introduction

There are two different options for using ENI with the control loop(s) of class 3 Assisted Systems: (1) ENI is defined as a modular addition that is able to directly affect the control loop(s) of the Assisted System, or (2) ENI is defined as a module that is attached to the control loop(s) of the Assisted System. The difference is that the first option treats ENI as a logically integrated module of the Assisted System, while the second option distinguishes between Assisted Systems that communicate directly with ENI vs. those that do not communicate directly with ENI. In this second alternative, Assisted Systems that do not communicate directly with ENI are only allowed to only receive ENI recommendations and commands through an Assisted System that does communicate directly with ENI;, this implies the use of peer-to-peer Reference Points.

4.4.2.4.2.2 Communication Alternatives that are Common to Both Class 3 Options

There are two common alternatives for communication between ENI and the Assisted System in each of the above options. In alternative 1, ENI communicates, using its API Broker, with the Operator or Designated Entity of the Assisted System. Data ingestion by ENI (e.g. "Current State Info") is performed via the API Broker. This is shown in Figure 4-3 by using the brown dashed line. Again, as for Classes 1 and 2, in alternative 2, ENI preferentially communicates with the APIs of the Assisted System. Both communication alternatives are able to be used interchangeably. For example, the API of the Assisted System (e.g. the solid blue line) is used to implement some policy changes, while other policy changes are implemented by communicating directly to the operator or Assisted System (e.g. the dashed brown line).

4.4.2.4.2.3 Option 1 Feedback

Option 1 emphasizes two key points:

- ENI is only communicating with a single Assisted System (compared to option 2).
- The Assisted System has one or more closed feedback control loops.

Since there is only a single Assisted System, ENI is concerned with managing and optimizing the performance of just a single Assisted System.

ENI is not aware of the number or type of control loops that the Assisted System has. The two typical types of control loops are shown in Option 1.

Both types of control loops are internal to the Assisted System, and the use of ENI recommendations and commands does not interfere in their operation. For informational purposes, the first type is to feedback information from the control loop to other Functional Blocks; this enables the control loop to affect the behaviour of multiple Functional Blocks (regardless of what type of task each Functional Block performs). The second type is to feedback information via the API. This enables Functional Blocks that are under different administrative control to participate in behavioural decisions. This theoretically enables ENI to tap into the control loop of the Assisted System. However, this is not appropriate for some real-time decisions due to the communication and security constraints of using an API. For example, a delay in communication caused by using the API (versus, for example, remote method invocation or other types of communication), as well as additional time required to secure the communication between two entities that are not co-located, is likely not going to be acceptable in certain scenarios. In addition, Assisted Systems often need to add different types of descriptive and/or prescriptive information to data and information communicated using the API so that ENI is able to provide preferential processing; this is for further study in Release 2 of the present document (see clause 9).

4.4.2.4.2.4 Option 2 Communication

Option 2 emphasizes two key points:

- ENI is communicating with multiple Assisted Systems (compared to option 1).
- ENI does not have to be aware that it is communicating with all some (versus all) participating Assisted Systems.

ENI registers or otherwise logs that it is communicating with specific Assisted Systems, as long as it is either communicating with them directly or is told that an Assisted System is communicating with other Assisted Systems on its behalf. The structure and composition of information, policies, recommendations, and commands exchanged between ENI and the Assisted System(s) are likely affected, but the protocols and communication mechanisms used by ENI are not affected.

4.4.2.4.2.5 Working as an External Discrete System

In both Class 3 alternatives, ENI also works as its own discrete system external to the Assisted System, as shown in Figure 4-3. However, most Assisted Systems of this class were not designed to accommodate new internal modules, so ENI needs to be incorporated as a discrete external system. The burden is on ENI (and the API Broker, in the case of APIs) to present its results in a form that the Assisted System (or its Designated Entity) are able to understand. Hence, nothing has to change in the Assisted System, or in its Designated Entity, in order for ENI to be used.

4.4.2.5 Summary of Interaction between the Assisted System and ENI

In each of the above classes of Assisted System, the ENI System interacts with either the Assisted System directly, or with the Designated Entity of the Assisted System. The Assisted System does not need to know if it is communicating with the ENI System or with the operator, an EMS, an NMS, or a controller. The point is to reduce the barrier of adoption for using an ENI System by not requiring new APIs from the Assisted System to interact with the ENI System. The use of an API Broker insulates the ENI System from having to know what entity it is specifically communicating with.

Common Features are shown in Table 4-1.

Table 4-1: Common Characteristics and Behaviour of Assisted Systems

Feature	Class 1 Assisted System	Class 2 Assisted System	Class 3 Assisted System
Assisted System Need to Change to Use ENI?	NO	NO	NO
AI Capabilities of Assisted System	NONE	Only in non-real-time decision making	Real- or non-real-time decision making

Feature	Class 1 Assisted System	Class 2 Assisted System	Class 3 Assisted System
Scope of Assisted System APIs	None in Option 1, limited to non-AI functionality in Option 2	None in Option 1, limited to non-AI functionality in Option 2	Real- or non-real-time decision making
Use of API Broker	Needed for communication with ENI if the Assisted System does not support ENI External Reference Points and ENI APIs	Needed for communication with ENI if the Assisted System does not support ENI External Reference Points and ENI APIs	Needed for communication with ENI if the Assisted System does not support ENI External Reference Points and ENI APIs
Communication Types	Option 1: through Operator or Designated Entity; Option 2: through APIs	Option 1: through Operator or Designated Entity; Option 2: through APIs	Option 1: through Operator or Designated Entity; Option 2: through APIs
ENI Operation as an External System to Assisted System	YES; embedding is not an option	YES; embedding is not an option	YES; embedding is not an option
Directly Affect Control Loop?	NO	NO	YES
ENI is able to interact with modules of Assisted System	NO	NO	YES, as defined by the Assisted System

All three classes of Assisted System are applicable to Release 1. It is likely that Class 3 system communication evolves in future releases; this is for further study in Release 2 of the present document (see clause 9).

4.4.3 Mode of Operation

4.4.3.1 Allowed Modes of Operation

The ENI System operates in two different modes, called "recommendation mode" and "management mode".

The operation of the ENI System in recommendation mode is analogous to that of a Recommender system - it provides recommendations to the Assisted System. In contrast, when the ENI System is operating in management mode, the ENI System provides decisions and commands to be implemented by the Assisted System. These decisions and commands are subject to the approval of the Assisted System (or its Designated Entity).

The ENI System supports a recommendation mode and optionally supports a management mode of operation. A mixed mode of operation (i.e. the combination of recommendation and management modes) for different sets of decisions is also optionally supported. Both modes may be used with the same Assisted System. For example, if the Designated Entity of the Assisted System decides that recommendations belonging to a particular category have been verified, it then changes decisions for that category to operate in management mode; this is done independently of other categories, so that other operations stay in recommendation mode.

4.4.3.2 Setting the Mode of Operation

The ENI System initially sets the mode of operation. In order to do that, the ENI System selects its mode of operation based on the capabilities of the Assisted System (see clause 4.4.4.2 describing how these capabilities are acquired/discovered), along with other applicable information (e.g. regulatory policies, status of the infrastructure, and goals input by the operator) in a given context or situation (see clauses 4.4.5.3 and 4.4.5.4). All standardized communication between the ENI System and the Assisted System is through Reference Points defined in the present document (see clause 4.4.5.4).

The ENI System optionally has a default mode of operation that is used when:

1) the ENI System is initialized; and/or

2) a class of decisions is reached that is not specified by the Assisted System.

For example, suppose the user has specified how to handle best effort traffic, but not how to handle prioritized service class traffic. In this example, if the ENI System has defined defaults that are appropriate to each service class, then best effort traffic is handled as specified by the user, while prioritized traffic is handled by the default specified by the ENI System. The default mode of operation is optionally changed as desired by either the Assisted System or its Designated Entity.

4.4.3.3 Interaction with the Assisted System

All communication regarding mode of operation change(s) is done using a request-reply interaction paradigm over an appropriate External Reference Point defined in the present document. This paradigm applies to any entity that requests a change (e.g. the User, the Designated Entity of the Assisted System, or an agent ([i.8] and [i.9]) acting on behalf of these).

In any mode of operation, the ENI System interacts with the data, control, management, and/or orchestration planes of the Assisted System, underlying infrastructure, and network. That interaction requires the Assisted System to expose new data for analysis by the ENI System; this is communicated as requests to change the configuration of the affected components, devices, and/or systems.

4.4.3.4 Selecting a Mode of Operation for a Class of Decisions

The Designated Entity of the Assisted System optionally selects a desired mode of operation for a class of decisions to the ENI System.

Alternatively, since the ENI System is an intelligent system, the ENI System optionally suggests that a particular mode of operation is used for a set of decisions that the Assisted System (or the Designated Entity on its behalf) has not specified or envisaged. In this case, the Assisted System needs to approve that it accepts ENI recommendations and/or management commands, as appropriate. These decision classes may be described by a set of capabilities (e.g. metadata) advertised by ENI.

4.4.3.5 Communication of Mode of Operation

Communication is done by the Designated Entity of the Assisted System, which by definition is a trusted entity [4]. This means that the ENI System and the Designated Entity of the Assisted System collectively operate as expected, according to design and policy, despite environmental disruption, human user and operator errors, and attacks by hostile parties. In addition, in either mode of operation, this behaviour is validated by each entity before appliance.

4.4.3.6 Normal Operation of the Selected Mode of Operation

The default configuration of the ENI System is to start in recommendation mode when first booted. This mode of operation is defined for all types of decisions, since ENI has not been able to learn about its environment, services and resources that it is managing, as well as the preferences of the operator.

After booting, the ENI System then queries the Designated Entity of the Assisted System to determine the set of capabilities the Assisted System has, which Reference Points are defined, and what additional information and requirements (e.g. regulatory policies) are appropriate to be used. Given these initial inputs, the ENI System then asks the Designated Entity of the Assisted System if there are any types of decisions that it would like assistance with:

- If not, the ENI System is limited to passively learning through observing the actions of the Designated Entity of the Assisted System, and their effect on the infrastructure, as shared with ENI through information provided on the external Reference Points of ENI.
- If so, then the ENI System then asks which mode of operation is desired for each class of decision.

The Designated Entity of the Assisted System optionally defines a preferred mode for the ENI System to operate in for a given class of decisions. ENI acknowledges each such request, and also confirms when the requested mode of operation is ready to be used for each class of decisions.

Operation then continues with the ENI System learning through observation, except that the ENI System provides recommendations and/or commands based on the selected mode of operation.

If the ENI System requires data that is not being monitored, it asks the Designated Entity of the Assisted System to change its configuration to report those data. The configuration change is required to enable data that are not currently being monitored to be made available to the ENI System.

4.4.3.7 Exception Handling for the Selected Mode of Operation

An exception is raised any time that:

1) a request or a response from either the Assisted System or ENI is not delivered; or

2) ENI is not able to change the mode of operation as requested.

4.4.4 Communication

4.4.4.1 Overview

All standardized communication between the ENI System and the Assisted System (or its Designated Entity) uses External Reference Points, which are defined in the present document. The ENI System is defined as a set of Functional Blocks (see clauses 4.4.5.2 and 6.3). Functional Blocks define a recursive mechanism to represent functionality with well-defined inputs and outputs; if these inputs and/or outputs are meant to be interoperable, then they are defined as Reference Points. An External Reference Point (see clause 4.4.6.1) is a Reference Point between an ENI System Functional Block and an external system, whereas an Internal Reference Point (see clause 4.4.6.2) is a Reference Point between different ENI System Functional Blocks.

NOTE: Internal Reference Points are for further study in Release 2 of the present document (see clause 9).

Communication is not limited to a single mechanism: direct (e.g. point-to-point) and/or indirect (e.g. systems connected by a bus) communication are used.

Communication is performed between two or more trusted entities, and forms a trustworthy system (see [MOP.11]).

In general, the ENI System acknowledges all commands given to it by the Assisted System or its Designated Entity. Note that only the Designated Entity behaves as a trusted entity. The ENI System sends notifications of other important events, such as completing the change to a new mode of operation, to the Designated Entity.

4.4.4.2 Discovery

A basic assumption is that when a device, application, or system starts up, it has no information about any peer devices, the network structure, or what specific set of roles [i.1], [i.4] and [i.6] it is assigned to play. This means that discovery needs to be repeated as often as necessary in order to find peers that support each function required. The discovery process also needs to be flexible enough to accommodate different topologies. The ENI System discovers (directly or utilizing the Designated Entity) the capabilities of the Assisted System. These include support for the desired mode of operation and External Reference Points, current and future situation and context of the Assisted System, underlying infrastructure and network, and appropriate regulatory policies and operator goals. For example, the ENI System requires data from the Assisted System in order for the ENI System to supply recommendations and commands. Thus, the ENI System requires knowledge of the types of data available from the Assisted System.

4.4.4.3 Direct Configuration

ENI Functional Blocks optionally support direct configuration of control and management parameters through sequences of messages.

EXAMPLE: Routing protocols use a model based on distributed devices that communicate repeatedly with each other. Current routing protocols mainly consider simple data, such as link status. This is a form of information synchronization between peers.

4.4.4.4 Negotiation

ENI Functional Blocks optionally support negotiation of control and management parameters. Negotiation is fundamental to agent interaction, as it enables interaction between entities that have different functions, and helps entities arrive at a mutually agreed upon compromise in functionality.

A negotiation process achieves agreement between two or more entities in a system. In the context of ENI, digital agents of each entity communicate with each other to reach an agreement on the use of services and/or resources; this optionally includes a set of rules governing the use of each.

ENI is an experiential system, which constantly learns while it is operating. This means that if the context in which ENI is operating changes the behaviour and functionality required by ENI Functional Blocks typically also changes dynamically.

Therefore, static configuration of the behaviour of ENI Functional Blocks is not desirable, since the provided functionality cannot change in response to changing context. Negotiated behaviour enables each negotiating entity to customize the number and priority of parameters being negotiated dynamically at runtime. More importantly, it ensures that the collaborating entities arrive at a mutually agreed-upon solution. For example, suppose that ENI has a network slice management system to deliver sets of services to different customers from the same networked infrastructure. It is possible to use AI to dynamically analyse current operational performance, as well as trends in that performance, and recommend or change network resources and services to adapt to changing user needs, business goals, and environmental changes. In this example, the benefit of negotiation is efficiency. For example, if the Functional Blocks that are supplying the changed behaviour have multiple programmable parameters, negotiating a new service is likely to be more effectively done by asking for a desired overall service and using negotiation to fine-tune applicable parameters. This also enables the negotiation to optionally include additional compensation, such as credit issued. See clause 6.2.3 for additional details.

The negotiation process is a request-response message exchange pattern that is guaranteed to terminate with success or failure. More specifically, negotiation is a process by which two or more entities iteratively interact to agree on parameter settings that best satisfy the objectives of all participants.

Robust negotiation processes include loop prevention, time-outs, and generic tie-breaking rules for each parameter that is being negotiated. In addition, if negotiation is used in ENI, then the following additional communication requirements are typical:

- discover each other;
- synchronize state with each other;
- agree (negotiate) on parameters and resources to use directly with each other.

If negotiation is not used, or if it is used but is not successful, architectural requirements [MOP.11] and [MOP.12] apply.

4.4.4.5 Switching the Mode of Operation

4.4.4.5.1 Overview

The ENI System notifies the Assisted System (or its Designated Entity) when it wants to change the mode of operation for a particular decision. It does so by sending a request to the Assisted System (or its Designated Entity) using an appropriate External Reference Point. These two options are described in clauses 4.4.4.5.2 and 4.4.4.5.3, respectively. If this mode of operation is not supported by the Assisted System for this decision, then the Assisted System (or its Designated Entity) informs the ENI System that this is not allowed. If this operation is supported by the Assisted System, then the Assisted System analyses the proposed change to determine if it is acceptable. If it is not, then optionally, negotiation is then initiated by either the Assisted System or the ENI System. Once agreed, the Assisted System (or its Designated Entity) informs the ENI System that the change is allowed. The ENI System operates the same way in either mode of operation; the difference is the set of outputs (information, recommendations, and commands) generated by the ENI System that are sent to the Assisted System.

In both modes of operation, the ENI System interacts with the data, control, management, and/or orchestration planes of the Assisted System. That interaction optionally requires the Assisted System to expose new interfaces, or change some existing interfaces, in order to adhere to the ENI Reference Points.

All standardized communication with ENI is done using the External Reference Points specified in the present document. If an Assisted System is not capable of supporting some or all of these ENI External Reference Points, a possible solution is to use an API Broker to mediate between ENI and the Assisted System (or its Designated Entity) as described above.

A precondition of the entity requesting a change in the mode of operation is that it is a *trusted* entity (see [MOP.11 and MOP.12]). The Designated Entity of the Assisted System chooses to either trust all decisions to be made, or to only trust decisions for a particular set of functions belonging to a particular category.

Ultimately, the Designated Entity of the Assisted System has the final decision regarding what mode of operation is used for any (and all) decisions. Conceptually, there is no difference between switching from recommendation to management mode or vice-versa. There are two cases to be considered, depending on which entity the ENI System communicates with, as detailed below.

4.4.4.5.2 Case 1: ENI Indirectly Instructs the Assisted System to Switch Modes

This case assumes that the Assisted System is not able to decide on its own whether it is able to switch modes, and always defers this decision to its Designated Entity. Hence, the ENI System communicates directly with the Designated Entity of the Assisted System, which in turn instructs the Assisted System to switch modes.

4.4.4.5.3 Case 2: ENI Directly Instructs the Assisted System to Switch Modes

This case assumes that the Assisted System decides on its own whether it is able to switch modes. Hence, the ENI System communicates directly with its Designated Entity; either the Assisted System or its Designated Entity informs its manager (either network manager or orchestrator) that it has switched modes.

4.4.5 Functional Concepts

4.4.5.1 Introduction

This clause provides definitions of important concepts used in the ENI functional architecture.

4.4.5.2 Functional Block Definitions

A Functional Block is an abstraction of the characteristics and behaviour of a portion of a system. A Functional Block is completely self-contained; this means that it is free of side-effects. This further implies that a Functional Block does not rely on global variables, I/O, or system-wide communication paths.

A Functional Block is a modular unit that describes a portion of a system. It defines a collection of features to describe an element of interest. The specific kinds of Functional Blocks, the kinds of connections between them, and the way these elements combine to define the total system is defined according to the goals of a particular system model. Functional Blocks interact by one or more means, such as software operations, discrete state transitions, flows of inputs and outputs, or continuous interactions.

If the inputs and outputs of a Functional Block are externally visible, they are defined as External Reference Points (see clause 4.4.6.2).

A Functional Block optionally contains other Functional Blocks; in addition, it communicates with and asks services of different Functional Blocks (see clause 6.2.2). Functional Block diagrams are optionally used to pictorially represent the flow of control between Functional Blocks. SysML [1] is proposed for specifying Functional Blocks, as it is able to unambiguously represent different types of control flow; SysML is able to precisely specify the semantics and behaviour of the Functional Block.

4.4.5.3 State

The efficacy of the ENI System depends on the amount of state information available to it. In order for the ENI System to make a decision based on end-to-end goals, it is proposed that the ENI System possesses knowledge of the current state and the current goals of the systems that it is providing recommendations and/or management decisions to. Hence, state information is focused on the state of the Assisted System and its environment, but optionally also includes the state of affected Functional Blocks in the ENI System. For a large, complex system, it is unlikely that the ENI System knows the state of all elements of the Assisted System. This is because of the number of dynamically changing system elements, as well as the relatively high cost of communication. The level of compliance to ENI, as determined also by the support level for the ENI Reference Points and their proposed grouping, defined herewith, determines the functionality and optimizations provided by the ENI System. This is why the ENI System is based on a cognition framework; this framework allows the ENI System to dynamically adjust its knowledge, and infer new knowledge, as required. Such knowledge typically affects, or is reflected in, state. For example, new knowledge can direct the change of state of the Assisted System. Similarly, system operation can be associated with a current (or past) state of the Assisted System; this enables such knowledge to be optionally added to the knowledge base of the Assisted System.

The ENI System is therefore a *distributed* system, in which its Functional Blocks share information and work together, collectively, to manage the state of the Assisted System.

4.4.5.4 Control Loop Operation

In the ENI System, feedback (and feedforward) loops are comprised of outputs from the ENI System as well as signals from the Assisted System described herewith (see Figures 4-1, 4-2, and 4-3). These control loop signals play a critical role in not just stabilizing the system, but more importantly, providing mechanisms for the system to learn. For example, a simple feedback loop consists of taking past interactions with the environment and combining them with current information to guide current and future interactions.

The cognition framework Functional Block needs special types of interaction (both feedforward and feedback). In particular, it uses its set of overarching goals to guide the use of feedback. More specifically, the set of goals that guide the operation of the cognitive framework Functional Block exist *outside* of the feedback loop, and guide (at least in part) the perception and normalization of information. Together, this then provides a deeper understanding of the relevance and significance of past and current information, which in turn provides a better foundation in which to make decisions and actions.

4.4.5.5 Inferencing

Inferencing is a process, or set of processes, that produces knowledge by reasoning about evidence produced. There are three different types of inferencing:

- Deduction: deriving conclusions, through reasoning, from a set of premises (that are known or assumed to be true) to reach a (logically certain and consistent) conclusion.
- Induction: deriving conclusions, through reasoning, from a set of premises that supply some evidence for the conclusion. Critically, there is a possibility that an induced conclusion turns out to be false, even if all of its premises are true; this is not possible if the conclusion is deduced.
- Abduction: deriving the simplest and most plausible conclusion from analysing a set of observations. The conclusion is not certain.

Given a set of premises P and a conclusion C, then:

- Deduction derives C from P using formal logic (i.e. C is a logical consequence of P).
- Induction allows surmising that C follows from P, but does not ensure its validity.
- Abduction allows surmising P as an explanation of C: there is a possibility that P provides good reason to accept C, but does not guarantee its validity.

All three types of inferencing are used by ENI. Deduction and induction are typically used with logic programming systems. Inductive logic programming is often used for machine learning and probabilistic calculations. Abduction is particularly effective in fault detection and identification.

4.4.5.6 Data, Information, Knowledge, and Wisdom

4.4.5.6.1 Introduction

There are four types of knowledge that need to be managed by ENI. In general, these types of knowledge represent a progression of increased understanding and semantics (from Data to Wisdom) [i.7]. Formal semantics, which uses mathematical models, is used to help understanding of data and information that is textual in nature.

NOTE: The subjects of semantics, and especially formal semantics, are for further study in Release 2 of the present document (see clause 9).

Figure 4-4 presents a variation of the DIKW (Data-Information-Knowledge-Wisdom) model [i.7] that is used in a variety of disciplines. It has been adapted to more closely fit the operation of ENI. In this figure:

- Data corresponds to Sense
- Information corresponds to Perceive

- Knowledge corresponds to Learn
- Wisdom corresponds to Adapt

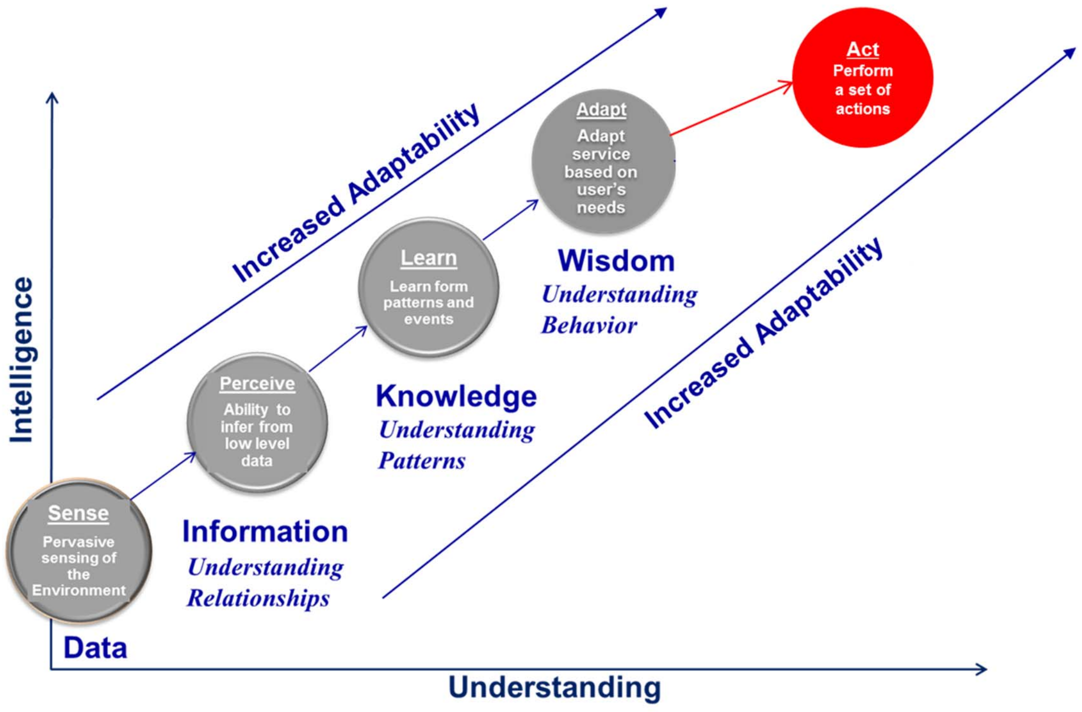


Figure 4-4: Understanding and Decision-Making

The Sense level gathers data and information from the system and environment. It is often directed to gather specific data by the management system (e.g. ENI). This level is made up of only measured knowledge (see clause 4.4.5.7). Data is thus defined as discrete unprocessed facts or observations, and therefore, have no meaning because of the lack of context and interpretation. This level is used to produce classes and class attributes.

The Perceive level uses a variety of techniques, including statistics and inferencing, to better understand the data (e.g. was the data complete, how likely was it that the data contain errors). This level also enables relationships between different concepts to be defined and discovered. This level, as well as subsequent levels, consists of measured and inferred knowledge (see clause 4.4.5.7). Information is data that is processed so that the resulting information now is relevant for a specific set of contexts or purposes. This level produces classes, class attributes, and relationships between classes.

The Learn level uses a variety of techniques to discover and learn patterns that cause data to be generated. Patterns are identified by comparing new information with existing knowledge. These patterns identify concepts, parts of concept (e.g. attributes of a class), and behaviour (e.g. methods of a class, as well as relationships between classes). Knowledge is characterized by justifiable belief, often in the form of formal mathematical proofs. This level enables a set of actions to be formulated and enacted, typically in response to a set of stimuli.

The Wisdom level applies current and new knowledge to understand why and how behaviours occurred, and supports predicting new behaviour. It is viewed as using knowledge to produce the right set of actions in a given context. This level enables plans of actions to be defined to achieve long-term goals. It also enables information and knowledge to be adapted to suit changing context, and enables existing knowledge to be corrected and augmented as necessary.

The first four levels (Sense, Perceive, Learn, and Adapt) correspond to the Observe, Orient, and Decide levels of the OODA control Loop. They culminate in a set of Actions (the Act part of OODA) that are executed [i.17], [i.1], [i.2] and [i.4]. This is discussed further in clause 5.4.

4.4.5.6.2 Data

Data consists of a sequence of symbols that are collected for a purpose (e.g. monitoring the bandwidth of a connection). Data represents a fact or statement of event without relation to other things. Data, by itself, has no meaning; it requires interpretation in a specific context to become information.. Examples of the interpretation context include the creator of the data, the time of the creation and reception of the data, and metadata.

4.4.5.6.3 Information

Information is data that have been processed in a specific context to give it meaning. Information embodies the understanding of a relationship of some sort, possibly cause and effect. Information is used to increase knowledge. More formally, information is defined as a set of data, where each element in each data sequence is well-formed (i.e. syntactically correct). Furthermore, each data sequence is meaningful (i.e. its semantics complies with the translation of data in a given context). Information provides answers to the fundamental questions "who", "what", "where", and "when".

4.4.5.6.4 Knowledge

Knowledge is the analysis of data and information, resulting in an understanding of what the data and information mean. Knowledge represents a set of patterns that are used to explain, as well as predict, what has happened, is happening, or is possible to happen in the future; it is based on acquisition of data, information, and skills through experience and education.

Knowledge is more than just collecting data and information, since merely collecting information does not allow its integration and making decisions. Knowledge is specific to the context(s) and situation(s) in which it was created.

There are many types of knowledge; however, in the context of ENI, ENI is typically concerned with what is called "propositional knowledge". More specifically, propositional knowledge is knowledge of a proposition, along with a set of conditions that are individually necessary and jointly sufficient to prove (or disprove) the propositional knowledge. This type of knowledge is implemented in formal logic systems. Knowledge answers the question "how".

Decision-making requires understanding, which includes basic cognition (i.e. the ability to use probability, as well as to synthesise new knowledge from the previously held knowledge). Knowledge enables the system to learn new information.

4.4.5.6.5 Wisdom

Wisdom is the ability to understand the current context, any regulations and restrictions (e.g. legal, ethical, temporal) that apply, and to reason and act using knowledge and experience. It uses the fundamental principles of information and knowledge to better understand the environment, its goals, and how best to achieve those goals. For example, it provides an explanation of why data, information, and knowledge occur.

Wisdom, in the specific context of ENI, is the collection of ideas and understandings that an entity possesses that are used to take effective action to achieve the goals of an entity.

Wisdom answers the question "why".

4.4.5.7 Measured vs. Inferred Knowledge

Two important concepts for ENI are the concepts of measured and inferred knowledge.

Measured knowledge is defined as knowledge that has resulted from the analysis of data and information that was measured or reported. For example, data sources could include metrics and statistics that were ingested.

In contrast, inferred knowledge is defined as knowledge that was created based on reasoning, using evidence provided. For example, observations could show that performance is decreasing; a system could then infer that an associated service level agreement is now at risk.

4.4.6 ENI Reference Points

4.4.6.1 Definition of an ENI Reference Point

An ENI Reference Point is the logical point of interaction between specific Functional Blocks. Each Reference Point defines a set of related interfaces that specify how the Functional Blocks communicate and interact with each other.

4.4.6.2 Definition of an ENI External Reference Point

An ENI External Reference Point is a Reference Point that is used to communicate between an ENI Functional Block and an external Functional Block of an external system (e.g. a Functional Block of the OSS, BSS, or Assisted System).

4.4.6.3 Definition of an ENI Internal Reference Point

An ENI Internal Reference Point is a Reference Point that is used to communicate between two or more Functional Blocks that belong to the ENI System. This communication stays within ENI, and is not be seen by systems that are external to ENI.

4.4.7 ENI Interfaces

4.4.7.1 Definition of an ENI Interface

An Interface is a point across which two or more components exchange information. An interface describes the public characteristics and behaviour that specify a contract for performing a service. In ENI, there are three types of Interfaces:

1) hardware interfaces;
2) software interfaces; and
3) application programming interfaces.

4.4.7.2 Definition of an ENI Hardware Interface

An ENI Hardware Interface is a point across which electrical, mechanical, and/or optical signals are conveyed from a sender to one or more receivers using one or more protocols. A Hardware Interface decouples the hardware implementation from other Functional Blocks in a system.

4.4.7.3 Definition of an ENI Software Interface

An ENI Software Interface defines a point through which communication with a set of resources (e.g. memory or CPU) of a set of objects is performed. This decouples the implementation of a software function from the rest of the system.

NOTE 1: Differentiation between "consumer" and "producer" is for further study in Release 2 of the present document (see clause 9). This further study includes considerations for separating data, control, and management from being communicated on the same interface. This is one of the reasons that the present document has differentiated between Reference Points that end with "-dat" and Reference Points that end with "-cmd".

NOTE 2: The interaction between Hardware and Software Interfaces, and how they are used to support API operations at a particular Reference Point, is for further study in Release 2 of the present document (see clause 9).

4.4.7.4 Definition of an ENI Application Programming Interface

An API is a set of communication mechanisms through which a developer constructs a computer program. It consists of tools, object methods, and other elements of a model and/or code. APIs simplify producing programs, since they abstract the underlying implementation and only expose the objects, and the characteristics and behaviour of those objects that are needed.

4.4.7.5 Comparison of ENI Software Interfaces with ENI APIs

An ENI Software Interface is a managed logical entity that supports communication of data, information, and/or knowledge. In contrast, an ENI API uses one or more ENI Software Interfaces to provide external access for developers to write programs to interact with ENI.

NOTE: ENI Hardware and Software Interfaces, as well as ENI APIs, are for further study in Release 2 of the present document (see clause 9).

4.5 Functional Architecture

4.5.1 Functional Block Diagram of the ENI System

A high-level Functional Block diagram that includes the use of an API Broker is shown in Figure 4-5. This is a simplified view of the main processing components of an ENI System. It is important to realize that a linear flow from input to output is not prescribed, and does not have to actually occur. This is explained in the following clauses, and elaborated further in clause 6. The arrows in Figure 4-5 represent the directionality of data and information using any of the twelve External Reference Points defined in clause 7.

NOTE 1: Two Internal Reference Points are also shown; however, these are for further study in Release 2 of the present document (see clause 9).

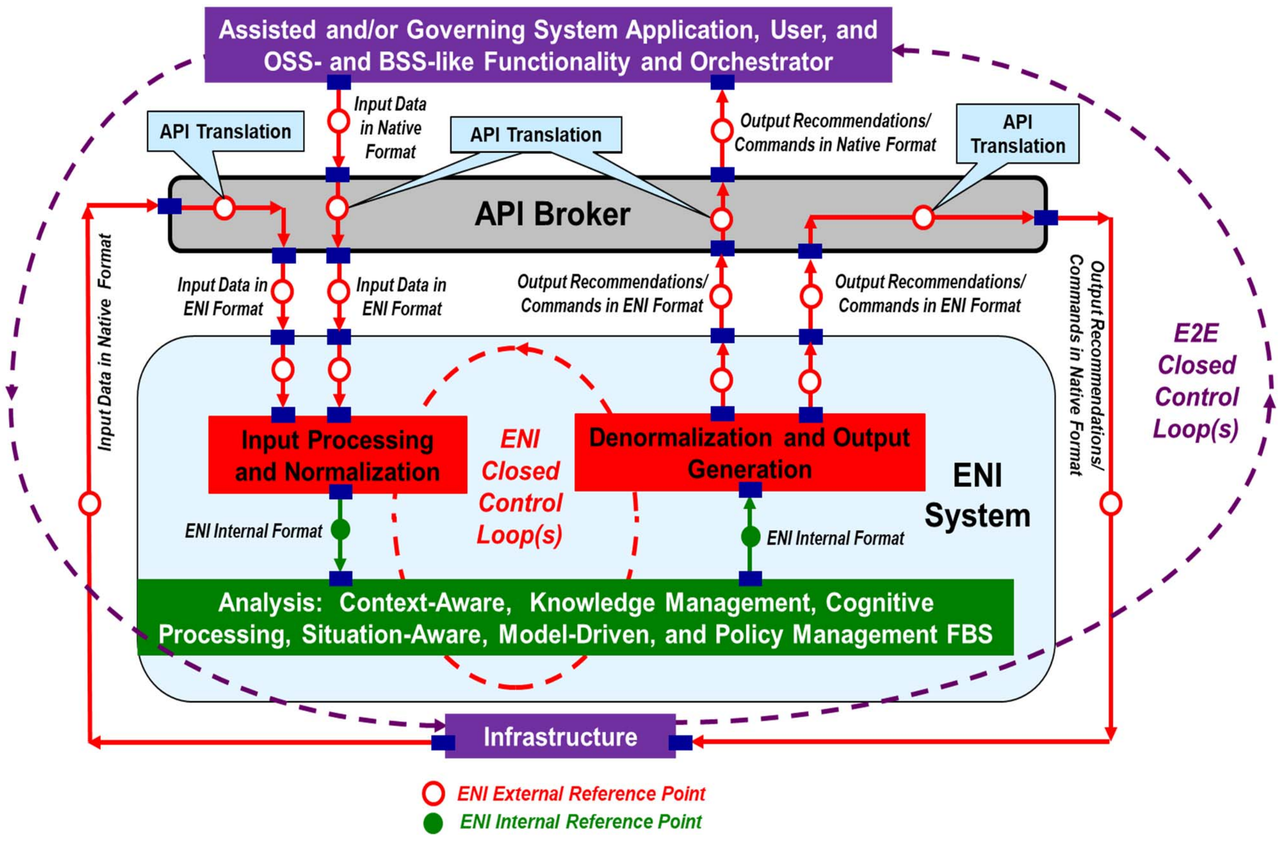

Assisted and/or Governing System Application, User, and OSS- and BSS-like Functionality and Orchestrator
API Translation
Input Data in Native Format
API Translation
Output Recommendations/ Commands in Native Format
API Translation
API Broker
Input Data in Native Format
Input Data in ENI Format
Input Data in ENI Format
Output Recommendations/ Commands in ENI Format
Output Recommendations/ Commands in ENI Format
Output Recommendations/ Commands in Native Format
E2E Closed Control Loop(s)
Input Processing and Normalization
ENI Closed Control Loop(s)
Denormalization and Output Generation
ENI System
ENI Internal Format
ENI Internal Format
Analysis: Context-Aware, Knowledge Management, Cognitive Processing, Situation-Aware, Model-Driven, and Policy Management FBS
Infrastructure
ENI External Reference Point
ENI Internal Reference Point

Figure 4-5: High-Level Functional Architecture of ENI When an API Broker Is Used

Figure 4-6 shows the high-level functional architecture of ENI when an API Broker is not utilized.

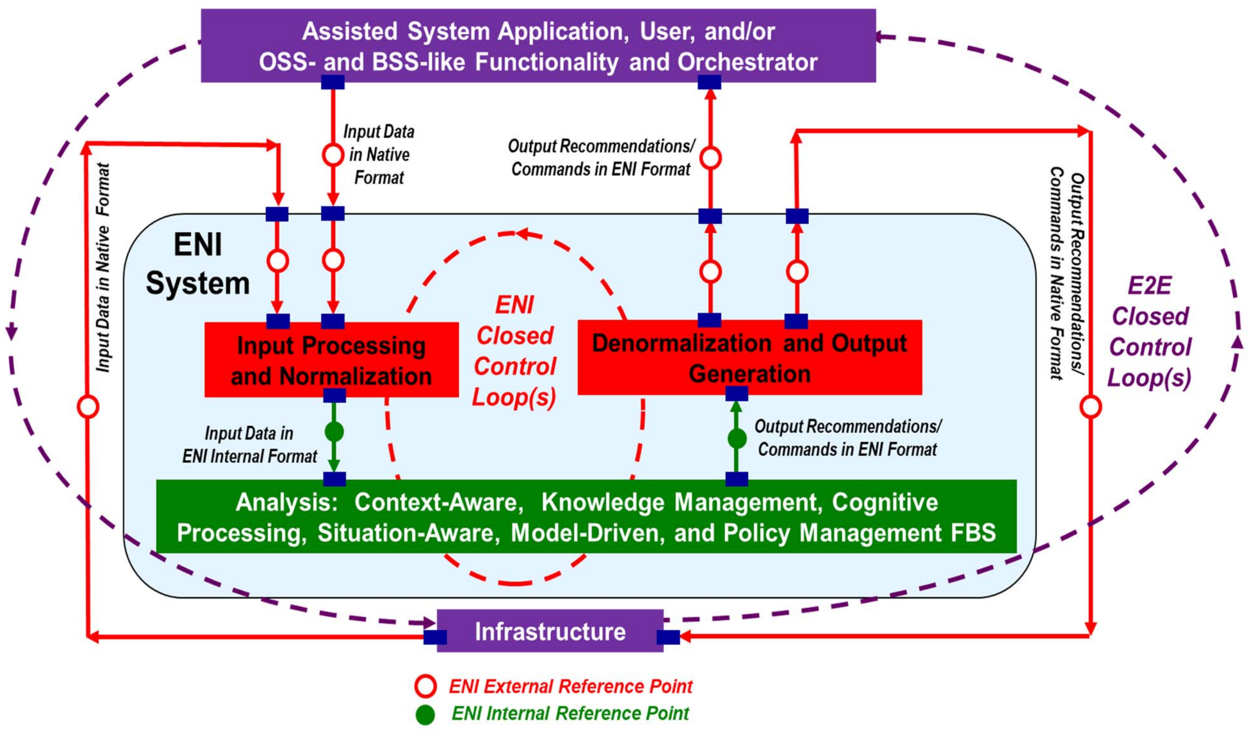

Assisted System Application, User, and/or OSS- and BSS-like Functionality and Orchestrator
Input Data in Native Format
Output Recommendations/ Commands in ENI Format
Input Data in Native Format
Output Recommendations/ Commands in Native Format
ENI System
ENI Closed Control Loop(s)
E2E Closed Control Loop(s)
Input Processing and Normalization
Denormalization and Output Generation
Input Data in ENI Internal Format
Output Recommendations/ Commands in ENI Format
Analysis: Context-Aware, Knowledge Management, Cognitive Processing, Situation-Aware, Model-Driven, and Policy Management FBS
Infrastructure
ENI External Reference Point
ENI Internal Reference Point

Figure 4-6: High-Level Functional Architecture of ENI When an API Broker is Not Used

The purpose of the API Broker is to serve as a gateway (i.e. translation mechanism) between different systems. There is currently no dedicated Functional Block to translate between the APIs and data formats used external to the ENI System and the APIs and data formats used internally by the ENI System. Therefore, without the API Broker, it is possible that the ENI System is not able to communicate directly to external systems (in this case, the Assisted System and its Designated Entity, Applications, Users, and the Infrastructure).

NOTE 2: The issue of how ENI communicates with external systems without an API Broker is for further study in Release 2 of the present document (see clause 9).

The ENI System functional architecture, with or without an API Broker, consists of three types of Functional Blocks:

1) One or more Functional Blocks for ingesting and normalizing data.

2) One or more Functional Blocks for denormalizing and generating recommendations and/or commands.

3) One or more Functional Blocks for analysing ingested data, determining if the Assisted System is operating as expected, and if not, providing a set of recommendations and/or commands.

A key assumption is that the ENI System functionality evolves over time. It is anticipated that the capabilities of various Functional Blocks also evolve over time. As human operators gain confidence in automated decision-making, it is likely that ENI Functional Blocks operate with different and increased levels of autonomicity. Therefore, no attempt is made to standardize the interaction among internal blocks.

Figures 4-5 and 4-6 define the main Functional Blocks of the ENI System. Different use cases (refer to ETSI GS ENI 001 [5]) are realized by different flows through the above Functional Blocks, where any Functional Block is visited zero or more times.

NOTE 3: The addition of new Internal and External Reference Points is for further study in Release 2 of the present document (see clause 9).

ENI is a system that functions in two different modes, as described in clause 4.4.3. Both modes are depicted by showing the interaction of an ENI System as a separate system that gets data from, and provides information (including recommendations and/or commands) to the Assisted System (or its Designated Entity); the difference between the modes is whether commands are provided to the Assisted System (or its Designated Entity) or not. The ENI System itself consists of three or more Functional Blocks; these perform input processing, analysis, and output processing, respectively. An overview of these functions is described in the following clauses; see clause 5.3 for normative descriptions and requirements.

ENI operates in recommendation or command mode, or a combination of these.

NOTE 4: Any Functional Block (except the Data Ingestion, Normalization, Denormalization, and Output Generation Functional Blocks) is allowed to simultaneously operate in recommendation and/or command mode. However, this is not externally observable by the Assisted System (or its Designated Entity).

4.5.2 API Broker

The purpose of the API Broker is to decouple the ENI System from other external systems that it communicates with. In general, each Assisted System has its own unique functionality and set of APIs. This implies that the ENI System would have to understand each of these different sets of APIs in order to communicate with them. This is not desirable, and makes the building of ENI very complex. Instead, the present document defines an API Broker to aid in the translation between external systems and the ENI System. This enables the ENI System to define a single set of APIs. Therefore, the purpose of the API Broker is to:

1) translate data communicated from the Assisted System (or its Designated Entity) into a normalized form that all ENI Functional Blocks are able to understand; and

2) translate recommendations and commands from the normalized form of ENI to a form that the Assisted System (or its Designated Entity) is able to understand.

NOTE: This topic is for further study in Release 2 of the present document (see clause 9).

4.5.3 ENI System Functional Blocks

4.5.3.1 Introduction

The ENI System shown in Figures 4-5 and 4-6 has three types of Functional Blocks. They are:

- Input Processing.
- Analysis, which includes Knowledge Management, Context Awareness, Cognition Management, and Situation- aware, Model-driven, Policy Management.
- Output Generation.

Different types of decision-making strategies that apply to all three types of ENI System Functional Blocks are described in clause 4.5.4.

4.5.3.2 Input Processing

4.5.3.2.1 Overview

Input processing consists of two Functional Blocks: the Data Ingestion and the Normalization Functional Blocks. All external data enters the Data Ingestion Functional Block. It is possible to combine the functionality of the Data Ingestion and Normalization Functional Blocks into a single Functional Block if desired.

4.5.3.2.2 Data Ingestion Functional Block

The purpose of the Data Ingestion Functional Block is to collect data from multiple input sources and implement common data processing techniques to enable ingested data to be further processed and analysed by other ENI Functional Blocks.

Data used by the Functional Blocks in an ENI System typically comes from different sources, is created using different applications and programming languages, and typically is ingested using different protocols. Data is produced by different entities, layers, and domains of the Assisted System (or its Designated Entity), and communicated to the API Broker of ENI via the exposed External Reference Points and/or APIs of the Assisted System (or its Designated Entity).

The API Broker may assist in parsing and possibly translating communicated data into a format defined by the Knowledge Management Functional Block.

It is possible to combine this Functional Block with the Normalization Functional Block if desired.

NOTE: The issue of how ENI communicates with external Systems without an API Broker is for further study in Release 2 of the present document.

4.5.3.2.3 Normalization Functional Block

The purpose of the Normalization Functional Block is to process and translate data received from the Data Ingestion Functional Block into a form that other ENI Functional Blocks are able to understand and use. This enables the data used by the Functional Blocks in an ENI System to be interpreted and understood in a unified and consistent manner; this facilitates the reuse of these Functional Blocks, enables the modularization and generalization of the design of the other Functional Blocks, and supports vendor neutrality. The normalization optionally includes pre- and/or post-processing of the data from different domains. The normalized data is then passed to other Functional Blocks in the ENI System for further processing.

Different data models are likely to be used by different ENI Functional Blocks. Each such data model typically uses different data structures, objects, and protocols to represent its concepts. It is possible for the same concept to be represented differently in different data models (e.g. customer data in a Lightweight Directory Access Protocol (LDAP) or electronic Directory Service (Recommendation ITU-T 9594-1 [i.14] and Recommendation ITU-T 9594-7 [i.15]) vs. the same customer data in a Relational Database Management System (RDBMS)). This is addressed by ensuring that each data model is derived from a single information model, which facilitates reconciling these different representations of the same concept into a single object.

It is possible to combine this Functional Block with the Data Ingestion Functional Block if desired.

NOTE: The topic of how data and information are normalized is for further study in Release 2 of the present document (see clause 9). In particular, the following topics need to be explored:

- Is there a single preferred data model that is used by all Functional Blocks, or is there a single preferred data model that is used by a particular Functional Block, or is there neither of these?
- If there are multiple data models used, how are their differences reconciled?
- The above questions apply to both input ingestion and normalization and denormalization and output generation.

4.5.3.3 Analysis

4.5.3.3.1 Knowledge Management and Processing

4.5.3.3.1.1 Overview

Knowledge Management and Processing consists of three Functional Blocks: Knowledge Management, Context Awareness, and Cognition Management Functional Blocks. It is possible to combine the functionality of these three Functional Blocks into a single Functional Block if desired.

4.5.3.3.1.2 Knowledge Management Functional Block

The purpose of the Knowledge Management Functional Block is to represent information about both the ENI System as well as the system being managed. This includes differentiating between known facts, axioms, and inferences. There are many examples of knowledge representation formalisms, ranging in complexity from models and ontologies to semantic nets and automated reasoning subsystems. It is likely that the ENI System Architecture relies heavily on logic-based as well as various inferencing mechanisms. A more complete and formal definition of knowledge representation is defined in http://www.kr.org/index.php?page=resources. This Functional Block is used by all other Functional Blocks of the ENI System.

4.5.3.3.1.3 Context Awareness Functional Block

The purpose of the Context-Aware Management Functional Block is to describe the state and environment in which a set of entities in the Assisted System (i.e. system being assisted and/or governed) exists or has existed. Context-aware management is used to continuously update the context in which decisions are made.

Context consists of measured and inferred knowledge, and typically changes over time. For example, it is possible that a company has a business rule that prevents any user from accessing the code server unless that user is connected using the company intranet. This business rule is context-dependent, and the system is required to detect the type of connection of a user, and adjust access privileges of that user dynamically.

A more complete and formal definition of context, and how it is used in management, is defined in [i.1], [i.2] and [i.3].

4.5.3.3.1.4 Cognition Management Functional Block

For the purposes of the present document, cognition is defined in clause 3.1 as the "process of understanding data and information and producing new data, information, and knowledge".

The purpose of the Cognition Management Functional Block is to enable the ENI System to understand normalized ingested data and information, as well as the context that defines how those data were produced; once that understanding is achieved, the Cognition Management Functional Block then evaluates the meaning of the data, and determines if any actions need to be taken to ensure that the goals and objectives of the system are met. This includes improving or optimizing performance, reliability, and/or availability. These metrics are exemplary - it is possible to use any other metric used for evaluation of the ENI and/or the Assisted System. The Cognition Framework Functional Block mimics some of the processes involved in human decision-making to better comprehend the relevance and meaning of ingested data.

NOTE: Detailed description of cognition, and how it is used in ENI, is for further study in Release 2 of the present document (see clause 9).

4.5.3.4 Situation-based, Model-driven, Policy Generation

4.5.3.4.1 Overview

Situation-based Policy Generation Functional Block consists of three Functional Blocks: Situation Awareness, Model-Driven Engineering, and Policy Management Functional Blocks.

4.5.3.4.2 Situation Awareness Functional Block

The purpose of the Situation Awareness Functional Block is to enable the ENI System to be aware of events and behaviour that are relevant to a set of entities in the environment of the Assisted System (i.e. system being assisted and/or governed). This includes the ability to understand how information, events, and recommended commands given by the ENI System impact the management and operational goals and behaviour, both immediately and in the near future. Situation awareness is especially important in environments where the information flow is high, and poor decisions have the possibility to lead to serious consequences (e.g. violation of SLAs). A thorough introduction to situation awareness is defined in [i.4].

4.5.3.4.3 Model Driven Engineering Functional Block

The purpose of the Model Driven Engineering Functional Block is to use a set of domain models that collectively abstract all important concepts for managing the behaviour of objects in the system(s) governed by the ENI System.

The use of reusable models defines a set of concepts that are shared by all constituencies that use them. it is permissible for a given constituency to use them directly or indirectly; examples of these are an application developer and a business user, respectively. It maximizes productivity by defining common definitions and usage of concepts used by different entities, including Functional Blocks and systems.

A model is made reusable through the use of common elements that are arranged in similar patterns, or templates. This practice is called design patterns, and occurs at multiple levels of abstraction. ENI is principally concerned with the use of software design patterns and architecture design patterns. A software design pattern is a general, reusable solution in a given context to a commonly occurring problem in the design of a software system. It is not a finished design that is able to be transformed directly into code; rather, it describes how to build the elements of a solution to a type of problem that commonly occurs in designing software. It typically shows concepts and interactions between those concepts; these are translated into classes, attributes, operations, and relationships. An architectural design pattern is similar to a software design pattern, but has a broader scope. It is not an architecture and not even a finished design; rather, it describes how to build the elements of a solution to a type of architecture problem that commonly occurs.

Model Driven Engineering is a software development and implementation methodology that creates reusable domain models, which are conceptual models of concepts, including how concepts relate to each other, which are important to the system being managed. It simplifies the design and implementation processes, and promotes a common understanding of terminology and concepts of the system by different teams working on the system. Put another way, a set of domain models abstract the concepts and activities to be managed. The MDE approach is meant to increase productivity by maximizing compatibility between Functional Blocks and systems through the reuse of standardized models.

A domain is modelled using existing and new design patterns wherever possible.

4.5.3.4.4 Policy Management Functional Block

The purpose of the Policy Management Functional Block is to provide decisions to ensure that the system goals and objectives are met (see clause 6.3.9 for more information on how decisions are made). Policies are used to provide scalable and consistent decision-making. Policies are generated from data and information received by the Knowledge Management and Processing set of Functional Blocks. Formally, according to [i.4], the definition of policy is:

> *"Policy is a set of rules that is used to manage and control the changing and/or maintaining of the state of one or more managed objects"*, see [i.17], [7] and [8].

Policies may be used in several ways in ENI:

- Policies are defined by ENI for managing, monitoring, controlling, and orchestrating behaviour of Functional Blocks in the Assisted System.
- Policies are defined by ENI to request changes in the Assisted System (e.g. for monitoring a new output).
- Policies that are input to ENI by an external entity (e.g. end-user or application) are subject to verification by ENI (e.g. they need to pass a parsing or compilation stage with no errors or warnings produced).

In each case, policies may represent goals, recommendations, or commands. Typically, any information to be conveyed to the Assisted System or its Designated Entity take the form of a set of policies. Each set of policies may be made up of one or more imperative, declarative, and/or intent policy. The details of policy definition, generation, and processing are defined in clause 6.3.9.

4.5.3.5 Output Generation

4.5.3.5.1 Overview

Output Generation consists of two Functional Blocks: Denormalization and Output Generation Functional Blocks. It is permissible that the functionality of these two Functional Blocks is combined into a single Functional Block if desired.

4.5.3.5.2 Denormalization Functional Block

The purpose of the Denormalization Functional Block is to process and translate data received from other Functional Blocks of the ENI System into a form that facilitates subsequent translation to a form that a set of targeted entities are able to understand. For example, different data models are likely to be used by different ENI Functional Blocks. Each such data model typically uses different data structures, objects, and protocols to represent its concepts. It is possible to represent the same concept differently in different data models (e.g. customer data in an LDAP or X.500 directory vs. the same customer data in an RDBMS). This is addressed by ensuring that each data model is derived from a single information model, which facilitates reconciling these different representations of the same concept into a single object.

NOTE: The topic of denormalization is for further study in Release 2 of the present document (see clause 9).

The Denormalization Functional Block receives policies, recommendations, and/or commands. The Denormalization also optionally includes pre- and/or post-processing of the data from different domains to facilitate the translation of these data for a targeted set of entities. The denormalized data is then passed to the Output Generation Functional Block for further processing.

It is possible to combine this Functional Block with the Output Generation Functional Block if desired.

4.5.3.5.3 Output Generation Functional Block

The purpose of the Output Generation Functional Blocks is to convert data received by the Denormalization Functional Block into a form that the Assisted System (or its Designated Entity) is able to understand. This includes defining an appropriate set of protocols, changing the encoding of the data, and other related functions. If an API Broker exists, this output is sent to the API Broker; otherwise, it is sent directly to the Assisted System (or its Designated Entity).

It is possible to combine this Functional Block with the Denormalization Functional Block if desired.

4.5.4 Decision-Making

4.5.4.1 Overview

Decision-making is the cognitive process of selecting a course of action from several possible alternative actions based on the information available and the preferences of the decision maker. Every decision-making process produces a final choice. The final choice does not have to be implemented immediately; this depends on the situation that the decision is made in.

The ENI architecture is specifically designed to provide collaborative interaction of each Functional Block. This enables the domain-specific functionality of each Functional Block to be reused by other Functional Blocks. This avoids having to build too much processing functionality in any one particular Functional Block, and more importantly, implementing the same function differently in different Functional Blocks. ENI combines localized and domain-specific knowledge to develop higher-level, global decisions.

The following clauses describe the progression of decision-making envisioned by the ENI System, from hindsight to reactive to predictive to (full) cognitive processing.

4.5.4.2 Decision-Making using Hindsight

Hindsight is defined as the "realization of the significance and nature of data and events after they have occurred". This realization is typically based on understanding, which in turn requires data, information, and knowledge; wisdom is of course very helpful and needed for all but simple and straightforward (i.e. easy to understand) occurrences. This type of decision-making dominated early management systems until more functionality was available.

4.5.4.3 Decision-Making using Deterministic Processing

Deterministic processing is a step beyond hindsight. Deterministic processing recognizes the occurrence of significant events and data, and invokes predefined rules, processes, and/or policies in response to correct behaviour. A simple example is an expert system, which defines rules to handle different inputs. This type of system responds very quickly and accurately to some events, because the input matches one or more rules that already exist. However, if an input does not match any rule, then this type of system has a problem - it has no choice to give, and no extensible way to make a decision. It has to wait for a new rule.

Put another way, this type of system reacts to past events, and cannot predict future behaviour. However, it represents a significant improvement over hindsight, which is only able to make a decision after the fact.

4.5.4.4 Decision-Making using Predictive Processing

Predictive processing uses a set of processes to predict, or estimate the likelihood of, a future event, state, or behaviour. The prediction or estimation is based on analysing current and historical facts in order to identify risks and opportunities to improve the likelihood of the prediction. Predictive processing also typically allows probability and/or risk assessment. Predictive models often perform calculations during transactions in order to estimate the risk or opportunity involved with making a decision; the actual decision is implemented by the predictive processing entity or some other entity.

4.5.4.5 Decision-Making using Cognitive Processing

A cognitive processing system uses cognition (see clause 6.3.6) to prove or disprove a hypothesis about how to correct an undesirable state in the system. Cognitive processing requires the tight integration of different mechanisms (e.g. knowledge, short- and long-term memory, perception, planning, and reasoning) that together understand what has happened and plan a corrective set of actions. This tight integration is in contrast to modern software architectures that emphasize loose coupling of Functional Blocks.

There is a significant difference between machine learning and cognitive processing. Some examples include:

- Machine Learning is based on concrete mechanisms, such as classification and statistical pattern recognition. In contrast, cognitive processing emphasizes abstract reasoning and problem solving.
- Machine Learning is based on dissimilar, discrete mechanisms. In contrast, cognitive processing emphasizes uniform and integrated representation of data and information.
- Machine Learning is based on pipelines of different functions. In contrast, cognitive processing emphasizes integrated architectures that provide seamless integration between various cognitive functions.

The ENI System architecture employs both traditional AI mechanisms as well as cognitive processes to achieve the best of both worlds.

5 ENI Architectural Requirements

5.1 Introduction

The following clauses define requirements for the ENI system architecture.

5.2 Functional Architectural Requirements for ENI Operation

The ENI System Architecture shall meet the following requirements.

[FAR1] ENI shall operate as a model-driven system.

[FAR2] A domain shall be modelled using existing and new design patterns wherever possible.

[FAR3] Real-time ENI control loop decisions may only be allowed for class 3 Assisted Systems.

- [FAR3.1] Class 1 and Class 2 Assisted Systems may only use ENI recommendations and/or commands in non-real-time decisions.
- [FAR3.2] The ENI System may only be involved in real-time control loop decisions for class 3 Assisted Systems.

[FAR4] ENI shall use one or more closed control loops as part of its main processing architecture.

- [FAR4.1] ENI shall use an OODA-like closed control loop.
- [FAR4.2] ENI shall use one or more closed control loops as part of its architecture.
- [FAR4.3] If ENI uses more than 1 closed control loop, each control loop may operate separately or as part of a hierarchy.
- [FAR4.4] ENI may provide the ability to accelerate the processing through a control loop (i.e. not have to examine each of the constituent elements of the control loop).
- [FAR4.5] ENI shall provide the ability to interrupt the processing of a control loop to perform other actions.

[FAR5] ENI may use agents, organized in various architectures, to implement various functions defined in one or more of its Functional Blocks.

- [FAR5.1] A set of autonomous and/or intelligent agents may be used for data ingestion, normalization, denormalization, and output generation.
- [FAR5.2] ENI may use a variety of agent architectures, ranging from simple reactive architectures to fully cognitive architectures.
- [FAR5.3] ENI may use a distributed or hybrid agent architecture.

[FAR6] ENI shall produce normalized data and information in a consistent format that all ENI Functional Blocks can understand.

- [FAR6.1] ENI shall provide the ability to ingest structured, semi-structured, and unstructured data from different data sources.
- [FAR6.2] ENI may provide the ability to ingest telemetry data in streaming and batch modes, as well as on-demand.
- [FAR6.3] ENI shall provide the ability to contextually change the sources of data to be ingested.
- [FAR6.4] ENI shall pre-process all ingested data to provide a uniform and consistent input format.
- [FAR6.5] The API Broker may help translate input data to be ingested.
- [FAR6.6] The functions of data ingestion and normalization may be separated or combined.
- [FAR6.7] ENI shall provide the ability to contextually change the sources of data to be ingested.
- [FAR6.8] ENI may use a set of models, including data types and data structures, to perform the transformation into a unified data format.

[FAR7] ENI shall process new data and information using its stored knowledge.

- [FAR7.1] If new data, information, or knowledge does not conform to the internal ENI models, then either the models and/or the new data, information, or knowledge shall be changed to realign them.
- [FAR7.2] New data, information, and knowledge ~~provided~~ that is fully processed by one ENI Functional Block shall conform to the internal models held by ENI, at that moment in time, before being passed to another ENI Functional Block.
- [FAR7.3] Alignment of new knowledge with existing knowledge may be done by any set of Functional Blocks at any time in the ENI architecture.
- [FAR7.4] Data, information, or knowledge shall be semantically annotated.

[FAR8] Context-Aware Processing in ENI shall be formally modelled as reusable objects [i.17], [8], [i.1], [i.2] and [i.3].

- [FAR8.1] Context shall be modelled in a hierarchical nature, where different elements of an environment can each have their own changing context.
- [FAR8.2] Context shall be used for assigning different roles to perform context-sensitive behaviour and processing.
- [FAR8.3] Any modelled object may have zero or more contextual roles assigned to it.
- [FAR8.4] Context shall be used for establishing the identity of an object of interest that needs to interact with the ENI System.
- [FAR8.5] Context shall be used for establishing the authentication, authorization, accounting, auditing, and similar privileges of an entity that needs to interact with the ENI System.
- [FAR8.6] Context shall be used for determining what information and data should be retrieved for a given context, as well as how often the information and data need refreshing.
- [FAR8.7] Context shall be used for determining the relevance of ingested information and data.

NOTE 1: ENI uses a model-driven architecture, which means that the current context, as well as changes to that context, can be mapped directly to model elements (e.g. objects, attributes, and relationships). This enables various types of functions, such as fuzzy logic, to be used to define and alter the relevance of ingested information and data.

- [FAR8.8] Context shall be used for determining whether goals are being satisfied.

NOTE 2: Goals are represented as formal (i.e. mathematically well-defined) policies or rules. ENI may use various mechanisms, such as simple weighting or decision trees, as well as fuzzy logic, for measuring relevance and to determine if goals are being satisfied.

[FAR9] Cognition Processing in ENI shall use formal structures to model and represent each stage of the cognition process [8].

- [FAR9.1] Cognition shall use various types of repositories, such as ones dedicated for storing short- and long-term knowledge.
- [FAR9.2] Cognition processing may be used for providing explanations as to which actions ENI took.
- [FAR9.3] Cognition shall be aware of its own capabilities.

NOTE 3: Cognition is the process of acquiring data and information, understanding the data and/or information, and producing new knowledge. its previously stored information and knowledge, as well as modelled objects and behaviours, define the capabilities of an ENI cognitive System. This is similar to how Autonomic Systems are designed and implemented. See [8] and [i.2].

- [FAR9.4] Cognition shall examine its own behaviour to ensure that inappropriate conclusions were not reached.

NOTE 4: Examination of recommendations and commands issued by ENI may be done using a variety of methods. For example, modelled information and data may be translated into finite state automata, and the projected action to be taken by ENI may then be matched to the actual behaviour observed.

- [FAR9.5] Cognition processes shall continuously evaluate and dynamically update the knowledge it has.
- [FAR9.6] Cognition shall be used to meet, preserve, or protect the end-to-end goals of the system.
- [FAR9.7] ENI shall use DSLs to specify end-to-end goals to the Cognitive System.
- [FAR9.8] ENI may use different processes, including first-order logic or computational linguistics, to derive or infer the understanding of a new fact or behaviour.
- [FAR9.9] ENI may store decisions, along with their stimuli, for future use.
- [FAR9.10] ENI shall review decisions to assess their accuracy and to determine if that was the best, or optimal, decision at that particular time.

[FAR10] The ENI System may use negotiation to arrive at a decision that is satisfactory to both the ENI System and the entity that it is communicating with.

- [FAR10.1] If negotiation is not used, or if it is used but fails, the Designated Entity of the Assisted may define a preferred mode for the ENI System to operate in.
- [FAR10.2] If negotiation is not used, or if it is used but fails, the ENI System may have a default mode of operation that specifies the mode of operation to use when a class of decision is reached that is not specified by the Assisted System.
- [FAR10.3] Negotiation shall include the ability for each entity participating in the negotiation process to discover the capabilities of all other entities that it is negotiating with.
- [FAR10.4] Negotiation shall include the ability for each entity participating in the negotiation process to synchronize state with each other.
- [FAR10.5] Negotiation shall include the ability for each entity participating in the negotiation process to agree on parameters and resources to use as part of the negotiation process.

[FAR11] Policies may be used to implement recommendations or commands.

- [FAR11.1] ENI Policies may be defined as imperative, declarative, and/or intent policies.
- [FAR11.2] ENI Policies shall be stored in a form that facilitates their reuse.
- [FAR11.3] ENI Policies may be defined and implemented using one or more Domain Specific Languages.

NOTE 5: ENI DSLs are defined by developing a formal grammar that is used to formally verify the structure and correctness of the grammar, and hence, are testable by definition.

- [FAR11.4] ENI Policies shall reflect any contextual changes that affect their application.
- [FAR11.5] ENI Policies may be situationally aware.

5.3 Architectural Requirements for Mode of Operation

The ENI Mode of Operation shall meet the following requirements.

[MOP1] The ENI System shall support a recommendation mode.

[MOP2] The ENI System may support a management mode of operation.

[MOP3] The ENI System may support a mixed mode of operation for different sets of decision classes (e.g. some types of decisions are made using one mode of operation, and other sets of decisions are made using the other mode of operation).

[MOP4] The ENI System shall discover (and/or be told) the capabilities of the Assisted System in supporting the desired mode of operation.

[MOP5] The ENI System shall support and adapt to external inputs for each mode of operation.

- [MOP5.1] The ENI System shall support and adapt to changes in the context and/or situation of the Assisted System.
- [MOP5.2] The ENI System shall support external input of regulatory policies and operator goals.

[MOP6] The ENI System shall use the above two factors in [MOP.5] to select its mode of operation.

[MOP7] The ENI System shall ask permission from the Operator or Designated Entity, to change modes of operation. This shall be done using an agreed Reference Point.

[MOP8] The Assisted System, or its Designated Entity, shall ask the ENI System to change modes of operation. This shall be done using an agreed Reference Point.

[MOP9] The ENI System shall confirm through the agreed Reference Point to the Operator or Designated Entity of the Assisted System when it has successfully switched modes of operation.

[MOP10] The ENI System may suggest that a particular mode of operation is used when a class of decision is reached that is not specified by the Assisted System.

[MOP11] The Assisted System and/or its Designated Entity need not accept the recommendations offered by ENI when in recommendation mode. This includes those decisions that apply to recommendations when the Assisted System is in a mixed mode of operation.

[MOP12] The Assisted System and/or its Designated Entity need not accept the recommendations offered by ENI when in management mode. This includes those decisions that apply to commands when the Assisted System is in a mixed mode of operation.

[MOP13] Decisions and commands in management mode are subject to the approval of the Assisted System (or its Designated Entity).

NOTE: If the Assisted System has chosen management mode, it (or its Designated Entity) still has the ability to reject commands sent to it by the ENI System.

[MOP14] If the Assisted System (or its Designated Entity) rejects a command set to it by the ENI System when it is in management mode, it shall send a notification to the ENI System.

5.4 Non-Functional Architectural Requirements for ENI Operation

ENI shall meet the following non-functional requirements.

[NFA1] All classes of Assisted System need not change in order to benefit from, and use the outputs of, the ENI System.

- [NFA1.1] The ENI System shall not be responsible for translating its recommendations and commands to a form that the Assisted System, of any class, can understand.
- [NFA1.2] The ENI System may use an API Broker, or similar function, to perform the translation specified in [NFA1.1].
- [NFA1.3] All classes of Assisted System need not change in order for the ENI System to ingest data from the Assisted System.

5.5 Reference Point Requirements

ENI shall meet the following requirements for Reference Points that it defines.

NOTE 1: This topic is for further study in Release 2 of the present document (see clause 9).

[RPR1] The ENI System shall use Reference Points defined in the present document (see clause 6) for all communication and interaction with the Designated Entity of the Assisted System.

- [RPR1.1] The ENI System shall communicate with either the Assisted System or the Designated Entity of the Assisted System by using established ENI Reference Points.

NOTE 2: In both cases, the operator (or its Designated Entity) is in control.

- [RPR1.2] The ENI System need not communicate with the Assisted System by using non-ENI Reference Points.

NOTE 3: Reference Points not defined by ENI are not within the current scope of the ENI Architectural Framework.

[RPR2] All External Reference Points shall provide common characteristics and behaviour.

- [RPR2.1] The ENI System shall provide RESTful interfaces for each External Reference Point.
- [RPR2.2] All External Reference Points may have the capability to define operations that work on lists of objects.
- [RPR2.3] All External Reference Points shall provide standardized exception handling for all operations.
- [RPR2.4] All External Reference Points may provide one or more mechanisms to process multiple messages at a given Reference Point in parallel.

[RPR3] All Internal Reference Points shall provide common characteristics and behaviour.

- [RPR3.1] All Internal Reference Points may have the capability to define operations that work on lists of objects.
- [RPR3.2] All Internal Reference Points shall provide standardized exception handling for all operations.
- [RPR3.3] All Internal Reference Points may provide one or more mechanisms to process multiple messages at a given Reference Point in parallel.

[RPR4] All Internal and External Reference Points shall provide common messaging behaviour.

- [RPR4.1] ENI shall employ standardized messaging patterns to promote interoperability.
- [RPR4.2] ENI may use synchronous and/or asynchronous messaging for internal and external communication using Reference Points.
- [RPR4.3] ENI shall use a set of dedicated message channels for communication over its Reference Points.
- [RPR4.4] ENI shall use messages with appropriate properties to enable data to be exchanged over a channel.
- [RPR4.5] ENI Functional Blocks may use routing to decouple a message sender from a message receiver.

NOTE 4: Communication can be from channel-to-channel, or employ different Functional Blocks to create more complex messaging flows.

- [RPR4.6] ENI Functional Blocks may use filtering or similar operations to determine the destination of the message.

NOTE 5: Filtering enables a message to be forwarded or routed based on it matching certain criteria. This includes the presence or absence of fields in a message as well as their content.

- [RPR4.7] ENI Functional Blocks may use point-to-point, point-to-multi-point, publish-subscribe, or receiver lists to communicate with recipients.

NOTE 6: Combinations of these (e.g. receive a message, split, process, and recombine) may also be supported.

- [RPR4.8] ENI Functional Blocks may use a set of translation mechanisms to change the format and/or structure of the message to suit the needs of message receivers.
- [RPR4.9] ENI Functional Blocks may use a set of enrichment mechanisms to change the content of the message to suit the needs of message receivers.
- [RPR4.10] ENI Functional Blocks may use a set of patterns to consume messages.
- [RPR4.11] ENI Functional Blocks may support durable messaging.

NOTE 7: For the purposes of the present document, durable messaging ensures that if a subscriber is be temporarily disconnected when a message is sent, the subscriber is guaranteed to see this message when it reconnects.

[RPR5] All Internal and External Reference Points shall provide testing to verify that the operation occurring on that Reference Point was successful.

6 ENI Reference Architectural Framework

6.1 Introduction

This architectural framework describes the ENI System as a set of Functional Blocks. Each Functional Block is described in terms of its inputs, outputs, state, and optionally, transfer function. This specifically means that this architectural framework does not define a specific implementation.

6.2 Design Principles of the ENI System architecture

6.2.1 Overview

The following are generic design principles that shall be followed in the design of the ENI System Architecture.

1) The internal implementation of a Functional Block shall not be defined.

2) Functional Blocks may be nested, where the nested Functional Blocks provide greater detail for the containing Functional Block. See clause 6.2.2 for more information.

3) Management, control, interaction with the Assisted System and orchestration of the ENI System shall be defined in terms of Reference Points that may use APIs and/or DSLs.

4) The ENI System may provide management, control, and/or orchestration commands and recommendations to the Assisted System.

5) Architectures from other SDOs shall be interfaced to using a subset of dedicated Reference Points for that purpose.

6) It is desirable for the ENI System to communicate with external actors. To maximize the ease with which this is accomplished, the ENI System shall translate external policies, data, services, and other information to a form it can understand, and shall not rely on external actors understanding its internal functionality. Similarly, outputs from the ENI System shall be translated to a form that external actors can consume. ENI may use an API Broker to insulate ENI from having to know what entity it is specifically communicating with.

7) Any Functional Block may use negotiation to achieve its goals. See clause 6.2.3 for more information.

8) ENI shall use role-based modelling [i.5] and [i.6] to enable different functions and services to be viewed, accessed, and managed. This includes individual and groups of users, objects, Functional Blocks, and applications representing these entities.

6.2.2 Nesting of Functional Blocks

Figure 6-1 shows a simplified notation (compared to UML) for Functional Blocks. The following terms are defined:

- Nested Functional Block: a Functional Block that is contained by another Functional Block
- External Reference Point: a Reference Point that is external to this Functional Block
- Internal Reference Point: a Reference Point that is internal to this Functional Block
- Port: an interaction point between the Functional Block and its environment
- Logical Connectives: logical AND, OR, and NOT operators may be used to define flow paths

Ports may be uni- or bi-directional; this is indicated with one or two arrows, respectively.

Two examples of nested Functional Blocks are shown in Figure 6-1. There are two top-level Functional Blocks, named A and B. A contains two nested Functional Blocks, named A.1 and A.2. B contains two nested Functional Blocks, named B.1 and B.2; B.1 contains two nested Functional Blocks, named B.2.1 and B.2.2.

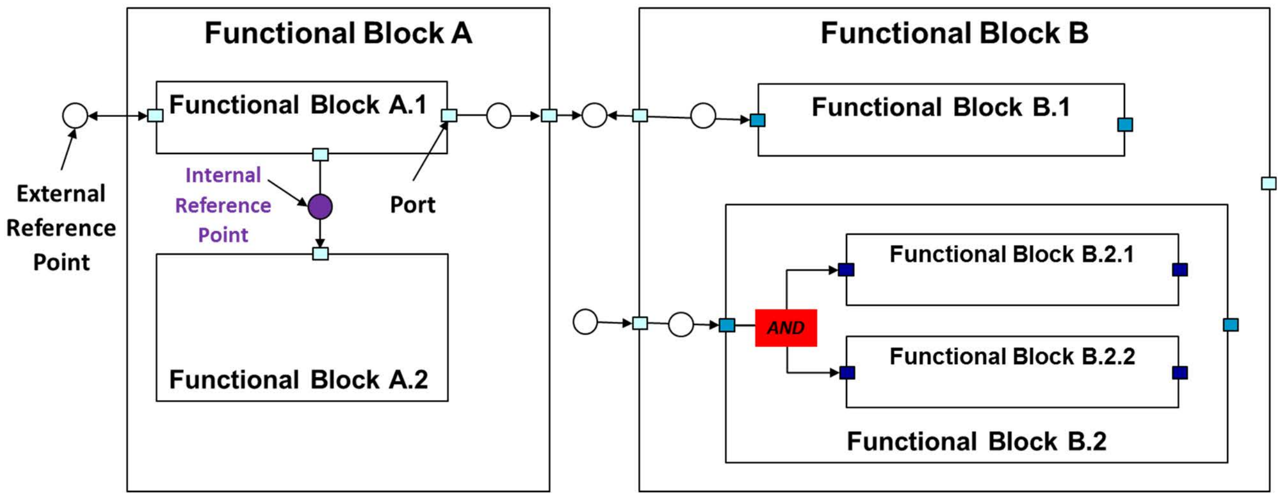


Figure 6-1: Functional Block Notation

In text form, using indentation to indicate nesting, Figure 6-1 may be represented as follows:

- Functional Block A:
    - Functional Block A.1
    - Functional Block A.2
- Functional Block B:
    - Functional Block B.1
    - Functional Block B.2
        - Functional Block B.2.1
        - Functional Block B.2.2

In Functional Block B, a logical AND connector is used to indicate that the input interface (at the level of Functional Block B) shall connect to both Functional Block B.2.1 and Functional Block B.2.2. Logical connectives are not limited to occurring within a Functional Block; they may control flows between any Functional Blocks at any level.

6.2.3 Negotiation

6.2.3.1 Introduction

Negotiation is fundamentally a decentralised process that requires at least two entities, one designated as the buyer and one or more designated as sellers. It is fundamental to various architectures, including message-driven and reactive architectures (see https://www.reactivemanifesto.org/).

ENI shall use role-based modelling to identify buyers and sellers. ENI entities, such as services available in a Functional Block, may participate as a buyer and/or a seller when negotiating externally (i.e. with the Assisted System or its Designated Entity) or internally (i.e. with other Functional Blocks that are in different administrative domains of an ENI System).

6.2.3.2 Distributive Negotiation

Distributive negotiation may be used by the buyer and/or the seller when either wants to gain as much in the negotiating process. Distributive negotiation is a zero-sum game (i.e. each agent's gain or loss is balanced by the losses or gains of the other agents). The participants may adopt different fixed positions (which may be extreme in nature, such as being overly expensive); then, each seeks to give as little as possible before reaching a deal. This encourages one participant to view the other as an adversary, rather than a partner. Put another way, distributive negotiation assumes that there is a fixed amount of value (which could also be services and/or resources) to be divided between the bidding agents.

6.2.3.3 Integrative Negotiation

Integrative negotiation may be used by the buyer and/or the seller when either wants to form partners in the negotiating process to ensure that it, and its partners, can maximize their gain. Integrative negotiation is a win-win (or non-zero-sum) game (i.e. all participating agents can profit). In theory, this improves the quality and likelihood of negotiated agreement by taking advantage of the fact that different parties often value various outcomes differently. Integrative negotiation may have some distributive elements, especially when different agents value different items the same, or when critical details are left to be allocated at the end of the negotiation. Integrative negotiation may involve a higher degree of trust and the formation of relationships, which enables each collaborating agent to "win". Hence, a good agreement is not one with maximum individual gain, but rather, one that provides optimum gain for all collaborating agents. Put another way, integrative negotiation attempts to create value in the course of the negotiation by either compensating the loss of one item with the gain of another item, or by restructuring the contract to specifically enable all collaborating agents to benefit.

6.2.3.4 Functional Model: an Informative Example

A simplified version of a market-based distributed negotiation model is as follows:

- An agent declares itself as a buyer.
- The buyer publishes one or more contracts, where each contract includes a specification of the set of tasks to be performed. The specification encodes a description of the task, any constraints, and metadata information.
- Agents that receive the contract request then decide if they want to bid on the contract. If so, then:
    - Each bidding agent announces itself as a seller.
    - Each seller then replies to the contract by either accepting the contract as is, or presenting a counter-proposal. The counter-proposal may include differences on one or more parameters in each task in each contract.
    - Any seller may generate a set of sub-contracts to help it meet the terms of the contract. If this is done, then the seller becomes a contract-buyer, and follows the above steps until it can satisfy the original contract.
- The (original) buyer then may either accept a seller's bid, or provide a counter-proposal, to all agents that submitted a bid on the contract.

6.3 Architectural Functional Blocks of the ENI System

6.3.1 Introduction

This clause provides a more detailed definition of the ENI System. It elaborates on the four categories of Functional Blocks defined in clause 4.4.5 in the following clauses.

Figure 6-2 shows a more detailed Functional Block Diagram that contains all of its input Reference Points (see clause 6).

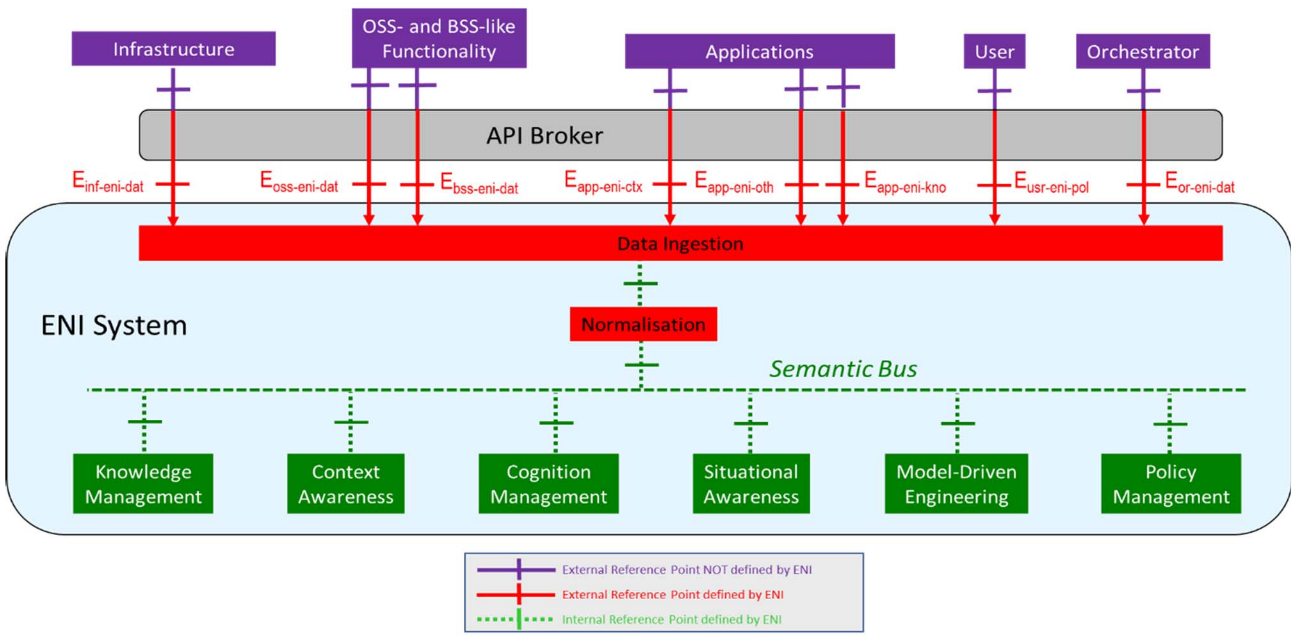

Infrastructure
OSS- and BSS-like Functionality
Applications
User
Orchestrator
API Broker
$E_{inf\text{-}eni\text{-}dat}$
$E_{oss\text{-}eni\text{-}dat}$
$E_{bss\text{-}eni\text{-}dat}$
$E_{app\text{-}eni\text{-}ctx}$
$E_{app\text{-}eni\text{-}oth}$
$E_{app\text{-}eni\text{-}kno}$
$E_{usr\text{-}eni\text{-}pol}$
$E_{or\text{-}eni\text{-}dat}$
Data Ingestion
ENI System
Normalisation
Semantic Bus
Knowledge Management
Context Awareness
Cognition Management
Situational Awareness
Model-Driven Engineering
Policy Management
External Reference Point NOT defined by ENI
External Reference Point defined by ENI
Internal Reference Point defined by ENI

Figure 6-2: Functional Architecture with its Input Reference Points

Similarly, Figure 6-3 shows a more detailed Functional Block Diagram that contains all of its output Reference Points (see clause 7).

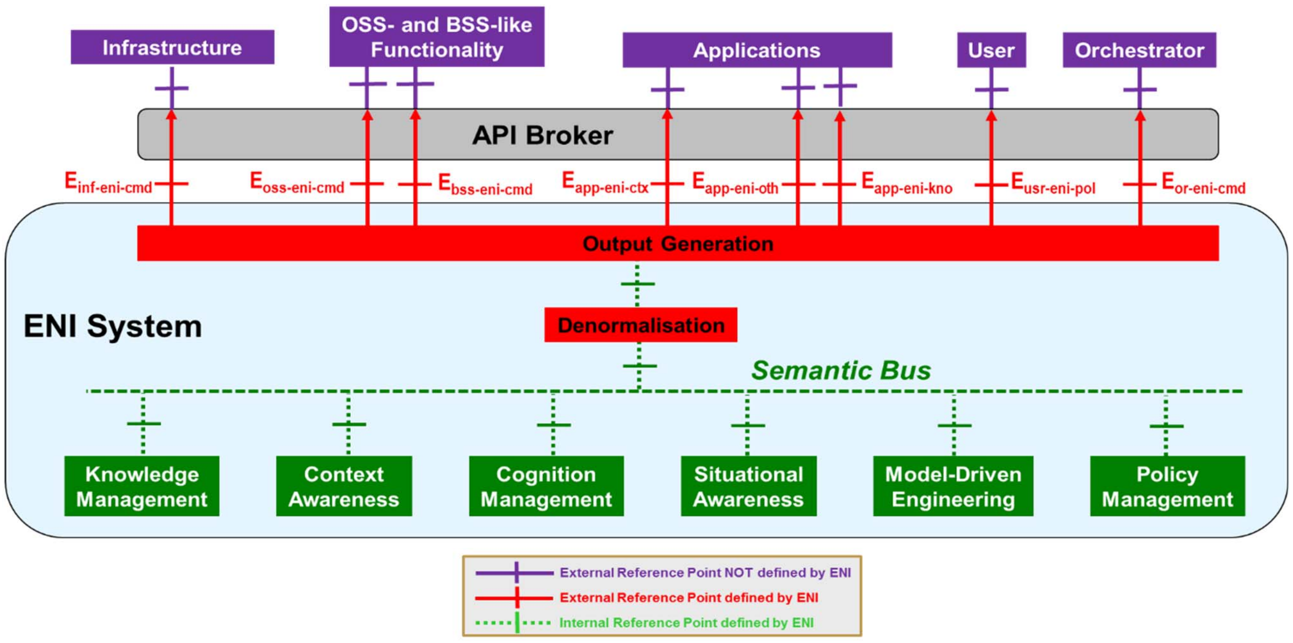

Infrastructure
OSS- and BSS-like Functionality
Applications
User
Orchestrator
API Broker
$E_{inf\text{-}eni\text{-}cmd}$
$E_{oss\text{-}eni\text{-}cmd}$
$E_{bss\text{-}eni\text{-}cmd}$
$E_{app\text{-}eni\text{-}ctx}$
$E_{app\text{-}eni\text{-}oth}$
$E_{app\text{-}eni\text{-}kno}$
$E_{usr\text{-}eni\text{-}pol}$
$E_{or\text{-}eni\text{-}cmd}$
Output Generation
ENI System
Denormalisation
Semantic Bus
Knowledge Management
Context Awareness
Cognition Management
Situational Awareness
Model-Driven Engineering
Policy Management
External Reference Point NOT defined by ENI
External Reference Point defined by ENI
Internal Reference Point defined by ENI

Figure 6-3: Functional Architecture with its Output Reference Points

6.3.2 Data Ingestion Functional Block

6.3.2.1 Introduction

This clause describes the processes associated with ingesting different data from various data sources. Once this is complete, data is then sent to the Normalization Functional Block, which translates the data into a common format for further processing by the other Functional Blocks of the ENI System.

A network has different domains (e.g. RAN/Fixed Access, Transport, and Core). Each domain has its specific functions and services, as well as specific APIs. In a case where the ENI System helps with a localized network function in a specific domain (e.g. optimizing resource allocation at the RAN/Fixed Access), the ENI System may interact with the interfaces of the Assisted Systems of that domain, and may collect data from that domain only. In a more likely case, where the ENI System helps with a cross-domain function (e.g. end-to-end network service assurance), the ENI System may interact with multiple domains of the network. In either case, this shall be done using External Reference Points, and may use the API Broker to insulate ENI from having to change its functionality to accommodate the characteristics and behaviour of different APIs from different Assisted Systems.

These characteristics give rise to the following requirements:

- ENI shall provide the ability to ingest structured, semi-structured, and unstructured data from different data sources. This may be efficiently implemented using a multi-agent architecture.
- ENI shall also provide the ability to ingest data in streaming and batch mode, as well as on-demand. In addition, ENI shall provide the ability to contextually change the sources of data to be ingested. In all cases, the data collected shall be normalized to a uniform data format.
- Data Ingestion may be realized as a Functional Block that is separate from the Data Normalization Functional Block. This adheres to the Single Responsibility Principle [i.10], and enables a more scalable and robust system to be designed and built.

6.3.2.2 Motivation

Each domain has its specific functions, services, APIs, and may run on its specific time cycle. Thus, the data ingestion Functional Block shall be able to operate on different types of domain-specific data. Similarly, the pre- and/or post-processing operations applied to the ingested data are also a function of what Functional Blocks will use the data created from this Functional Block, as well as the nature of functions performed in that domain. The normalization of the time granularity may also be needed, where up/down sampling and interpolation may be applied.

The ENI System shall collect data based on the tasks it needs to perform (e.g. configuration vs. monitoring changes) as well as the nature of the analysis being done (e.g. as congestion is being controlled, different points in the network and different types of data may need to be monitored). The tasks are defined by either the Assisted System (or its Designated Entity) or ENI, typically in response to externally defined goals and policies. In the case when the tasks are defined by the Assisted System (or its Designated Entity), the Assisted System (or its Designated Entity) shall send a request of performing such an action to ENI. In the case when the tasks are defined by ENI, ENI shall send appropriate requests to the Assisted System (or its Designated Entity). Functional Blocks of ENI (e.g. policy management, or cognition management, or situation awareness management) interpret the requirements of each task and are then responsible for defining the types of data, when, and how they are collected.

6.3.2.3 Function of the Data Ingestion Functional Block

6.3.2.3.1 Introduction

The pre-processing may include learning and inferring from the available raw data of one or more domains; once these data are analysed, the pre-processing process shall then decide on what knowledge is forwarded to other Functional Blocks of ENI. The pre-processing may also benefit from model-driven engineering (MDE) mechanisms, since modelled data define how the data should appear and behave in an error-free state. This is discussed more in clause 6.3.8.

In certain cases, the pre-processing process may save the raw form of the ingested data for further use. For example, many types of trend processing require access to raw data. In most cases, the pre-processing function may save the pre-processed form of the data; this is both faster and more efficient. The choice of whether to save the raw or pre-processed form of the ingested data is dependent on the current context (see clause 6.3.5) and/or the current and anticipated situations (see clause 6.3.7). The Situation Awareness Functional Block may also define a set of Policy Rules that take action based on, for example, the type of data or the intended Functional Block that will use the ingested data. The Cognition Framework Functional Block (see clause 6.3.6) may aid in the understanding of ingested data.

The post-processing process may include aggregation and correlation functions (e.g. to reduce dimensionality) as well as machine learning (e.g. this may yield faster results by dealing with significantly smaller data sets, and enable what-if analysis and other game-theoretic algorithms to be used). In such a case, the resulting normalized data may also contain knowledge of a specific domain, or multiple domains.

The particular set of post-processing functions required is dependent on the current context (see clause 6.3.5), the current and anticipated situations (see clause 6.3.7). The Situation Awareness Functional Block may also define a set of Policy Rules that take action based on, for example, the type of data or the intended Functional Block that will use the post-processed data. The Cognition Framework Functional Block (see clause 5.3.6) may augment the meaning of ingested data (e.g. by adding metadata).

Clause 6.3.2 provides examples of pre- and post-processing operations performed by the Data Ingestion Functional Block. Clause 6.3.3 provides examples of operations performed by the Data Normalisation Functional Block.

6.3.2.3.2 Data Filtering

Data filtering is the removal of unnecessary or unwanted information. This is done to simplify and possibly increase the speed of the analysis being performed. This is similar to removing the "noise" in a signal. Filtering is generally (but not always) temporary - the complete data set is kept, but only part of it is used for the calculation.

Filtering requires the specification of rules and/or business logic to identify the data that shall be included in the analysis. Examples include outlier removal, time-series filtering, aggregation (e.g. constructing one data stream from pieces of other data streams, such as merging name, IP address, and application data), validation (i.e. data is rejected because it does not meet value restrictions), and deduplication.

6.3.2.3.3 Data Correlation

Data correlation refers to an association or relationship between data. It expresses one set of data in terms of its relationship with other sets of data. For example, the number of upsells to a higher class of service may increase due to targeted advertising, and may increase even more when offering free time-limited trials. Another example is collecting the complete set of data related to a subscriber. These data are usually collected using different mechanisms, and hence, are fragmented among different collection points. Data correlation can use rules and/or business logic to collect the scattered data and combine it to improve analysis. Data correlation is the first step in gaining increased understanding of relationships between data and their underlying objects.

6.3.2.3.4 Data Cleansing

Data Cleansing is a set of processes that detect and then correct or remove corrupt, incomplete, inaccurate, and/or irrelevant data. Data cleansing typically is performed on batches of data. It differs from data validation in that data validation is performed at the time of ingestion, whereas data cleansing is performed later.

Data cleansing solutions may also enhance the data, either by making it more complete by adding related information or by adding metadata. Finally, data cleansing may also involve harmonisation and standardization of data. For example, abbreviations may be replaced by what they stand for, and data such as phone numbers may be reformatted to a standard format.

6.3.2.3.5 Data Anonymization and Pseudonymization

Data Anonymization is the process of either removing or encrypting information that can be used to identify people from a data set. For the purposes of the present document, the Anonymization process is defined as irreversibly severing data that can be used to identify a person from the data set. Any future re-identification is no longer possible.

Data Pseudonymization is the process of replacing information that can be used to identify a person with one or more artificial identifiers (i.e. pseudonyms). For the purposes of the present document, the Pseudonymization process is reversible by certain trusted entities, since the identifying data was not removed, but rather substituted with other data.

6.3.2.3.6 Data Augmentation

For the purposes of the present document, data augmentation is the process of adding other types of data to the existing data set to enrich it in some way. Two examples are the addition of metadata and the addition of ontological data to a data set to increase the understanding of the data. For example, metadata consists of additional information that describe or prescribe the characteristics, behaviour, and operation of the data. Ontological data is logic-based data that has one or more relationships to the original data that help explain those data. For example, ontological data could provide linguistically related information to provide additional information about the data.

6.3.2.3.7 Data Labelling

Data labelling is the process of adding corresponding class labels to data based on real information provided by the Assisted System (or its Designated Entity). The class labels represent the state or attribute of the data. For example, for a fault detection case, labels could identify normal or abnormal traffic types in a traffic identification case. The labelled data could be used in model training with supervised ML algorithms.

NOTE: Release 2 of the present document will examine if there is enough information at the data ingestion stage to perform this operation, or if this operation is moved to another place in the processing. For example, if new data are ingested, there is a possibility that the normalization process does not know their labels.

6.3.2.4 Operation of the Data Ingestion Functional Block

Different data sources use different languages and protocols to communicate their data. One way to accommodate this is to use a set of agents that each understands a particular data source and the data that it sends. This approach is shown in Figure 6-4. The functional block diagram shown in Figure 6-4 does not prescribe an implementation. Rather, it describes the high-level Functional Blocks that are needed to implement the needs of data ingestion. Different implementations may need to add other Functional Blocks to meet their particular operational requirements.

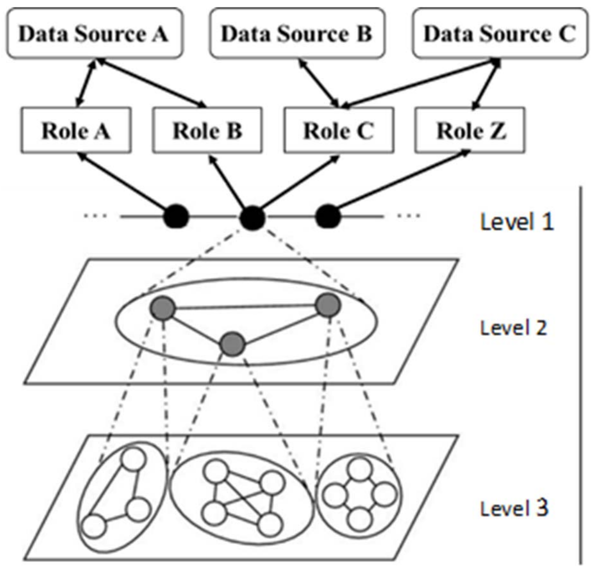


Figure 6-4: Data Ingestion Operation

Figure 6-4 shows a preferred method of operation of the Data Ingestion Functional Block. The bottom three levels represent agents. Level 1 is the agent having the highest level of abstraction, while level 2 is a set of agents that are either controlled by the corresponding level 1 agent or represent functionality required by the corresponding level 1 agent. Similarly, level 3 represent a set of agents that are either controlled by or represent functionality required by the corresponding level 2 agent. This is an example; there is no requirement to implement an agent hierarchy or a hierarchy that has a fixed number of levels for an ENI Agent System. Each agent may take on zero or more roles, and each role may be assigned a set of functions to a given data source. For example, Role A may be assigned the ability to perform any configuration operation required for monitoring, while Role B is assigned the ability to perform traffic engineering operations.

The advantages of this type of architecture include:

- Flexibility: the implementation may change (e.g. lower levels of agents may change) without affecting the API (built from more abstract levels).
- Extensibility: new levels of agents may be added or removed to suit the needs of the application.
- Manageability: instead of having to manage individual agents, the management focus is abstracted to managing roles.

6.3.3 Data Normalization Functional Block

6.3.3.1 Introduction

The Data Normalization Functional Block receives ingested data from the Data Ingestion Functional Block, which translates the data into a common format for further processing by the other Functional Blocks of the ENI System.

These characteristics give rise to the following requirements:

- ENI shall provide the ability to transform structured, semi-structured, and unstructured data from different data sources into a single common, uniform format.
- ENI shall use a set of models, including data types and data structures, to perform the transformation into a unified data format.
- Data Normalization may be realized as a Functional Block that is separate from the Data Ingestion Functional Block. This adheres to the Single Responsibility Principle [i.10], and enables a more scalable and robust system to be designed and built.

6.3.3.2 Motivation

Each domain produces data of interest to ENI using its own processes. Each such process may use different programming languages and protocols, which means that the same data may be received in completely different formats. For example, the same data about a customer may be processed using relational databases and directories; if these customer data are both ingested by the Data Ingestion Functional Block, then the data comprising the customer will be organized differently and may contain different information (e.g. a relational database is easier to query and can store richer data than a directory). If these differences can be identified early in the data acquisition process, then the knowledge processing process is greatly simplified.

In this example, recognizing that the object is a Customer, even though the data is different, may be done by comparing the ingested data with expected data from the model. For example, the Customer object in an information model has a set of mandatory (and optional) attributes, as well as relationships to other objects. It may also have metadata that disambiguates this instance of a Customer from all other instances of a Customer. The normalization process arranges and formats the ingested data that facilitates the comparison and recognition of these and other common characteristics and behaviours.

6.3.3.3 Function of the Data Normalization Functional Block

This clause explains the importance of data normalization.

Data normalization translates the data to a standardized form. This includes the use of pre-defined data structures, where data is converted to a standard format that uses a standard encoding. In the case of ENI, this is facilitated using its set of models. In ENI, models represent objects as templates; this includes defining a set of mandatory and optional attributes, operations, relationships, and other standard features. This enables ingested data to be compared to the same data that is error-free. This is discussed more in clause 6.3.3.4.

A normalized form is critical for the operation of other ENI Functional Blocks. A normalized form means that every ENI Functional Block may be designed using knowledge about the characteristics and behaviour of pertinent data and objects. Otherwise, each ENI Functional Block would have to accommodate data that could be represented in different formats using different data structures.

6.3.3.4 Operation of the Data Normalization Functional Block

6.3.3.4.1 Introduction

The principles of normalization, as used in relational calculus, are a good analogy for the operation of the Data Normalization Functional Block. This analogy is explained in clause 6.3.3.4.2, followed by an example of how data normalization may be used in machine learning in clause 6.3.3.4.3. Finally, clause 6.3.3.4.4 relates these concepts more specifically to the needs of ENI.

6.3.3.4.2 Database Design Analogy

Normalization is a relational calculus design technique that organizes tables in a manner that reduces redundancy and dependency of data. This is done by dividing larger tables to smaller tables and linking the tables using relationships. This ensures that a table is about a specific topic, and that only supporting topics are included. For example, a table that contains information about sales people, support technicians, and customers serves several purposes:

- Identify sales people in an organization
- Identify support technicians in an organization
- Identify all customers of an organization
- Identify which sales people call on which customers
- Identify which support technicians help which customers
- Identify which sales people rely on which support technicians

There are several advantages of having a highly normalized data schema:

- Increased consistency. Information is stored in one place only, reducing the possibility of inconsistent data.
- Single purpose. By limiting a table to one purpose, the number of duplicate data contained within the database is reduced; this eliminates many issues encountered when the database is modified.
- Simplify queries. Normalizing a database identifies different dependencies between data, and makes querying more efficient.
- Easier object-to-data mapping. Highly-normalized data schemata are closer conceptually to object-oriented schemata because the object-oriented goals of promoting high cohesion and loose coupling between classes result in similar solutions.

There are five standardized forms of data normalization. Brief descriptions of the first three are:

- An entity type is in First Normal Form when all entities have a unique identifier, and when each column does not contain any repeating groups of data (i.e. each table cell has a single value, and each record is unique) or composite fields
- An entity type is in Second Normal Form when it is in First Normal Form and when all columns that do not contain unique identifiers depend on the entire unique identifier(s), and not just a part of the identifier
- An entity type is in Third Normal Form when it is in Second Normal Form and when each column that is not part of the unique identifier does not depend on another column that is not part of the unique identifier (i.e. changing a non-key column does not cause other non-key columns to change - this is called a transitive functional dependency)

The primary disadvantage of normalization is slower reporting performance. This is accomplished through using a separate, denormalized data model.

6.3.3.4.3 Normalization for Machine Learning

A neural network may be trained by supplying data and then comparing the expected output to the true output of the network. The model parameters may then be updated using a number of algorithms. For example, gradient descent updates the parameters of the model in the direction that minimizes the difference between the expected (or ideal) outcome and the true outcome. There are different types of gradient descent; however, all of them scale the magnitude of the parameter update by a learning rate. This rate ensures that the parameter is not being changed too drastically, which can cause the update to overshoot and fail to find the optimal value. The normalization of the input data to a standard scale enables the network to *more quickly* learn the optimal parameters for each input node. It also reduces complex computational problems associated with floating point number precision.

In a neural network, changing one weight affects subsequent layers, which then affect subsequent layers, and so on. This means changing one weight can affect activations in subsequent layers in complex ways. By ensuring the activations of *each* layer are normalized, the overall loss function topology is simplified. This is especially helpful for the hidden layers of the network, since the distribution of unnormalized activations from previous layers will change as the network evolves and learns more optimal parameters.

One method for doing this is called batch normalization. This controls the magnitude and mean of the activations independent from all other layers. Other methods include weight normalization (i.e., instead of normalizing activations in the intermediate layers of neural networks, the weight vectors of a neural network are reparameterised to enable optimisation to converge more quickly); layer normalization (i.e. instead of normalizing the input features across the batch dimension as in batch normalization, it normalizes the inputs across the features); group normalization (i.e. the mean and standard deviation over groups of channels for each training example).

6.3.3.4.4 Applying Normalization to ENI

In ENI, the Normalization Functional Block uses modelled information (e.g. objects, attributes, operations, and relationships) to *standardize* the information being sent to other ENI Functional Blocks. ENI may use a number of different mechanisms, including those described in the previous two clauses, to identify and convert data into model elements (e.g. classes, attributes, operations, and relationships). Metadata may also be used to describe and prescribe how normalization is performed; this may help reversing the process when ENI needs to output recommendations and commands to the Assisted System (and/or its Designated Entity). In order to do this, accumulated knowledge from other ENI Functional Blocks (included predefined model information) shall be made available to the Normalization Functional Block, as shown in Figure 6-5. The functional block diagram shown in Figure 6-5 does not prescribe an implementation. Rather, it describes the high-level Functional Blocks that are needed to implement the needs of normalization. Different implementations may need to add other Functional Blocks to meet their particular operational requirements.

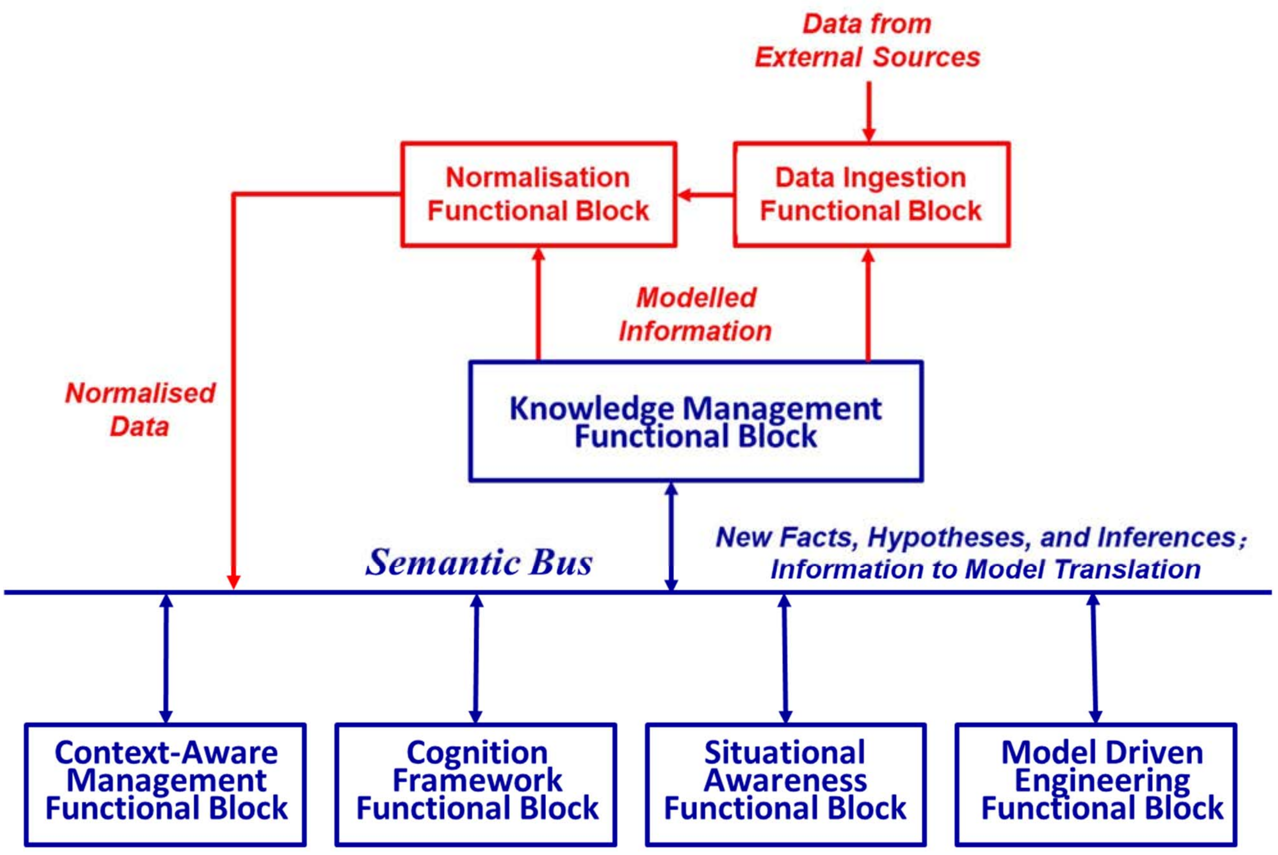

Data from External Sources
Normalisation Functional Block
Data Ingestion Functional Block
Modelled Information
Normalised Data
Knowledge Management Functional Block
Semantic Bus
New Facts, Hypotheses, and Inferences; Information to Model Translation
Context-Aware Management Functional Block
Cognition Framework Functional Block
Situational Awareness Functional Block
Model Driven Engineering Functional Block

Figure 6-5: Data Normalization Operation

Data from the Normalization Functional Block is sent to the Semantic Bus. This is a uni-directional communication; information from the Semantic Bus is not sent to the Normalization Functional Block; it is first filtered by the Knowledge Management Functional Block.

Knowledge from the Context-Aware, Cognition Framework, Situational Awareness, and Model Driven Engineering Functional Blocks result in changes to the models used by ENI. Changes can be in the form of new model elements as well as corrections and elaborations to existing model elements. These changes are important, as it enables the modelled information to evolve as the ENI System learns; it also enables the knowledge itself to adapt (e.g. be used in different ways) as the context and situation changes. Hence, the ENI models are a form of active repository. The results are sent to the Knowledge Management Functional Block. This is discussed more in clause 6.3.4.

6.3.4 Knowledge Management Functional Block

6.3.4.1 Introduction

Knowledge management is fundamental to all disciplines of modelling and AI. Briefly, knowledge representation defines a set of formalisms that define information and knowledge in a form that a computer system can utilize. Knowledge management includes processes, strategies, and systems that create, revise, sustain, and enhance the storage, assessment, use, sharing, and refinement of knowledge assets using a consensual knowledge representation.

Knowledge management also enables machine learning and reasoning - without a formal and consensual representation of knowledge, algorithms cannot be defined that reason (e.g. perform inferencing, correct errors, and derive new knowledge) about the knowledge. Knowledge representation defines mechanisms for the characteristics and behaviour of the set of entities being modelled; this enables the computer system to plan actions and determine consequences by reasoning using the knowledge representation, as opposed to taking direct action on the set of entities.

Knowledge management works with data, information, knowledge, and wisdom; this is explained in clause 4.4.5.6. In particular, this Functional Block directs which context and situation information are applied to the raw data, transforming it to information and then knowledge. If appropriate, wisdom is then derived from those two translations. As such, it is recommended that data is always first transformed into information; this is because there is no context or situation knowledge to understand the data.

The Knowledge Management Functional Block provides foundational functionality that shall be used to analyse, validate, and infer new and changed data, information, knowledge, and wisdom. Thus, it is used in a variety of tasks, including analysis and decision-making.

6.3.4.2 Inferencing

An inference system automatically extends the knowledge base of the system. If inductive or abductive inferencing is used, then the conclusions are not guaranteed, and some knowledge may be in error. ENI shall use inferencing to extend and correct its knowledge base. For example, an inference system can gather evidence and develop a hypothesis, which can later be proved to be true or false. Such hypotheses can also be generalized. For instance, suppose that ENI has observed multiple occurrences of a set of faults that cause performance degradation that eventually lead to a Service Level Agreement (SLA) violation. A possible conclusion could be that whenever this set of faults occurs, an SLA will always occur. That may not be true. In this situation, ENI may add this conclusion to its knowledge base while using metadata to describe its source and/or add its approximate probability. Since ENI is an experiential system, ENI shall update the metadata (and most importantly, its probability) as it gathers more evidence to prove or disprove this hypothesis. ENI may also use multiple types of inferencing to validate hypotheses. For example, in addition to abduction, ENI may use reinforcement learning to more quickly converge to a conclusion as to the validity of the hypothesis.

Inferencing is critical for realizing cognition. ENI uses a closed control loop powered by a model-driven architecture. The closed control loop is based on FOCALE's extensions to the Observe-Orient-Decide-Act (OODA) control loop [i.17], [i.1], [i.2] and [i.4]. See clause 6.3.4.5 for more detail.

6.3.4.3 Motivation

A knowledge representation framework defines a set of primitives that define and describe the meaning, and optionally additional semantic characteristics (e.g. synonyms and antonyms), of concepts important to the managed environment that can be processed by a computer system. This includes how these concepts relate to each other.

Key concepts that determine the function and nature of a knowledge representation framework include:

- How can knowledge be represented
- How can knowledge be discovered, extracted, translated, and searched
- How can knowledge be used for further inference
- How can knowledge from different sources be normalized and composed

A knowledge framework can be as simple as a collection of data structures and tools to manipulate those data structures, to an architecture for representing and processing knowledge (e.g. semantic networks), to dedicated languages for representing knowledge (these also include languages known as theorem provers that are typically based on First Order Logic). Ontologies, and ontological tools, are typically used to provide and/or reinforce knowledge that is linguistic in nature and/or is amenable to logic-based processing. Finally, dedicated tools for knowledge extraction, filtering, and fusion may be required in environments where knowledge from different data sources needs to be combined, filtered, correlated, or otherwise processed in order to produce new knowledge.

Knowledge frameworks are specific to the applications that use them. The knowledge framework of ENI is designed to support situation awareness and cognitive decision-making.

NOTE: The topic of knowledge representation and management is for further study in Release 2 of the present document (see clause 9).

6.3.4.4 Function of the Knowledge Management Functional Block

Context defines the knowledge about an Entity that exists or has existed. It is modelled as a set of objects and relationships. The context model shall include mechanisms that represent changes in the environment, as well as changes in behaviour of an ENI Functional Block as a set of closed loop systems.

There are a number of existing technologies and open source that can be used to represent context and knowledge.

NOTE: Knowledge management and processing is for further study in Release 2 of the present document (see clause 9).

6.3.4.5 Operation of the Knowledge Management Functional Block

6.3.4.5.1 Introduction

Figure 6-6 shows a highly simplified view of basic knowledge processing based on the OODA control loop [9]. The functional block diagram shown in Figure 6-6 does not prescribe an implementation. Rather, it describes the high-level Functional Blocks that are needed to implement the needs of knowledge management. Different implementations may need to add other Functional Blocks to meet their particular operational requirements.

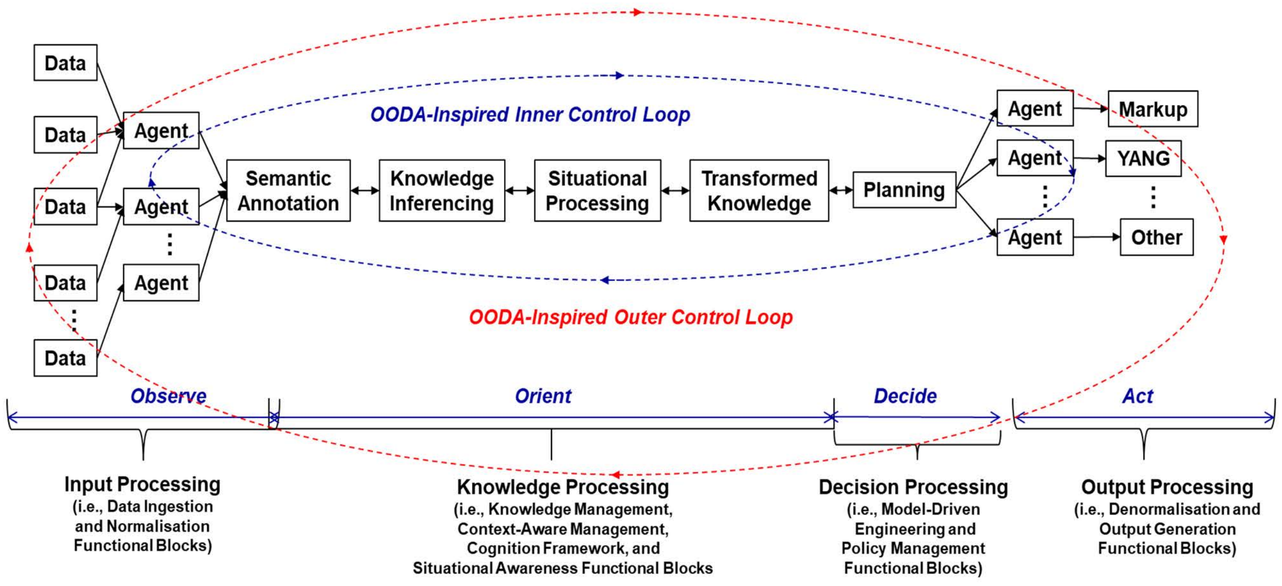

Data
Data
Data
Data
Data
Agent
Agent
Agent
OODA-Inspired Inner Control Loop
Semantic Annotation
Knowledge Inferencing
Situational Processing
Transformed Knowledge
Planning
Agent
Agent
Agent
Markup
YANG
Other
OODA-Inspired Outer Control Loop
Observe
Orient
Decide
Act
Input Processing (i.e., Data Ingestion and Normalisation Functional Blocks)
Knowledge Processing (i.e., Knowledge Management, Context-Aware Management, Cognition Framework, and Situational Awareness Functional Blocks
Decision Processing (i.e., Model-Driven Engineering and Policy Management Functional Blocks)
Output Processing (i.e., Denormalisation and Output Generation Functional Blocks)

Figure 6-6: Simplified Knowledge Processing Flow

Figure 6-6 uses a multi-level set of OODA-inspired control loops [9], similar to FOCALE [8]. Both of these are discussed in more detail in clauses 6.3.4.5.2 through 6.3.4.5.5.

The assignment of Functional Blocks to each stage of OODA processing illustrates the main activities of those Functional Blocks. Some Functional Blocks, such as Knowledge Management, may perform tasks in other parts of the OODA loop. For example, the Knowledge Management Functional Block provides the knowledge representation used by all ENI Functional Blocks. In FOCALE, this principle was used to facilitate input processing of ingested data by matching the data, along with observed patterns of data, to known knowledge to more quickly and efficiently validate and verify ingested data. As another example, the Situational Awareness Functional Block is responsible for understanding the relevant changes that pertain to the data, information, and behaviour of the Assisted System, how those changes affect the goals of the ENI System, and to project what is likely to happen in the future (see clause 6.3.6). This spans the Orient and Decide functions of OODA.

Figure 6-6 is depicted as a sequential flow for simplicity. However, OODA is a form of closed control loop, and hence, the operations in the Knowledge and Decision Processing groups of functional blocks typically interact in a non-sequential manner. This is made more apparent in FOCALE, which is an autonomic architecture using an enhanced version of the OODA control loop.

The following clauses describe the functionality of the Knowledge Processing and Decision Processing sections in the context of an OODA control loop. In actuality, this is a hierarchical set of OODA-inspired control loops; the "outer" control loop consists of the four sets of processing operations, and the "inner" control loop is the Knowledge and Decision Processing operations. This inner loop is necessary to properly process and derive knowledge and wisdom, and perform accurate inferencing (see clause 6.3.5), on contextually-changing information.

Reference [10] describes why the OODA loop was chosen over other methodologies, such as MAPE.

6.3.4.5.2 Observe Functionality

Input processing uses a set of agents to ingest input data. It consists of the Data Ingestion and Normalization Functional Blocks, as described in clauses 6.3.2 and 6.3.3. This stage corresponds to the "Observe" part of the OODA control loop. The purpose of this part of the OODA control loop is to ensure that the correct data and information is gathered. The output of the input processing agents is sent to the Knowledge Processing functional block.

The OODA loop assumed that different data and information was normalized and combined. This process was first explicitly realized using the FOCALE architecture. This is why it is recommended that the input processing consist of two Functional Blocks; this adheres to the Single Responsibility Principle [i.10], a crucial software design tenet for building scalable software-intensive systems.

6.3.4.5.3 Orient Functionality

Knowledge Processing consists of four main Functional Blocks: Knowledge Management, Context-Aware, Cognition Framework, and Situational Awareness. The last three are described in clauses 6.3.5, 6.3.6, and 6.3.7. This set of Functional Blocks correspond to the "Orient" part of the OODA control loop. The purpose of this part of the OODA control loop is to ensure that input data and information is adapted to the current context and situation.

The "Orient" part of the OODA control loop consists of three types of processing: semantic annotation, knowledge inferencing, and situational processing. In each case, one or more of these four Functional Blocks may perform each of these three functions.

Semantic Annotation is a set of processes that examine the input data, and annotates it where possible to enable it to be better understood by subsequent Functional Blocks. The annotation is based on the current context and situation. This enables behaviour to be adapted based on changes to the current context and/or situation to ensure that the goals of the system and protected and maintained. Knowledge Inferencing, which is done primarily by the Cognition Framework and the Situation Awareness Functional Blocks, then follows.

The contextually-aware data is then ready for inferencing operations to be performed to discover more about the nature and meaning of the data. Different types of inferencing (e.g. structural and logical) may then be performed. Inferencing produces knowledge from knowledge, and the newly produced knowledge shall also be integrated into the collective ENI knowledge base (this is not shown in Figure 6-6). The inferencing may take different forms, but is predominantly semantic in nature. For example, if a Customer object is recognized, then the annotations may describe relevant SLAs and contracted Services that are currently active. As another example, it may define applicable business and regulatory policies that shall be considered that are based on the current context. At this point, the newly ingested information has completed "pre-processing", and is ready for situational processing (the last stage of the "Orient" operation of the OODA control loop).

Situational processing analyses the "pre-processed" knowledge to determine what (if anything) has just occurred that does not align with its system goals. This could take many forms, such as determining that possible system degradation may occur, or that the system could be better optimized, or that one or more of its goals are in jeopardy. If no problems have occurred, then actions need not be taken. Otherwise, the situational processing then decides what is likely to happen, and how that affects the goals that the system is trying to achieve. This produces a set of possible alternative actions. ENI shall examine each set of actions, and choose the set that best meet its current set of goals.

6.3.4.5.4 Decide Functionality

The "Decide" part of the OODA control loop consists of two Functional Blocks: Model-Driven Engineering and Policy Management. These are described in clauses 6.3.8 and 6.3.9, respectively. The Model-Driven Engineering Functional Block acts as the "brains", and the Policy Management Functional Block acts as the "brawn". The purpose of this part of the OODA control loop is to decide whether any action should be taken to preserve the goals of the system.

The Model-Driven Engineering Functional Block takes its input from the "Orient" function, that compares the current state and situation of the system being managed to its desired state for that situation, and has recommended a set of corrective actions to take. Up to this point, all analysis has been done on the system model, which is a logical model that defines the manageable objects, relationships, and constraints that are being managed. The Model-Driven Engineering processes take the changes to the system model and then translates the appropriate set of actions into a form that can be implemented by the Assisted System or its Designated Entity. The result of this translation is then given to the Policy Management Functional Block.

NOTE: The present document does not define how the above translation is done, as that is implementation-specific. In Release 2 (see clause 9), the input and output requirements of the above translation will be defined, enabling this Functional Block to be more completely specified.

The Policy Management Functional Block translates the output of the Model Driven Engineering Functional Block to a reusable set of policies. ENI Policies are made reusable in two ways: (1) storing the policy as an object in one or more models and repositories, and (2) storing the language definition of the policy in one or more repositories. Each ENI policy may be written in imperative, declarative, and/or intent forms, and may represent recommendations or commands. The use of policies provides two important benefits. First, it defines a reusable output that can be stored and retrieved; this is critical for explanation-based auditing of what decisions the ENI System made. Second, it simplifies construction and integration issues, since any policy is an instance of an underlying language. ENI may use Domain Specific Languages (DSLs) to implement one or more forms of policies.

6.3.4.5.5 Act Functionality

The "Act" part of the OODA control loop consists of two Functional Blocks: Denormalization and Output Generation. These are described in clauses 6.3.10 and 6.3.11, respectively. The functionality of the Denormalization and Output Generation Functional Blocks may be combined into a single Functional Block if desired. The purpose of this part of the OODA control loop is to take one or more actions to preserve the goals of the system.

The input to the Denormalization Functional Block is the set of Policies defined by the Policy Management Functional Block. These policies are in a normalized form for ENI; the Denormalization Functional Block translates them to a denormalized form to enable external Functional Blocks to understand each policy. This may be done using a set of output processing agents. Each agent is responsible for de-normalizing the data (e.g. translating the knowledge into a specific language and format for a given set of devices). For example, these agents could translate an ENI policy to an alternative form; one example is translating an ENI declarative policy to Open Stack Congress. This is then sent to the Output Generation Functional Block.

The Output Generation Functional Block packages the translated ENI Policy to a form appropriate for the external entity that is using the ENI Policy. Output processing may use a set of agents to translate the ENI Policy into specific forms (e.g. per vendor and per device, or into a payload for a specific protocol that uses a particular encoding format).

6.3.5 Context-Aware Management Functional Block

6.3.5.1 Introduction

This clause describes the motivation for using context-aware behaviour, the effect it has on the System Architecture, and the benefits that it provides. See clause 6.3.7or a description of situation awareness, and specifically clause 6.3.7.5 for a comparison between this Functional Block and the situation awareness Functional Block.

6.3.5.2 Motivation

Most management systems today are collections of functionality that operate as silos. They are characterized by:

- Static, end-to-end process-based management.
- Little or no understanding of context, and hence, no ability to provide behaviour and services personalized to a given context.
- No ability to change behaviour due to changes in context.

Context may produce a higher intrinsic value for data versus raw data, and hence, make the generation of information easier. Context-awareness assumes that either the data and/or the associated metadata deliver additional information about the characteristics and behaviour of the environment that ENI interacts with. Context ensures that ENI is focused on tasks that the operator defines as important. The additional knowledge provided by context may offer a greater level of reliability and usefulness, both for the information and knowledge gathered by ENI as well as for the actions taken by ENI.

6.3.5.3 Function of Context Awareness

Context-awareness enables a system to gather information about itself and its environment. This enables the system to provide personalized and customized services and resources corresponding to that context. More importantly, it enables the system to *adapt its behaviour according to changes in context*.

Context-awareness enables diverse data and information to be more easily correlated, and hence, integrated, since context acts as a unifying filter. As such, identifying contextual information is critical for understanding both ingested data and information as well as how data and information, as well as existing knowledge and wisdom, can be affected.

The contextual history of a user, a user application, or a device, as well as its prior interactions with ENI (including, for example, session state), may be useful for driving policy decisions regarding the current and future interaction between ENI and that entity, including decisions made by ENI that affect that entity. For example, past behaviour can be used to more quickly arrive at a decision. Alternatively, historical information can be used to flag anomalies that need further action to resolve.

Contextual data may be categorized in terms of data and information ENI and entities that ENI interacts with. This includes a user, the applications of a user, the time and location of objects that the user interacts with, and different types of relationships that exist among entities that ENI interacts with as well as between ENI and those entities. Examples include:

- Personal and group information for a user (e.g. contact information, roles played by the user, and profile and preference information).
- Location of the user and any devices that the user is using or interacting with (e.g. geo-code, the centroid of a surrounding polygon).
- Characteristics and behaviour of a device (e.g. type of access provided (if any), MAC and IP addresses, and features such as forward, routing, encryption).
- Characteristics and behaviour of any applications used by a user.
- Types of relationships that exist between ENI and entities that use or interact with ENI, as well as the nature of those relationships (e.g. is it temporally sensitive or static).
- Types of relationships that exist between entities that ENI interacts with.

The Context of an Entity is a collection of measured and inferred knowledge that describe the state and environment in which an Entity exists or has existed [i.2]. In particular, this definition emphasizes two types of knowledge - facts (which can be measured) and *inferred* data, which results from machine learning and reasoning processes applied to past and current context. It also includes context history, so that current decisions based on context may benefit from past decisions, as well as observation of how the environment has changed.

6.3.5.4 Operation of the Context Awareness Functional Block

A good overview of designing for context-awareness is provided in [i.2].

Figure 6-7 shows the Context-Aware Management Functional Block connected to a Semantic Bus (see clause 6.3.4 for a description of the functionality of a Semantic Bus). The functional block diagram shown in Figure 6-7 does not prescribe an implementation. Rather, it describes the high-level Functional Blocks that are needed to implement the needs of context-awareness. Different implementations may need to add other Functional Blocks to meet their particular operational requirements.

Figure 6-8 shows some of the roles that can be used in a context-aware system. The overall (aggregate) context is used to derive one or more of the five exemplary roles shown, which in turn are used to define behaviour appropriate for that context. The five roles are described below.

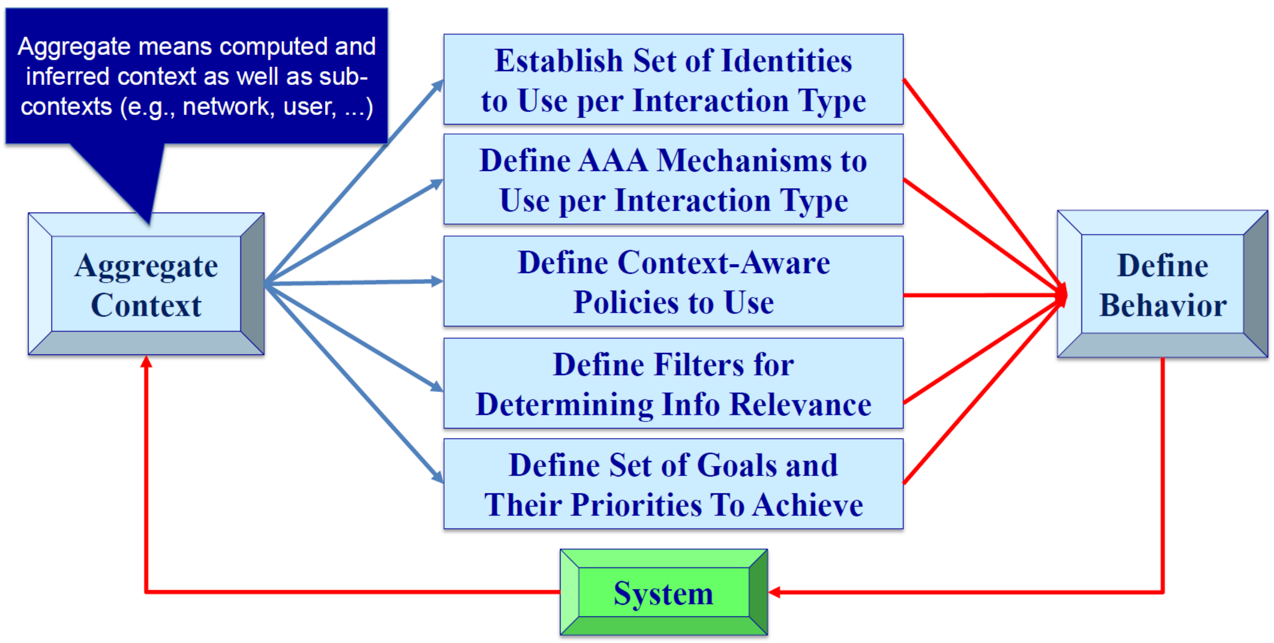

Aggregate means computed and inferred context as well as sub-contexts (e.g., network, user, ...)
Aggregate Context
Establish Set of Identities to Use per Interaction Type
Define AAA Mechanisms to Use per Interaction Type
Define Context-Aware Policies to Use
Define Filters for Determining Info Relevance
Define Set of Goals and Their Priorities To Achieve
Define Behavior
System

Figure 6-7: Contextual Roles

- **Identities per Interaction Type.** This role defines a set of processes that define the characteristics and behavior of a particular set of identities for objects in this context. Interaction type includes the set of entities in the context that this object interacts with, along with how the interaction is done (e.g. which set of protocols are used). For example, a passport is generally required to authorize travel to a foreign country, but is likely not usable when a user logs onto their personal computer.

- **AAA Mechanisms per Interaction Type.** This role defines the applicable set of authentication, authorization, and accounting interactions for this particular context. It may include other related operations, such as auditing.

- **Context-Aware Policies to Use.** This role defines a set of policies that shall be produced by the Policy Management Functional Block in response to changes in the current state deviating from the desired state for this context.

- **Filters for Determining Relevance.** This role defines a set of filters that determine the relative relevance of ingested data and information for this particular context. This enables the system to:

  1) adapt which data are gathered as context changes; and

  2) modify its processing according to the contextual changes.

- **Set of Goals to Achieve.** This role defines a set of goals to achieve for this particular context. For example, if the ratio of the users of the Platinum, Gold, Silver, and Bronze service classes change, then the ENI System may need to generate a new set of policies that govern the behavior of each service class in order to maximize revenue.

Figure 6-8 shows an exemplary set of operations that occur when a context is analysed.

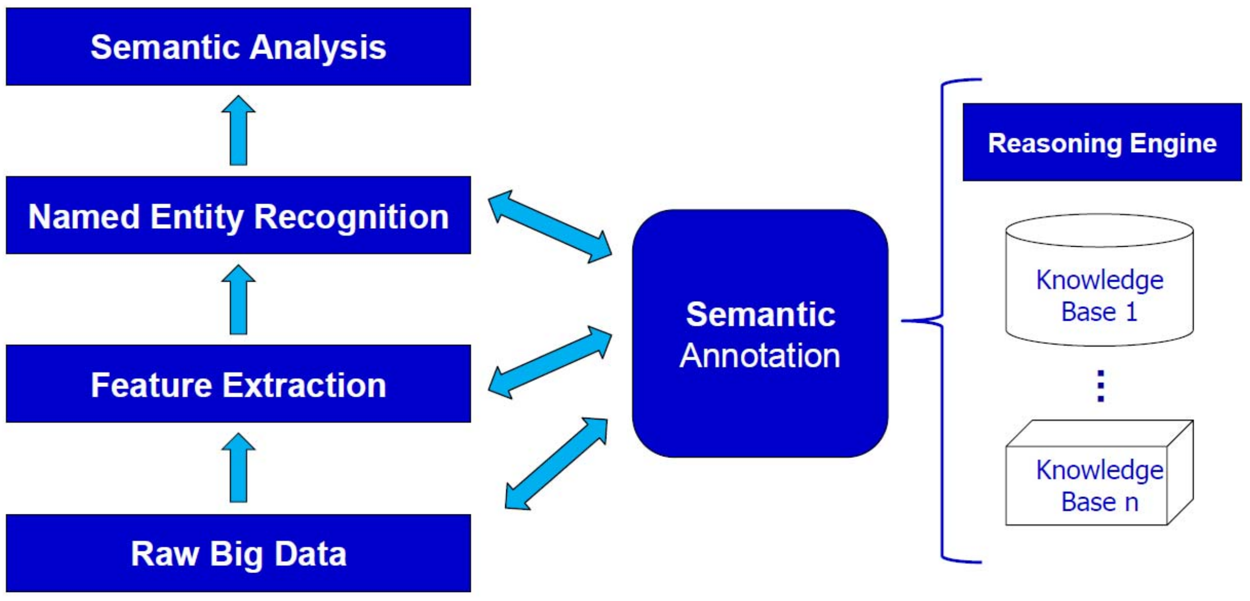


Figure 6-8: Context-Based Reasoning

Context may be modelled as Big Data, since there is an abundance of Big Data, and the critical factor is extracting value from Big Data. The three operations above Big Data in Figure 6-8 describe a set of increasingly specific operations that can be used to semantically annotate information.

A "feature" may be defined as an important element that helps describe and aid in the understanding of an object. For example, edges and corners are points of interest in an image. Feature extraction reduces the number of resources required to describe data. Reducing the number of features is important in analysis and reasoning, since if there are too many features, it could cause overfitting in training.

Named Entity Recognition identities and maps entities that have specific names into models. For example, the sentence "John worked at Cisco in the 1990s" could be analysed, producing three named entities: "John", "Cisco", and "1990s". This could be annotated as "John$_{\text{[person]}}$ worked at Cisco$_{\text{[organization]}}$ in the 1990s$_{\text{[time\_period]}}$".

Semantic Analysis analyses the information for specific semantic concepts, searching on those concepts, and then adding additional semantic relationships to enrich the information and provide more specific meaning. From a linguistic perspective, this analyses text and finds sets of syntactic structures that are related to each other. This may be represented as a graph, or network, of related words, phrases, and other elements of a sentence. From a machine learning perspective, this computes metrics such as semantic similarity (i.e. the meaning of an object compared to the meaning of other objects, where the comparison is done using synonymy, antonymy, hyponymy, hypernymy, and other types of relationships). This is a practical and more computationally tractable approach than "absolute understanding", since the latter requires a rigorous world model, which is np-complete.

Semantic Annotation is the process of providing annotations (either as metadata and/or in a specialized markup) to add meaning to a given object. An example of semantic annotation was given in the named entity recognition description.

6.3.6 Cognition Framework Functional Block

6.3.6.1 Introduction

The cognition framework is a collection of Functional Blocks that relates each of the above four Functional Blocks to the set of end-to-end goals that the ENI System is given. This means that the scope of the cognitive framework is conceptually operating above the scope of both the infrastructure and the other Functional Blocks of the ENI System. It tries to maintain a set of end-to-end goals (such as routing optimizations, connectivity, efficiencies, security, and trust management) by modifying the elements of the infrastructure and the directives of other Functional Blocks.

6.3.6.2 Motivation

The purpose of the Cognition Framework Functional Block is to enable the ENI System to understand ingested data and information, as well as the context and situation that defines how those data and information were produced. Once that understanding is achieved, the cognition framework Functional Block then provides the following functions:

- change existing knowledge and/or add new knowledge corresponding to those data and information;
- perform inferences about the ingested information and data to generate new knowledge;
- use raw data, inferences, and/or historical data to understand what is happening in a particular context and/or situation, why the data were generated, and which entities could be affected; and
- determine if any new actions should be taken to ensure that the goals and objectives of the system will be met.

In each of the four functions above, the cognition framework uses existing knowledge to validate and generate new knowledge. This means that new knowledge may be added, and in some cases, existing knowledge may be changed. Hence, the ENI System uses a *dynamically changing set of repositories* (as opposed to other management systems, which typically use repositories that use *fixed content*). A cognition framework uses multiple diverse processes and technologies, including linguistics, computer science, AI, formal logic, neuroscience, psychology, and philosophy, along with others, to analyse existing knowledge and synthesise new knowledge.

6.3.6.3 Function of the Cognition Framework Functional Block

Cognition seeks to understand concepts in a way similar to how the human brain understands concepts. This is done by using a set of specialized data structures and computational procedures that mimic how the human brain operates. Connectionist theories use this principle to define artificial neural networks. Other approaches, such as those involving formal logic, Bayesian models, and deep learning, provide different algorithms, but are still based on the above premise.

Cognition can be used to process new data and information, along with new inferences, and compare those to previously stored knowledge. The function of this Functional Block is to process and understand goals so that it can institute behaviour that protects and meets those goals. This is done using knowledge and inferencing to explain why input data occurred and how to adapt to it. Intelligent agents [i.8] and [i.9] are examples of entities that exhibit goal-directed behaviour. Other examples are cognitive architectures that solve problems by creating their own sub-goals to solve a problem; such cognitive architectures also learn from their experience.

Any entity that exhibits cognition shall have at least the following three functions:

1) interfaces that interact with the environment providing stimuli;
2) processing that can analyse and manipulate data, information, and knowledge; and
3) memories that hold data, information, and knowledge.

These three functions provide cognitive control and cognitive capabilities, which differentiate a cognitive architecture from other architectures.

Cognitive control includes reflexive and habitual behavior that respond to long-term intentions. Cognitive capabilities include functions that reflect processing as done in the human brain, such as perception, reasoning, learning and planning. Critically, a system that uses cognition shall be able to explain why it acted a certain way in response to stimuli, and more importantly, can learn whether that action was incorrect and, if correct, whether it was optimal.

6.3.6.4 Operation of the Cognition Framework Functional Block (informative)

6.3.6.4.1 Introduction

There are two main approaches to building cognitive systems: symbolic and connectionist. Hybrid architectures, which combine these approaches, are also starting to be researched.

NOTE: These three approaches will be further studied in Release 2 of the present document (see clause 9).

6.3.6.4.2 The Symbolic Approach

This approach (also called computationalism) views cognition as a set of computational processes that act on a set of structures that represent abstract (mental) representations of the world that can be manipulated by symbols. There is a clear separation of cognition into low-level (e.g. sensorimotor) and high-level (e.g. planning and reasoning) processes. This approach builds models of high-level cognition that resemble the structure of the brain (as opposed to connectionist models, which build models that resemble neurological structures). The symbolic models obey rules for processing data and information, as well as for interacting with other symbolic models. This leads to domain-specific symbolic sub-systems for processing different types of data and information (e.g. language processing, reasoning, and planning).

Each of the different domain-specific sub-systems are interconnected, forming a modular architecture. Actions in this architecture can be modelled as operations acting on these sub-systems that collectively form a modular program. The representations and actions taken in the cognitive architecture correspond to the real-world objects and their behavior. In other words, the symbolic abstractions used in the program are kept in sync with the external world. For example, in robotic systems, the sensorimotor processing is responsible for this synchronization.

An AI programming language can be used to construct a cognitive model. This is ideal for knowledge-based problem solving and learning. For example, the AI programming language could be used to construct a search through a problem space, where a problem space defines a set of states and operators that manipulate the states. The solving of a problem is achieved by traversing from an initial state to a final state using a set of operators. An example of such an architecture is provided in [i.11]. In this architecture, a set of IF-THEN (imperative) rules are used to operate on the symbolic representations. When the IF clause is satisfied, the set of actions in the THEN clause are executed. A high-level functional architecture of ACT-R, one example of a symbolic cognitive architecture, is shown below.

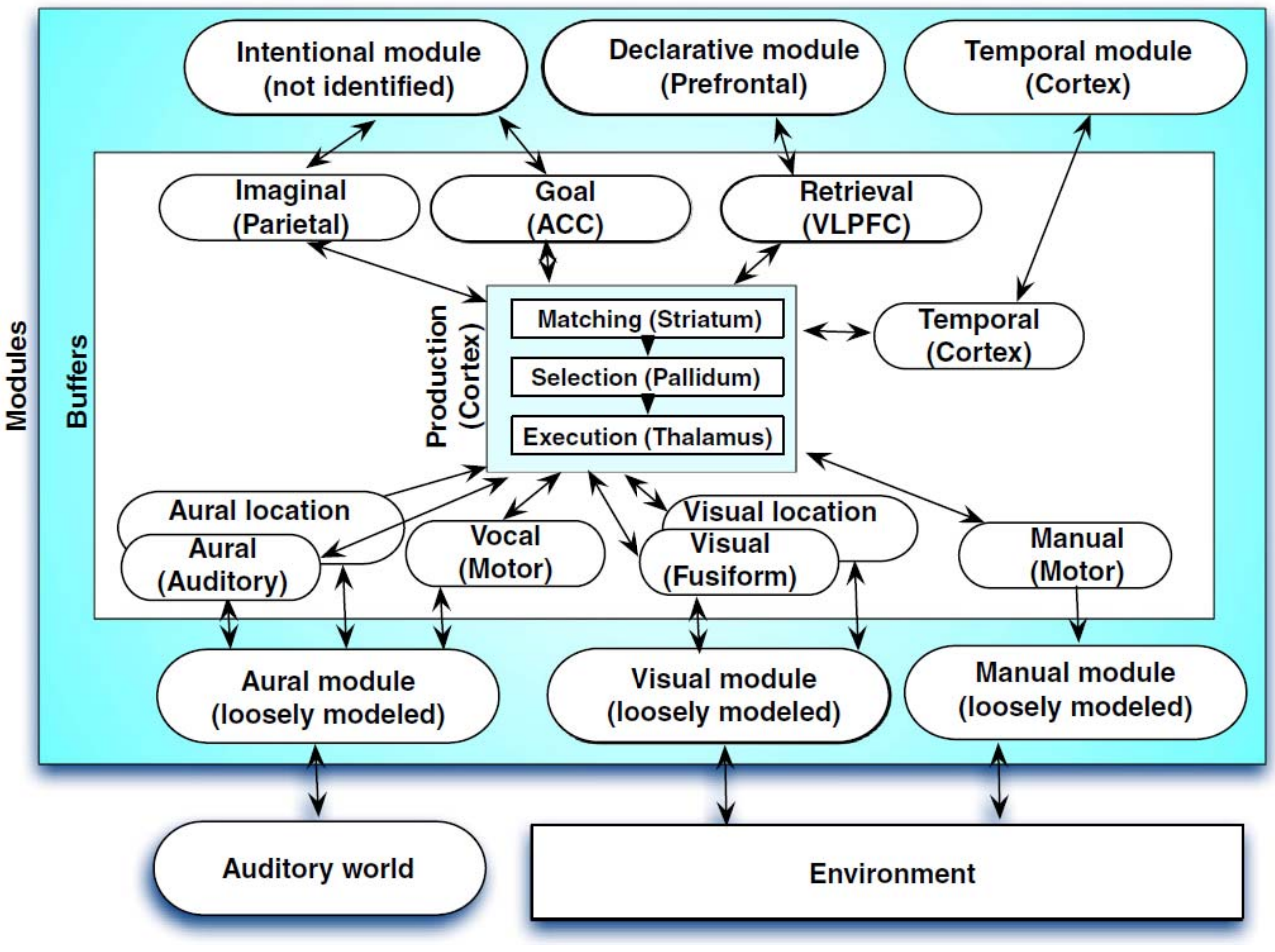


Figure 6-9: The ACT-R Cognitive Architecture

The main modules of ACT-R are:

1) a visual module for identifying objects in the visual field;

2) a goal module for keeping track of current goals and intentions;

3) a declarative module for retrieving information from memory;

4) a manual module for controlling the hands; and

5) a production system for taking actions and coordinating the communication and performance of these modules.

Actions in each of these modules take time, although they work concurrently. The time used within models is based on human performance. This time can be provided as a real-time trace. The default cycle for taking an action in the production system is 50 msec.

Information is passed between the modules and the central production system through buffers, which hold a limited amount of information (this is motivated from psychological and neuroscientific studies). The central production system matches the content of the buffers against the IF portions of its rules, and selects a single rule for execution. The selection mechanism is based on the computation of expected utility of a rule for the current goal and the input. The expected utility is learned from experience using Bayesian processing. A rule can have multiple actions, including updating declarative or goal memory, modifying the value of a buffer, and issuing a command.

6.3.6.4.3 The Connectionist Approach

This approach assumes that all cognitive processes are the same, and are derived from neural activation dynamics. In contrast to the symbolic approach, connectionism argues that mental representations shall not be structured as explicit models, but rather, shall be implicitly encoded in the activation values of neurons. This leads to the parallel processing of many simple modules that are connected as a network. Information is stored in the form of weights between connections.

Knowledge is not provided explicitly to the system, but rather, is learned by the system through the processing of training samples. Learning algorithms extract statistical information from sample pairs of input and output values, and the network adapts its connection weights to approximate the training data. Thus, task execution by connectionist approaches requires suitable training data, and depends on both the network structure methodology and the learning algorithm(s) used. A simplistic Recurrent Neural Network (RNN) is shown in Figure 6-10.

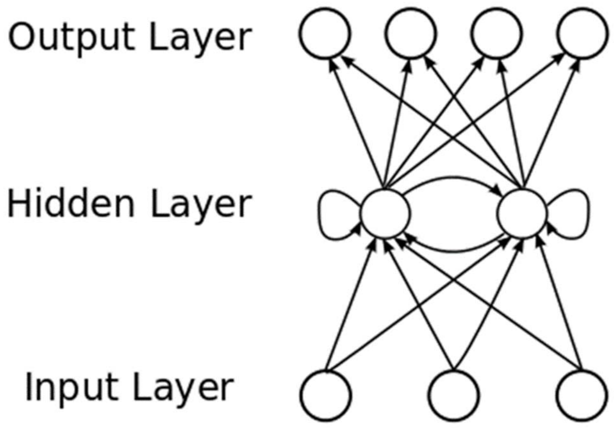


Figure 6-10: A Simplified Cognitive Processing Architecture using Neural Networks

RNNs use looping of the hidden layers back to themselves, which enables them to accept variable length sequences of inputs. RNNs provide a way to process data where time and order are important. For example, with textual data, the ordering of words is important. Changing the order or words can alter the meaning of a sentence. In contrast, in simple feed forward networks, the hidden layer only has access to the current input. It has no "memory" of any other input that was already processed. An RNN, by contrast, is able to "loop" over its inputs and see what has come before. This provides context for processing words that come later in a sentence.

6.3.7 Situational Awareness Functional Block

6.3.7.1 Introduction

This clause describes the motivation for using situational awareness, the effect it has on the System Architecture, and the benefits that it provides. See clause 6.3.5 for a description of the context-awareness Functional Block, and clause 6.3.7.5 for a comparison of this Functional Block to the context-awareness Functional Block.

Situation awareness enables the system to understand what has just happened, what is likely to happen, and how both may affect the goals that the system is trying to achieve. This implies the ability to understand how and why the current situation evolves. ENI shall observe the evolving of different situations, examining them for patterns within each situation and between different situations. Such knowledge shall be stored in the knowledge base of ENI. As such, identifying changes in both the current situation as well as possible future situations are critical for understanding how the environment is changing, and how those changes affect the goals that ENI is trying to achieve or maintain.

For example, security situation awareness could include being aware of the scope and impact of the attack, correlating that with the behaviour of the adversary, so that effective counter-measures can be implemented.

6.3.7.2 Motivation

Networks are fundamentally heterogeneous in nature. Current as well as legacy devices have different software, hardware, and use different protocols. Data may be represented in multiple ways. Hence, there is a need for a common and scalable mechanism to abstract this heterogeneity so that their functionality can be represented in a normalized manner. This would enable common approaches to be developed for enhancing interoperability between heterogeneous systems, letting developers create new applications that use these common characteristics and behaviour.

6.3.7.3 Function of Situational Awareness

The working definition of situation awareness for ENI is:

> *"The perception of data and behaviour that pertain to the relevant circumstances and/or conditions of a system or process ("the situation"), the comprehension of the meaning and significance of these data and behaviours, and how processes, actions, and new situations inferred from these data and processes are likely to evolve in the near future to enable more accurate and fruitful decision-making".*

It consists of five actions: gathering data (perception), understanding the significance of the gathered data (through both facts and inferences), determining what to do (if anything) in response to a given event, making a decision (or set of decisions), and performing those actions. It enables the application of context and policies to a particular situation, and can use inference as well as historical data to understand what is happening at a particular context, why, and what (if anything) should be done in response.

6.3.7.4 Operation of the Situational Awareness Functional Block

A situation shall be determined by the analysis of data and behaviour. The evolution of future situations is a function of understanding the particular context, the factors determining the evolution of that context, and inferring what the future situation will be based on the past and current data and behaviour. Semantics play an important role in understanding the significance and cause of data and behaviour, and shall be used to understand the underlying meaning of data and information that have been ingested.

A high-level functional block diagram of a situational awareness Functional Block is shown in Figure 6-11. The green rounded rectangle represents the Situational Awareness Functional Block; all other rounded rectangles define a Functional Block that is nested within the Situational Awareness Functional Block. The red arrows represent a closed control loop within the Situational Awareness Functional Block.

The functional block diagram shown in Figure 6-11 does not prescribe an implementation. Rather, it describes the high-level Functional Blocks that are needed to implement Situational Awareness. Different implementations may need to add other Functional Blocks to meet their particular operational requirements.

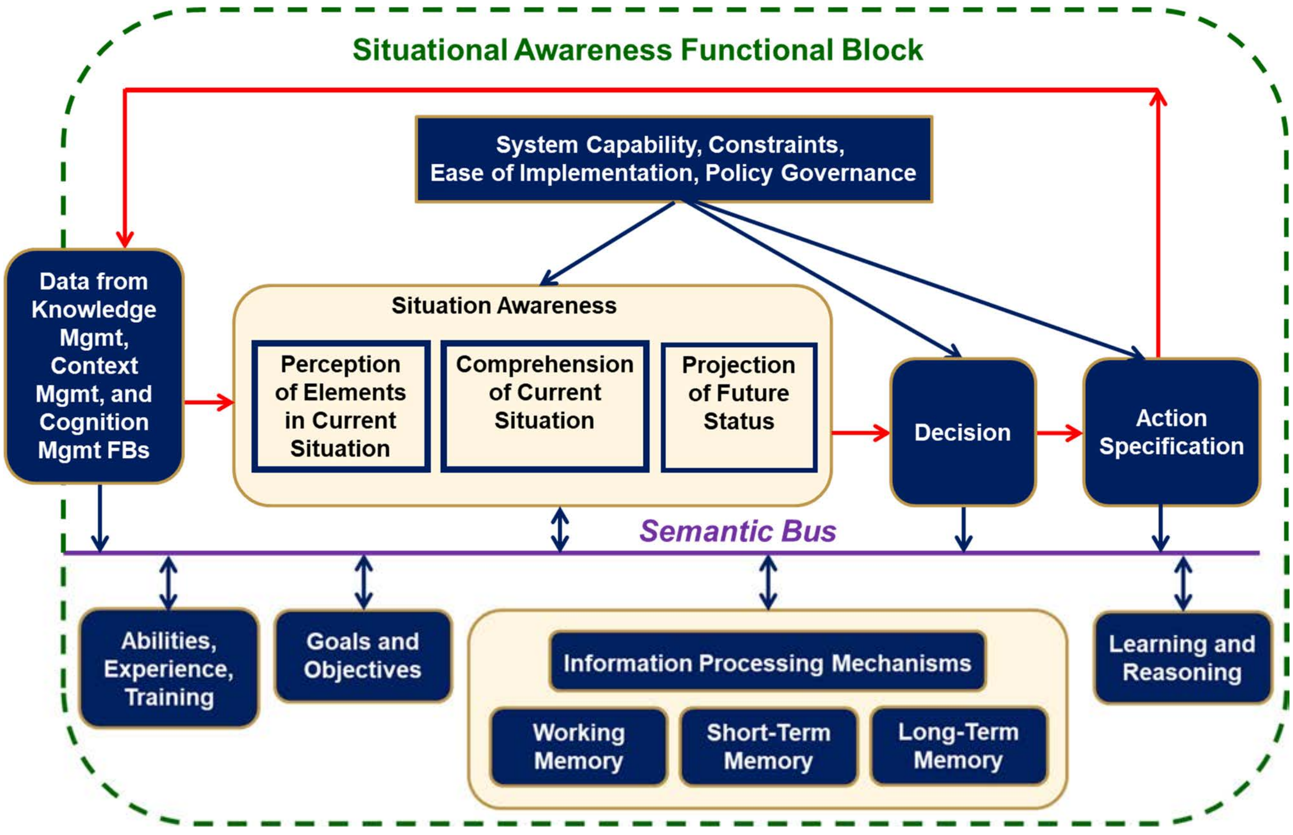

Situational Awareness Functional Block
System Capability, Constraints, Ease of Implementation, Policy Governance
Data from Knowledge Mgmt, Context Mgmt, and Cognition Mgmt FBs
Situation Awareness
Perception of Elements in Current Situation
Comprehension of Current Situation
Projection of Future Status
Decision
Action Specification
Semantic Bus
Abilities, Experience, Training
Goals and Objectives
Information Processing Mechanisms
Working Memory
Short-Term Memory
Long-Term Memory
Learning and Reasoning

Figure 6-11: A Simplified Cognitive Processing Architecture using Neural Networks

Each of the boxes and rounded rectangles in Figure 6-11 may be modelled as a Functional Block. They are defined as follows:

- **Data from Knowledge Management, Context-Aware Management, and Cognition Functional Blocks:**
  - The combination of data and information from this set of Functional Blocks shall serve as input that enables the Situation Awareness nested Functional Block determine the current state of the Assisted System and its Operational Environment.
  - The other main input comes from the Goals and Objectives (see below).
- **System Capabilities, Constraints, Ease of Implementation, Policy Governance:**
  - **System Capabilities** are optional information that define the various functions that the ENI System can perform. They may be specified by metadata.
  - **Constraints** take two forms. External constraints are provided by the Operator or Designated Entity of the Assisted System, and shall specify limitations that shall be obeyed when ENI defines its recommendations and commands. Internal constraints shall be defined by ENI, and represent restrictions that ENI imposes on its decision-making processes in order to achieve a set of goals.
  - **Ease of Implementation** is typically represented as either metadata or using a probabilistic or fuzzy logic mechanism. It enables the Situation Awareness Functional Block to take near-optimal courses of action if taking an optimal course of action is too costly (e.g. in time, resources, or other factors).
  - **Policy Governance** is a set of ENI-generated policies (see clause 6.3.9) that shall be used to manage the operation of the Situation Awareness Functional Block.

- **Situation Awareness:**
  - This defines the ENI System's internal model of the state of the environment. its input comes from the Knowledge Management, Context-Aware Management, and Cognition Functional Blocks and from external and internal goals. The Situation Awareness function shall be separate from decision-making and action specification functions. This is because even if there is a perfect understanding of the situation, incorrect decisions can still be made. The three following functions describe data analysis and fusion performed by this Functional Block.
  - **Perception** produces an awareness of situational elements (e.g. objects, events, people, systems, and environmental factors) and their current states (e.g. modes and locations). Perceived objects may be stored in any of the three types of memories shown in Figure 6-11.
  - **Comprehension** examines the situational elements that have been perceived in order to better understand how they fit together; this helps characterize the situation as a whole, and how this situation affects the goals that the ENI System is trying to achieve.
  - **Projection** is focused on predicting the most likely evolution of the current situation (it is typically too costly and time-consuming to enumerate all possible situations). This model also reflects the differences between working, short-, and long-term memory from cognitive psychology.
- **Decision:**
  - This represents the decision-making capabilities of the Situational Awareness Functional Block. its input is the set of future projected situations, along with goals and other relevant information. its output is the recommended course of action to take, and is sent to the Action Specification nested Functional Block.
- **Action Specification:**
  - This represents the action execution of the Situational Awareness Functional Block. its input is the set of decisions, along with goals and other relevant information. its output is a set of actions to take. These actions are in a format that is native to ENI, and shall be sent to the Model-Driven Engineering Functional Block for translation into policies using the Policy Management Functional Block. The policies produced by the latter are in a form that external entities will understand. This translation may include one or more imperative, declarative, and/or intent policies.
- **Abilities, Experience, Training:**
  - These are represented by metadata, and provide an assessment of how positive the Situational Awareness Functional Block is in performing a particular operation.
- **Goals and Objectives:**
  - Goals and objectives take two forms. External goals are provided by the Operator or Designated Entity of the Assisted System, and specify objectives that the Assisted System wants to achieve. Internal goals are defined by ENI, and represent goals and/or sub-goals that the ENI System has defined in order to make solving the goal easier. Both internal and external goals may be represented by a set of policies.
- **Information Processing Mechanisms:**
  - Information processing analyses and optionally changes ingested information. It is made up of four components: input, storage, processor, and output. The following three types of memory play critical roles in cognition and situational awareness.

NOTE: Information processing is for further study in Release 2 of the present document (see clause 9).

  - **Working Memory** is a memory of limited capacity for temporary storage of information. Information may be manipulated and transformed in a working memory. It consists of a number of different sub-systems, where each sub-system is optimized for a particular category of information. It is used to analyse information before committing it to short- or long-term memory.
  - **Short-Term Memory** is a memory of limited capacity for temporarily holding, but not manipulating, information. It is a cognitive memory that holds sensory events and concepts that have significance to the managed environment, such as names, numbers, and other managed objects.

  - **Long-Term Memory** is a memory of moderate to large capacity where information may be indefinitely held. Long-term memory examines information held in short-term memory and semantically augments it. Repeated occurrences of the same information in short-term memory cause that information to be strengthened (e.g. made more important and certain) in long-term memory. Different types of long-term memory exist; this is beyond the scope of the present document.

- **Learning and Reasoning:**

  - Learning is the process of acquiring new, or modifying existing, data, information, or knowledge. Reasoning is the process of understanding stimuli and the environment, verifying facts, making inferences, applying a decision-making mechanism (e.g. logic), and then implementing a set of actions to induce change. Both learning and reasoning are defined by different types of AI algorithms. Some important examples of learning include:

    - Non-associative learning is the strengthening of a response to a given stimulus due to repeated exposure to that stimulus.

    - Associative learning is the process of learning an association between different stimuli.

    - Episodic learning is the production of a change in behaviour as a result of one or more events.

Data fusion is necessitated when heterogeneous systems are required to interoperate in an open world, where the syntax and semantics of data provided by a sensor or a human is domain-specific and does not conform to any one specific vocabulary. This requires the translation of each vocabulary used into a common set of concepts and terms, so that the data can be integrated. The ENI System shall perform data fusion in order to associate different definitions of the same concept with each other; this is used to provide a more comprehensive understanding of situations.

A situation reflects an entity's contextual view of a collection of data and processes at a particular instance in time. Shared situational awareness is therefore a consensus view of a number of individual views that each describes the same situation. The ENI System may use any distributed mechanisms, such as agents, to realize shared situational awareness when needed.

6.3.7.5 Difference between Context Awareness and Situational Awareness

The Context of an Entity describe the state and environment in which an Entity exists or has existed; it uses a combination of historical data, as well as facts and inferences to do this. In contrast, situational awareness includes contextual information and other inputs in order to understand the meaning and significance of data and behaviour of the entire Assisted System and its operational environment; more importantly, situational awareness includes a prediction of the evolution of the situation, and how that evolution affects the goals that the ENI System is trying to achieve. Hence, context is one aspect of situational awareness.

6.3.8 Model Driven Engineering Functional Block

6.3.8.1 Introduction

The Model Driven Engineering (MDE) Functional Block is responsible for enabling software development to be accomplished using models instead of code. The advantage of MDE is that models are, by definition, machine-readable. Hence, they can be used to specify Functional Blocks, programs, and applications. An example of MDE is to generate code directly from a model.

MDE represents an approach to software development where models are used in the understanding, design, implementation, deployment, operation, maintenance and modification of software systems. A set of models may be defined based on different viewpoints. Formally, a viewpoint is an abstraction of the function and behaviour of a system using a selected set of architectural concepts; this facilitates focusing on a particular aspect or set of responsibilities of the system. Model transformation tools and services are used to align the different models (e.g. deriving a set of data models from an information model), and for generating code

6.3.8.2 Motivation

Software systems continue to grow in complexity. While modelling is commonly used, models are often only used for idea generation and design, and are not linked to implementation. Worse, models are not updated as frequently as code, which increases the separation between the models and the implementation.

One of the original reasons for using models was that concepts that were more familiar to domain experts could be more easily represented in a way that those experts could understand (as opposed to having to know how to write and debug code). Furthermore, this was deemed an easier and more straightforward way of specifying business log that was independent of the platform and technology used. [i.13] summarizes the reason for modelling:

"Modelling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality"."

Thus, the motivation for using MDE is to focus on the business logic, not code, by using a methodology that uses abstractions (instead of, for example, software libraries and function calls).

6.3.8.3 Function of the Model Driven Engineering Functional Block

The function of the MDE Functional Block is to decide how to implement the selected actions from the Situational Awareness Functional Block. It uses model-driven engineering mechanisms to convert the actions into a form that enables imperative, declarative, and/or intent policies to be constructed (by the Policy Management Functional Block). More specifically, information and data models shall represent key grammatical concepts (e.g. nouns and verbs) of policies, as well as other concepts that a policy grammar can refer to. Therefore, use of the model serves three important purposes:

1) It ensures that all different data models used in the ENI System maintain a consistent definition and understanding of concepts, even if a concept is represented using different data structures in different parts of the ENI System.

2) It enables different policies at different levels of abstraction to communicate with each other using a common vocabulary and data dictionary.

3) It decouples the need for policy (defined by the Situation Awareness Functional Block) from the specification of policy (defined in the MDE Functional Block) from the implementation of policy (defined in the Policy Management Functional Block).

6.3.8.4 Operation of the Model Driven Engineering Functional Block

A high-level functional block diagram of an MDE Functional Block is shown in Figure 6-12. The functional block diagram shown in Figure 6-12 does not prescribe an implementation. Rather, it describes the high-level Functional Blocks that are needed to implement the needs of model-driven engineering. Different implementations may need to add other Functional Blocks to meet their particular operational requirements.

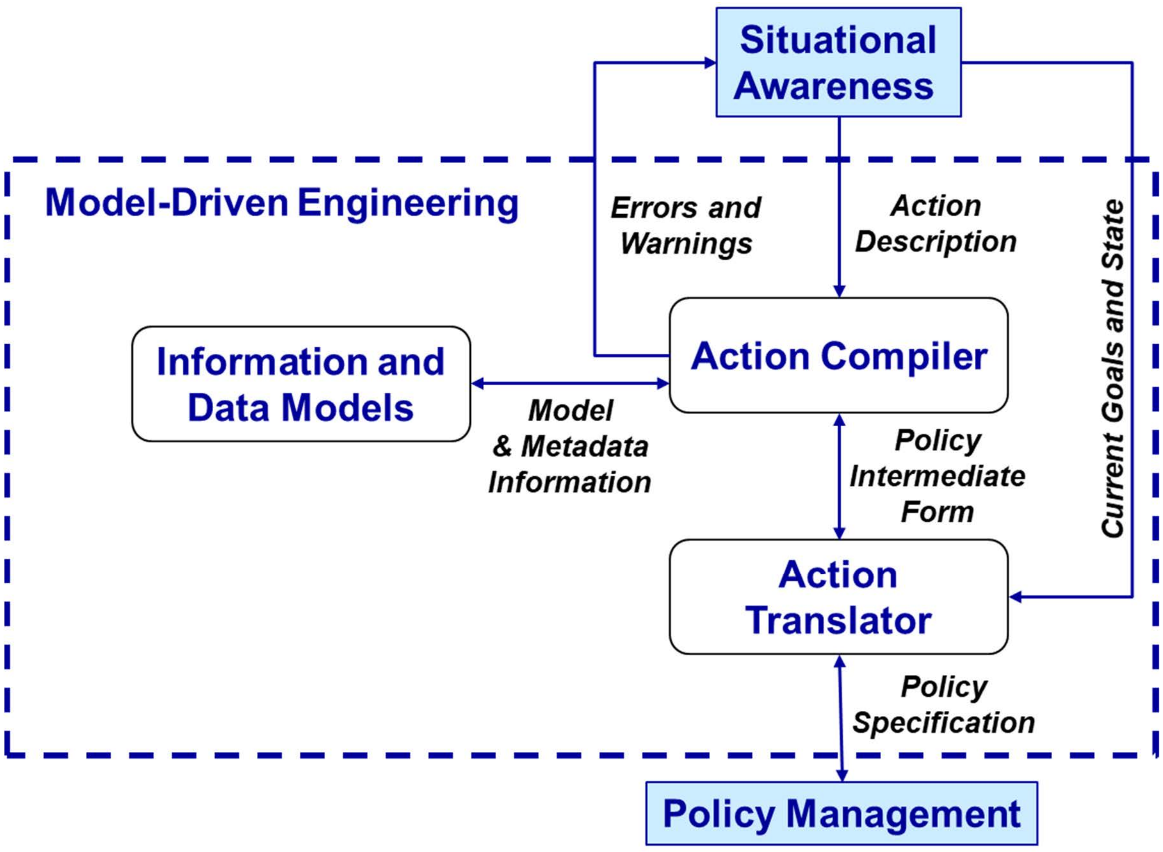


Figure 6-12: A Simplified Functional Architecture of the MDE Functional Block

A high-level description of the flow of operations represented by Figure 6-12 is as follows. First, the action descriptions (defined by the Situational Awareness Functional Block) are received by the Action Compiler. If any warnings or errors are found, feedback will be given to the Situational Awareness Functional Block, and the parsing or compiling will cease. Otherwise, the Action Compiler will produce a Policy Intermediate Form, which is the first step in translating the action descriptions to formal Policies to be sent to the Assisted System or its Designated Entity. The Policy Intermediate Form will be further refined by the Action Translator, and specific knowledge about the relationship between the goals of the system and its state are embedded in each policy. This simplifies subsequent evaluation of the efficacy of the set of Policies produced. Additional details are provided in the text below.

The set of actions to take are defined by the Situation Awareness Functional Block, as described in clause 6.3.7, and are sent to the Model-Driven Engineering Functional Block. These action descriptions shall consist of text, metadata, and/or modelled objects. The MDE Functional Block shall translate these inputs into a textual description and output that text to the Policy Management Functional Block.

The MDE Functional Block contains multiple active repositories. The two most important are the Information Model Repository and the Data Model Repository. The Information Model Repository shall contain the authoritative information model of record, and may contain one or more copies of that information model that are used to process and evaluate changes to be made to the information model of record. Changes are made to the information model of record to reflect new information that has been learned, including changes to existing information. Any changed information shall be verified by at least one other independent ENI Functional Block before a change is made to the information model of record.

The information model of record shall be used as the authoritative version of the model by all other ENI Functional Blocks. All data models used by ENI Functional Blocks shall be derived from the information model of record.

The Data Model Repository shall contain one or more data models. Each ENI Functional Block may use one or more data models to suit its needs. For example, an LDAP directory and an RDBMS may be used to store information about the same or different managed objects. Each data model shall be optimized to reflect the needs of the Functional Block that is using it. For example, a data model may flatten a set of objects defined in the information model of record to produce a single class that contains all of the attributes, operations, constraints, and behaviours of the information model classes; this may be done to increase and simplify access to data and behaviour. As another example, a data model may restructure and translate data into different data structures (e.g. in order to facilitate protocol operations), as long as in so doing, it does not alter the meaning of the original data.

The information and data models are used by the Action Compiler to help parse action descriptions received by the Situational Awareness Functional Block. The parsing verifies the structure and meaning of the action descriptions, and ensures that the MDE Functional Block understands the actions that the Situational Awareness Functional Block wants to take. The first part of this process is to verify the syntax used in the action description. For example, a Customer is a type of Named Entity (i.e. it shall exist in a data model as an instance of a Customer object); if that is not the case, then an error is produced. Any errors and warnings found are sent back to the Situational Awareness Functional Block, along with as much context as the Action Compiler can provide. Continuing the Customer example, the Action Compiler could annotate that the usage of a term indicates that it could possibly be a Customer, but that no Customer with such a name was found. Two mechanisms of doing this parsing are (1) using modelled objects to help understand words and phrases in the action description, and (2) using the models to build an (internal) language, whose syntax and semantics are formally defined, and then using the syntax and semantics of that formal language to compile the action descriptions.

NOTE: These and other methods for parsing and/or compiling the action descriptions are for further study in Release 2 of the present document (see clause 9).

Once the action descriptions are parsed or compiled, they are translated into what is known as the Policy Intermediate Form. The Policy Intermediate Form is a temporary internal format that is the start of defining a set of policies that represent the formal policy translation of the action descriptions sent by the Situational Awareness Functional Block. Formally, this is a compilation stage that connects the initial actions of the compiler (in the MDE Functional Block) to their final output form (in the Policy Management Functional Block). The Policy Intermediate Form may be structured in a tree or graph format. This Policy Intermediate Form is then sent to the Action Translator, which compares the Policy Intermediate Form with the active goals of the system; it then embeds information into each Policy (or set of Policies) to be constructed that describes how that Policy is related to one or more goals that the system is trying to achieve. This relationship may be embedded in each Policy using metadata or other similar mechanisms. These Policy Definitions take the form of text, and are output to the Policy Management Functional Block, which completes the compilation process by producing a set of imperative, declarative, and/or intent policies for each action description produced by the Situational Awareness Functional Block.

6.3.9 Policy Management Functional Block

6.3.9.1 Introduction

This clause describes the motivation for using policy management, the effect it has on the System Architecture, and the benefits that it provides. This Functional Block provides a set of uniform and intuitive mechanisms for providing consistent recommendations and commands. These characteristics give rise to the following requirements:

- ENI shall provide the ability to transform data and information from its own internal format to format that facilitates generating outputs that are understandable by the Assisted System and/or its Designated Entity.
- ENI shall use a set of models, including data types and data structures for producing outputs that are understandable by the Assisted System and/or its Designated Entity.
- Data Denormalization may be realized as a Functional Block that is separate from the Output Generation Functional Block. This adheres to the Single Responsibility Principle [i.10] and enables a more scalable and robust system to be designed and built.

6.3.9.2 Motivation

Management involves monitoring the activity of a system, making decisions about how the system is acting, and performing control actions to modify the behaviour of the system. The purpose of Policy Management is to ensure that consistent and scalable decisions are made governing the behaviour of a system.

Organizations are policy-driven entities. Policy is a natural way to express rules and restrictions on behaviour, and then automate the enforcement of those rules and restrictions. However, the number of policies can be very large (e.g. 100 000+), and the relationships between policies can be complex. In addition, policy can change *contextually*. For example, different actions can be taken based on type of connection, time of day, and network state.

This project will use the following definition of Policy:

Policy is a set of rules that is used to manage and control the changing and/or maintaining of the state of one or more managed objects [i.17], [7] and [8]. Policy is a mechanism for controlling the behaviour of an Entity, not the actual end result. For example, an access control list is created and managed using policy, but is not a policy instance or type of policy.

Policy is not absolute. The actions of a policy shall be verified. In addition, a goal of ENI is to continually evaluate and optimize policy, so that it becomes more effective with experience.

The size and complexity of modern systems has resulted in the need for automating management operations. If Policies are coded into a management component, their lifecycle becomes interlocked with that component, and their behavior can only be altered by recoding the component. Hence, Policies should be defined and management independent from management components to enable policies to be changed and reused without affecting the lifecycle of the management system. It also enables the Policies to adapt to evolutionary changes in the system being managed, as well as to accommodate new application requirements. **Ultimately, the business and operational policies that govern the construction and deployment of configuration changes are more important than the configuration changes themselves!**

6.3.9.3 Function of the Policy Management Functional Block

As described in [i.17] and [i.2], there are three different types of policies that are defined for an ENI System:

Imperative policy: a type of policy that uses statements to explicitly change the state of a set of targeted objects. Hence, the order of statements that make up the policy is explicitly defined. An example of an imperative policy, using informal English, is:

WHEN an Alarm is received
IF the severity of the Alarm is Critical
THEN execute the CriticalAlarm Policy

In the present document, Imperative Policy will refer to policies that are made up of Event, Condition, and Action clauses.

Declarative policy: a type of policy that uses statements to describe a set of computations that need to be done without defining how to execute those computations. Hence, state is not explicitly manipulated, and the order of statements that make up the policy is irrelevant. An example of a declarative policy, using First Order Logic, is:

$\exists x \exists y\ (Customer(x) \land SLA(y) \land have(x, y))$

The English equivalent is:

Some Customers have an SLA

In the present document, Declarative Policy will refer to policies that execute as theories of a formal logic. The syntax of a declarative policy typically uses some type of first order logic.

Intent policy: a type of policy that uses statements from a restricted natural language to express the goals of the policy, but not how to accomplish those goals. In particular, formal logic syntax is not used. Therefore, each statement in an Intent Policy may require the translation of one or more of its terms to a form that another managed functional entity can understand. An example of an intent policy is:

No processor shall run at more than 75 % utilization

In the present document, Intent Policy will refer to policies that do not execute as theories of a formal logic. They typically are expressed in a restricted natural language, and require a mapping to a form understandable by other managed functional entities.

The above example indicates different types of ambiguity that may exist in an intent statement. For example, does the term "processor" include both CPUs and GPUs? What about ASICs that have processing capabilities? As another example, the term "utilization" could refer to memory, I/O operations, or processor utilization.

An ENI System may use any combination of imperative, declarative, and intent policies to express recommendations and commands to be issued to the system that it is assisting and/or managing.

6.3.9.4 Operation of the Policy Management Functional Block

Figure 6-13 illustrates a key concept of Policy, called the Policy Continuum [i.17], [7] and [8].

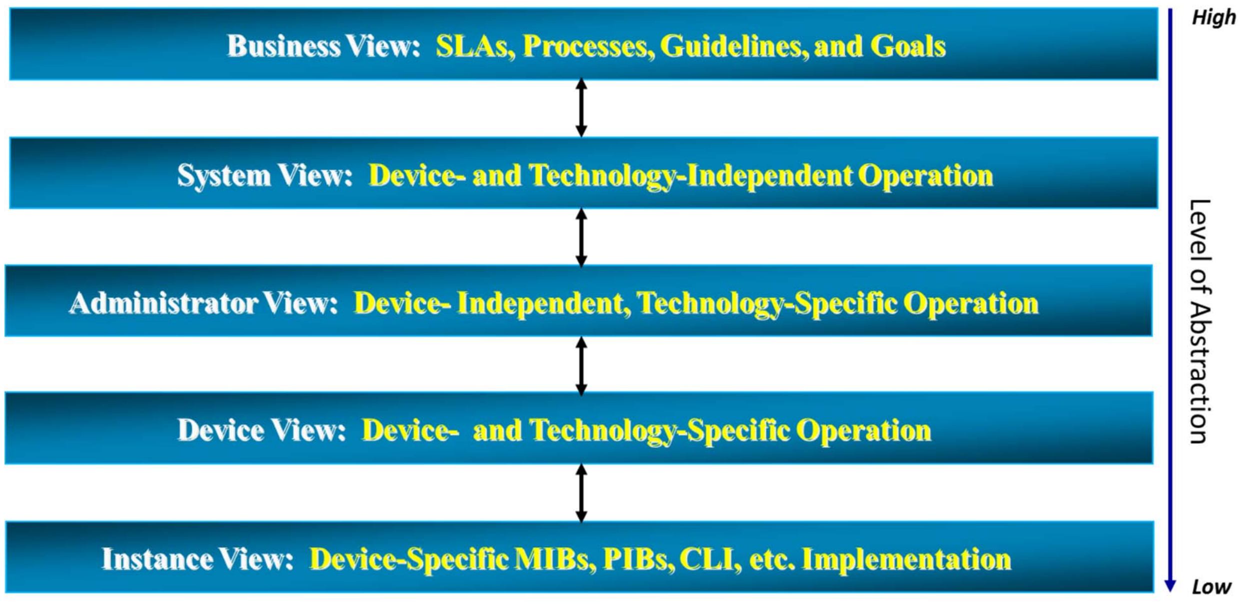

Business View: SLAs, Processes, Guidelines, and Goals
System View: Device- and Technology-Independent Operation
Administrator View: Device- Independent, Technology-Specific Operation
Device View: Device- and Technology-Specific Operation
Instance View: Device-Specific MIBs, PIBs, CLI, etc. Implementation
High
Level of Abstraction
Low

Figure 6-13: The Policy Continuum

The purpose of the Policy Continuum is to formally differentiate between the needs of different constituencies in defining and expressing policy. Each constituency is made up of a set of users that have similar business needs, and more importantly, use similar concepts and terminology. For example, business users and product managers use significantly different terminology than application developers or network administrators. The number of continua in the Policy Continuum shall be determined by the applications using it. There is no fixed number of continua. Figure 6-13 shows five, because this enables a set of much smaller translations of terms (e.g. from a representation without technology, to one with technology while being device, vendor, and technology independent, to successively lower levels that fix each of these three dimensions). However, Figure 6-13 is used to illustrate the principles of the Policy Continuum, not to define the type or number of continua used in ENI.

Figure 6-14 shows a simplified functional architecture of the Policy Management Functional Block. The functional block diagram shown in Figure 6-14 does not prescribe an implementation. Rather, it describes the high-level Functional Blocks that are needed to implement the needs of policy-based management in a given administrative domain. Different implementations may need to add other Functional Blocks to meet their particular operational requirements. An exemplary implementation is described in [i.12].

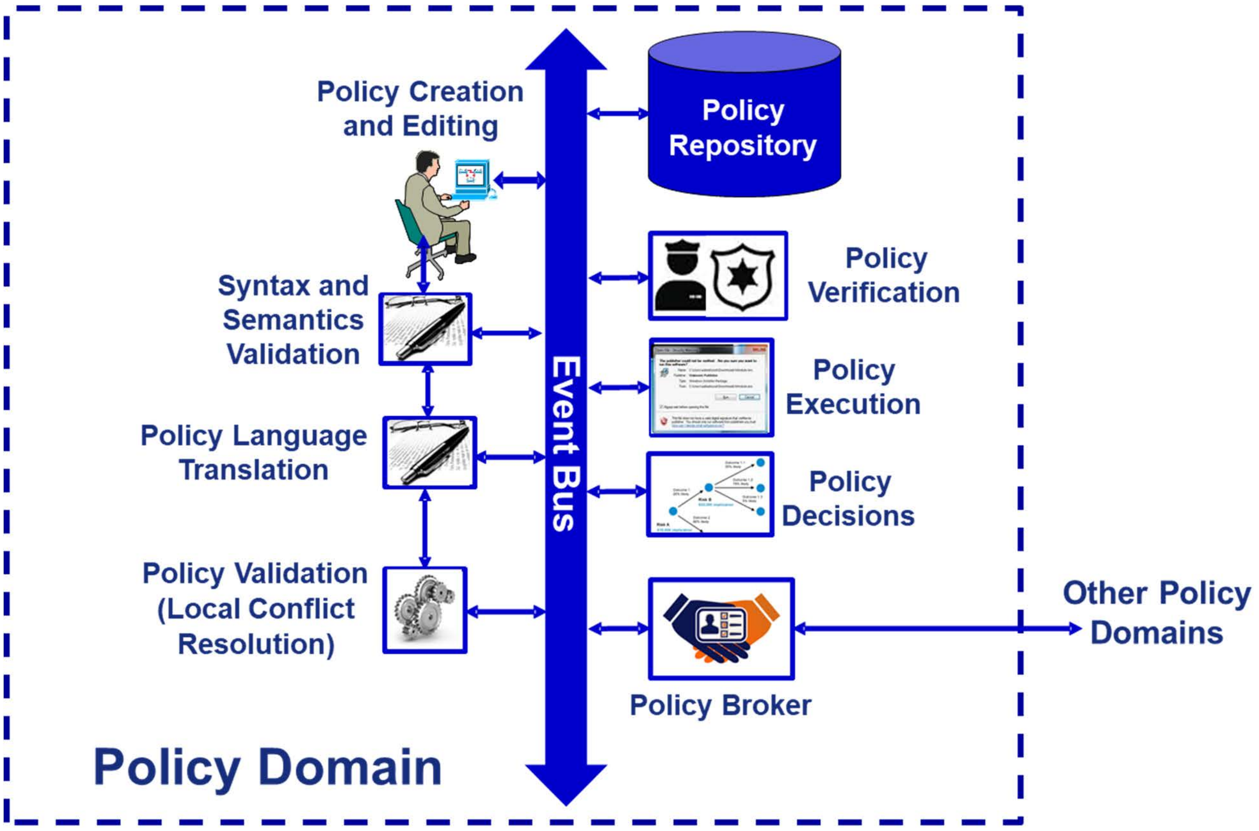

Policy Creation and Editing
Syntax and Semantics Validation
Policy Language Translation
Policy Validation (Local Conflict Resolution)
Event Bus
Policy Repository
Policy Verification
Policy Execution
Policy Decisions
Policy Broker
Other Policy Domains
Policy Domain

Figure 6-14: A Simplified Functional Architecture of the Policy Management Functional Block

Any combination of textual, graphical, and/or command-line tools may be used to author any type of Policy. The Policy Management Functional Block (PMFB) may assist in helping the user to author and edit policies. For example, the PMFB may identify errors, such as incorrect syntax, to the user, before and after the user submits the policy to the ENI System. An event may be created when the user is finished authoring or editing the policy and submits it to the ENI System.

Once a completed policy is received by the ENI System, that policy may be saved in a Policy Repository. A completed policy shall then be either parsed or compiled. In either case, the parsing or compilation shall first check for correct syntax and semantics. Any warnings or errors shall be sent back to the user to correct. Operation on a policy with errors and/or warnings shall not proceed until those errors and/or warnings are fixed.

Once a policy has been successfully parsed or compiled, additional operations may be necessary before it can be deployed, depending on the type of policy and the type of actions to be performed:

- **Policy Language Translation:**
  - Sometimes a policy may need to be translated from one form to another form. The most common case is when a policy is input at a high-level of abstraction and needs to be translated to a more concrete form in order to be validated and processed. This shall be done for all intent policies that are input to the ENI System, since by definition, an intent policy is written in a Controlled Language.
- **Policy Validation - Local Conflict Resolution:**
  - Local Conflict Resolution ensures that a new policy shall not conflict with any deployed policies. This ensures that policy management is always deterministic.
  - A policy is said to conflict with another policy if the actions of a policy cause the behaviour of other policies to be changed during the same execution scope. For example, within the same execution scope, if a Policy Foo sets an attribute bar to 1, and a subsequent policy sets the same attribute bar to a different value, that is a conflict. This subject is treated more thoroughly in [i.17], [7] and [8].
  - Local conflict resolution ensures that no actions in a set of policies conflict when they are executed.

- **Policy Language Validation - Global Conflict Resolution:**
  - Global Conflict Resolution ensures that policies that execute in different Policy Domains shall not conflict with each other.
  - Policy Domains communicate with other Policy Domains through a special mechanism called a Policy Broker. This entity facilitates communication between Policy Domains that are collaborators, consumers, and producers:
    - Two Policy Domains collaborate when the actions taken by both Policy Domains affect the same set of entities in a non-conflicting manner.
    - A Policy Domain C is a consumer of a Policy Domain P when one or more policies of Policy Domain P are used by Policy Domain C as part of its execution. Similarly, in this example, Policy Domain P is said to be a producer for Policy Domain C.
- **Policy Decision Entity:**
  - A Policy Decision Entity is a managed entity in a Policy Management Domain that is responsible for deciding which, if any, policies shall be executed in response to a request by another managed entity for a set of governance actions.
- **Policy Execution Entity:**
  - A Policy Execution Entity is a managed entity in a Policy Management Domain that is responsible for managing the execution of a set of Policies.
- **Policy Verification Entity:**
  - A Policy Verification Entity is a managed entity in a Policy Management Domain that is responsible for verifying that a Policy that was executed by a Policy Execution Entity operated as expected.
- **Policy Broker:**
  - The Policy Broker is responsible for facilitating communication between different Policy Domains.

NOTE: This subject is for further study in Release 2 of the present document (see clause 9).

Figure 6-14 shows that each Policy Domain may provide its own closed control loop. This may start at any one of the nested Functional Blocks (e.g. the Syntax and Semantics Validation) and return to that nest Functional Block via the Event Bus. In addition, the combination of Policy Decision Entity, Policy Execution Entity, and Policy Verification Entity provide feedback for any decisions made; this feedback may be logged by the Policy Domain. Once logged, this feedback may be published to other Policy Domains to aid their decision-making.

6.3.10 Denormalization Functional Block

6.3.10.1 Introduction

The Data Denormalization Functional Block receives processed data from the Knowledge Management Functional Block. These data and information are in one or more ENI internal formats. The Data Denormalization Functional Block then translates these data and information into a form that the Assisted System (and/or its Designated Entity) may understand.

These characteristics give rise to the following requirements:

- ENI shall provide the ability to transform recommendations and commands into a form that the Assisted System (and/or its Designated Entity) may understand.
- ENI shall use a set of models, including data types and data structures, to perform the transformation into native format(s) used by the Assisted System (and/or its Designated Entity).
- Data Denormalization may be realized as a Functional Block that is separate from the Output Generation Functional Block. This adheres to the Single Responsibility Principle [i.10], and enables a more scalable and robust system to be designed and built.

6.3.10.2 Motivation

Each ENI Functional Block may use different data structures and representation of information that is specific to its own processing. Such data structures and representation may be different in nature to the data structures and representation used by other ENI Functional Blocks. In all cases, these data structures and representations are internal to ENI for efficiency of internal understanding and operation. Therefore, each ENI Functional Block may use different programming languages and protocols, which means that the same data may be represented in completely different formats. For example, the same data about a customer may be processed using relational databases and directories. It is important to consolidate all changes to information and data needed by the Assisted System (and/or its Designated Entity); this ensures that conflicting commands and recommendations are not sent to the Assisted System (and/or its Designated Entity).

In this example, recognizing that the object is a Customer, even though the data is different, may be done by comparing the ingested data with expected data from the model. For example, the Customer object in an information model has a set of mandatory (and optional) attributes, as well as relationships to other objects. It may also have metadata that disambiguates this instance of a Customer from all other instances of a Customer. The Denormalization process arranges and formats the data and information to be output so that it may be more easily and efficiently translated into a form that is understandable by the Assisted System (and/or its Designated Entity).

6.3.10.3 Function of the Denormalization Functional Block

As stated previously, the ENI System may use one or more internal formats to represent, analyse, and process data and information. The ENI System shall not expect the Assisted System (and/or its Designated Entity) to be able to understand the internal format(s) of ENI. Data Denormalization translates information and data in one or more internal formats used by the ENI System to a standardized form; this facilitates the generation of recommendations and commands in a form that the Assisted System (and/or its Designated Entity) is able to understand.

A normalized form is critical for the operation of other ENI Functional Blocks. A normalized form means that every ENI Functional Block may be designed using knowledge about the characteristics and behaviour of pertinent data and objects. Otherwise, each ENI Functional Block would have to accommodate data that could be represented in different formats using different data structures. Similarly, a denormalized form is critical for the Assisted System (and/or its Designated Entity) to be able to understand recommendations and commands given to it by ENI without having to be modified. This lack of modification is important to facilitate early adoption of ENI. The Assisted System (and/or its Designated Entity) need not be able to understand recommendations and commands in the internal format(s) of the ENI System. Therefore, a denormalized form is critical for enabling the ENI System to produce outputs in a form that the Assisted System (and/or its Designated Entity are able to understand without any additional assistance. The purpose of the Denormalization Functional Block is to decouple the internal format(s) used by the ENI System from the native format(s) used by the Assisted System (and/or its Designated Entity); this enables both Systems to change independently without affecting the operation of the other.

6.3.10.4 Operation of the Denormalization Functional Block

Data and information Denormalization translate data and information from one or more internal forms to a single standardized form. The denormalization process includes the use of pre-defined data structures, where data is converted to a standard format that uses a standard encoding. In the case of ENI, this is facilitated using its set of models. In ENI, models represent objects as templates; this includes defining a set of mandatory and optional attributes, operations, relationships, and other standard features. Mertadata may be used to help describe and/or prescribe this process.

In order to do this, accumulated knowledge from other ENI Functional Blocks (included predefined model information) shall be made available to the Denormalization Functional Block, as shown in Figure 6-15. This is essentially the inverse of the example provided in clause 6.3.3.4. The functional block diagram shown in Figure 6-15 does not prescribe an implementation. Rather, it describes the high-level Functional Blocks that are needed to implement the needs of Denormalization. Different implementations may need to add other Functional Blocks to meet their particular operational requirements.

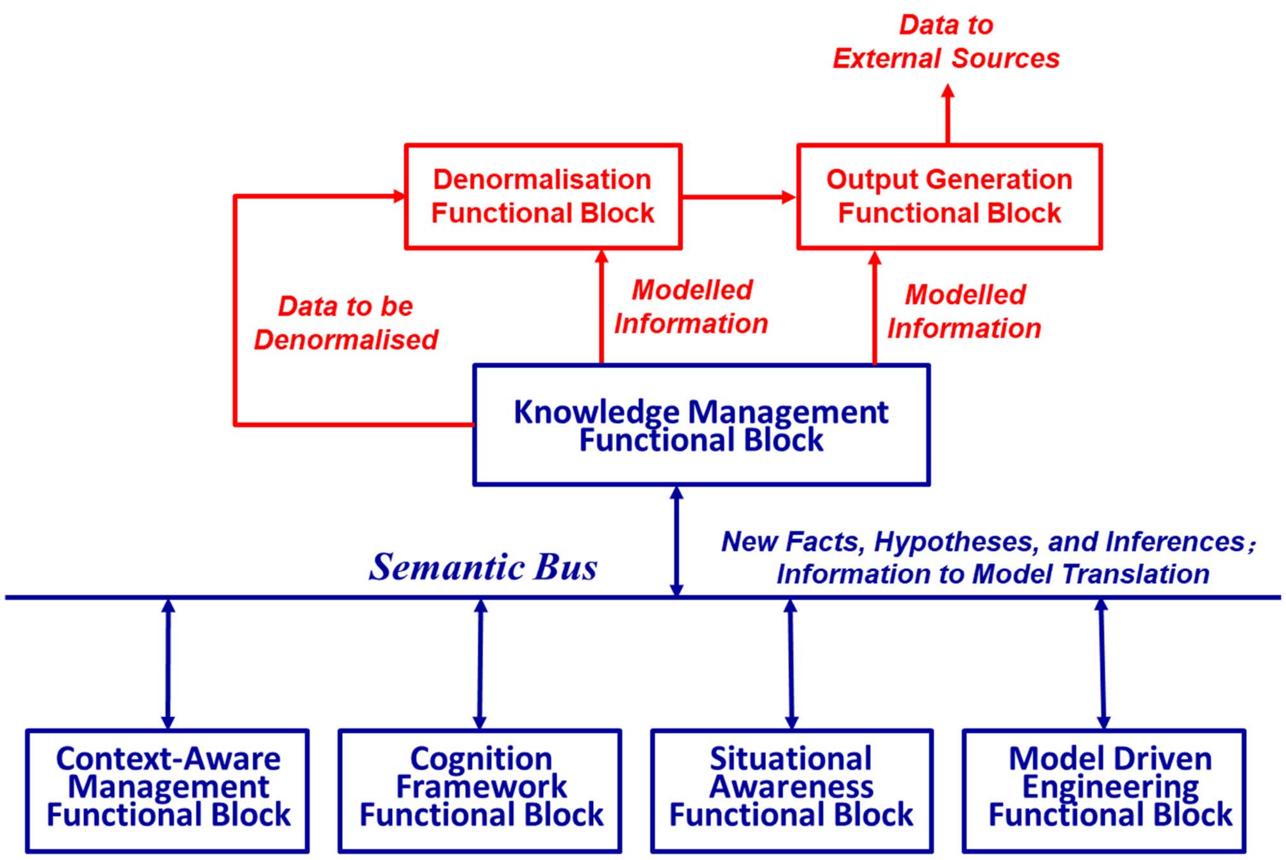

Data to External Sources
Denormalisation Functional Block
Output Generation Functional Block
Data to be Denormalised
Modelled Information
Modelled Information
Knowledge Management Functional Block
Semantic Bus
New Facts, Hypotheses, and Inferences; Information to Model Translation
Context-Aware Management Functional Block
Cognition Framework Functional Block
Situational Awareness Functional Block
Model Driven Engineering Functional Block

Figure 6-15: Data Denormalization Operation

Applicable data and information are not sent directly to the Denormalization Functional Block. Rather, they are first filtered by the Knowledge Management Functional Block, which also supplies modelled data and information to aid the Denormalization Functional Block in denormalizing the data and information that it receives. After the denormalization process is complete, it sends its result to the Output Generation Functional Block.

6.3.11 Output Generation Functional Block

6.3.11.1 Introduction

This clause describes the processes associated with generating recommendations and commands that are specific to the format(s) of a particular Assisted System (and/or its Designated Entity). Recommendations and commands are received from the Data Denormalization Functional Block. The Output Generation Functional Block then translates the data into a specific format (or set of formats) required by the Assisted System (and/or its Designated Entity). This is essentially the inverse of the example provided in clause 6.3.2.4.

These characteristics give rise to the following requirements:

- ENI shall provide the ability to output recommendations and commands in one or more formats specified by the Assisted System (and/or its Designated Entity). This may be efficiently implemented using a multi-agent architecture.
- ENI shall provide the ability to output recommendations and commands on-demand as well as in batch, push, or other agreed upon communication modes.
- Output Generation may be realized as a Functional Block that is separate from the Data Denormalization Functional Block. This adheres to the Single Responsibility Principle [i.10], and enables a more scalable and robust system to be designed and built.

6.3.11.2 Motivation

The motivation for this Functional Block is similar to that of the Denormalization Functional Block (see clause 6.3.10.2). It is important that the Assisted System (and/or its Designated Entity) are decoupled from the ENI System; this enables each to change independent of the other. In addition, it is important that the Assisted System (and/or its Designated Entity) need not have to change in order to benefit from interacting with the ENI System. Hence, the function of the Output Generation Functional Block is to translate recommendations and commands from the ENI System to the form(s) specified by the Assisted System (and/or its Designated Entity).

6.3.11.3 Function of the Output Generation Functional Block

As stated previously, the ENI System may use one or more internal formats to represent, analyse, and process data and information. The ENI System shall not expect the Assisted System (and/or its Designated Entity) to be able to understand the internal format(s) of ENI. Data Denormalization translates information and data in one or more internal formats used by the ENI System to a standardized form; this facilitates the generation of recommendations and commands in a form that the Assisted System (and/or its Designated Entity) is able to understand.

6.3.11.4 Operation of the Output Generation Functional Block

Figure 6-16 shows one possible functional block diagram of an agent-based system for generating outputs. The functional block diagram shown in Figure 6-16 does not prescribe an implementation. Rather, it describes the high-level Functional Blocks that are needed to implement the needs of output generation. Different implementations may need to add other Functional Blocks to meet their particular operational requirements.

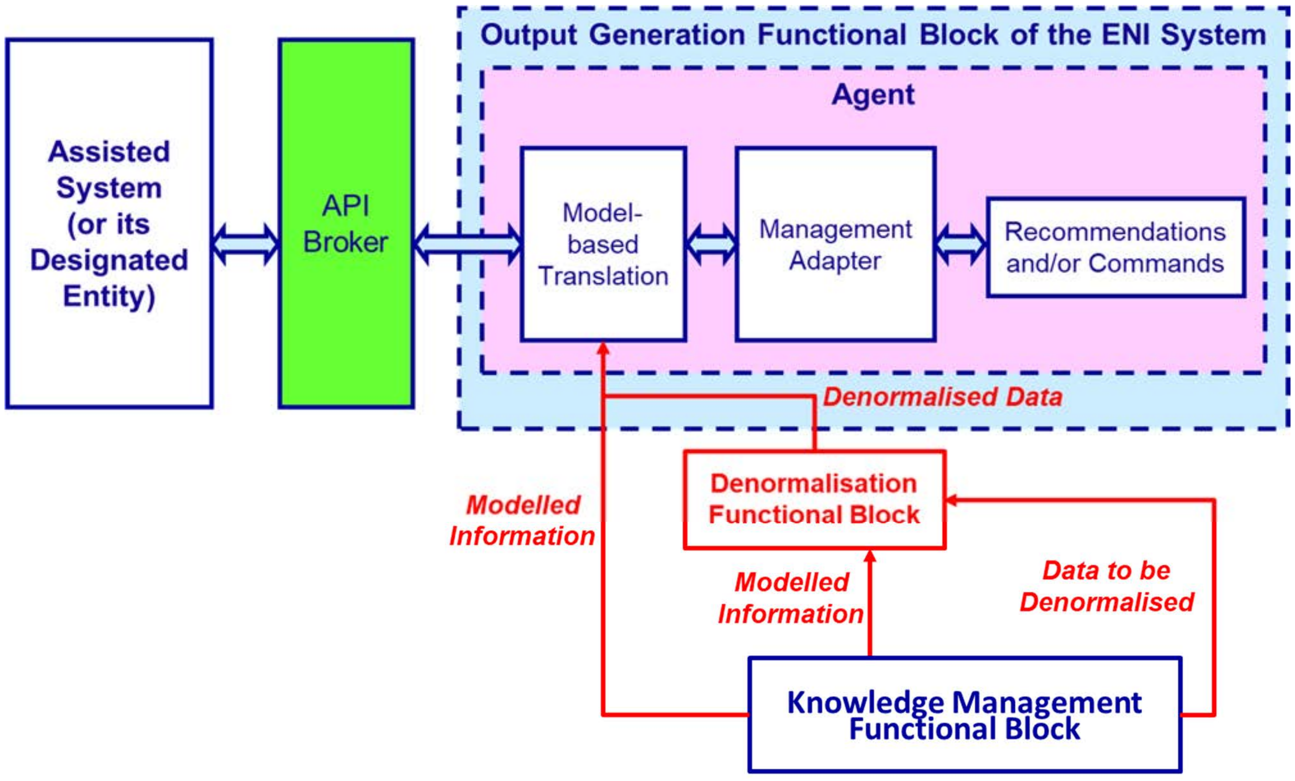

Output Generation Functional Block of the ENI System
Agent
Assisted System (or its Designated Entity)
API Broker
Model-based Translation
Management Adapter
Recommendations and/or Commands
Denormalised Data
Modelled Information
Denormalisation Functional Block
Modelled Information
Data to be Denormalised
Knowledge Management Functional Block

Figure 6-16: Output Generation Operation

The Output Generation Functional Block of the ENI System communicates with the Assisted System (and/or its Designated Entity). This communication may use an API Broker to translate between the APIs of the ENI System and the APIs of the Assisted System (and/or its Designated Entity). The Output Generation Functional Block is shown as an agent-based architecture. This architecture hides the native interfaces of both Assisted System (and/or its Designated Entity) as well as each managed entity of the Assisted System that is being managed by the ENI System.

In recommendation mode, communication from the Output Generation Functional Block shall go to the Assisted System (and/or its Designated Entity); it shall not be sent to managed entities of the Assisted System.

In management mode, communication from the Output Generation Functional Block shall go to the Assisted System (and/or its Designated Entity); it may also be sent to managed entities of the Assisted System, with the approval of the Assisted System (and/or its Designated Entity).

The model-based translation shown in the Agent is separate from the MDE Functional Block. The purpose of the agent's model-based translation is to use modelled information to ensure that the recommendations and commands are in a format that can be understood by the Assisted System (and/or its Designated Entity).

The advantages of this type of architecture include:

- Flexibility: the implementation may change (e.g. agent functionality may change) without affecting the API (built from more abstract levels) of either the ENI System or the Assisted System
- Extensibility: new agents may be added or removed to suit the needs of the application
- Manageability: instead of having to manage individual agents, the management focus is abstracted to managing roles

6.4 API Broker

The API Broker shall be implemented as a trusted entity. Only trusted entities can see, let alone interact with, the API Broker. However, it is an optional component, which implies that the API Broker shall be compliant with all ENI External Reference Points at the edge of the API Broker.

The function of the API Broker is currently limited to a gateway (i.e. a translator). Therefore, security does not require the presence of an API Broker.

NOTE: The following topics will be specified in Release 2 of the present document (see clause 9):

- RBAC [11], ABAC [12], or a similar mechanism shall be used to realize a scalable and consistent security mechanism with and without the API Broker.
- A mechanism is needed to ensure that if the API Broker is present, the same security functions are not done in both the API Broker and the ENI System.
- Some API calls can be resource-intensive. This enables attacks similar in nature to a DOS attack to be implemented by repeatedly calling the API. This requires further study.

7 Reference Points

7.1 Introduction

This clause uses the definitions of Reference Points from clause 4.4.6 to define the set of Reference Points used by ENI for communication between its different components as well as between it and the Assisted System.

ENI only standardizes communications that occur over a Reference Point.

NOTE: See clause 9 for applicable future work.

7.2 Reference Point Overview

Figure 7-1 shows an overview of the External Reference Points that provide internally facing External Reference Points (i.e. External Reference Points that communicate data and information from external entities to ENI. This figure shows ENI Reference Points for a single domain only.

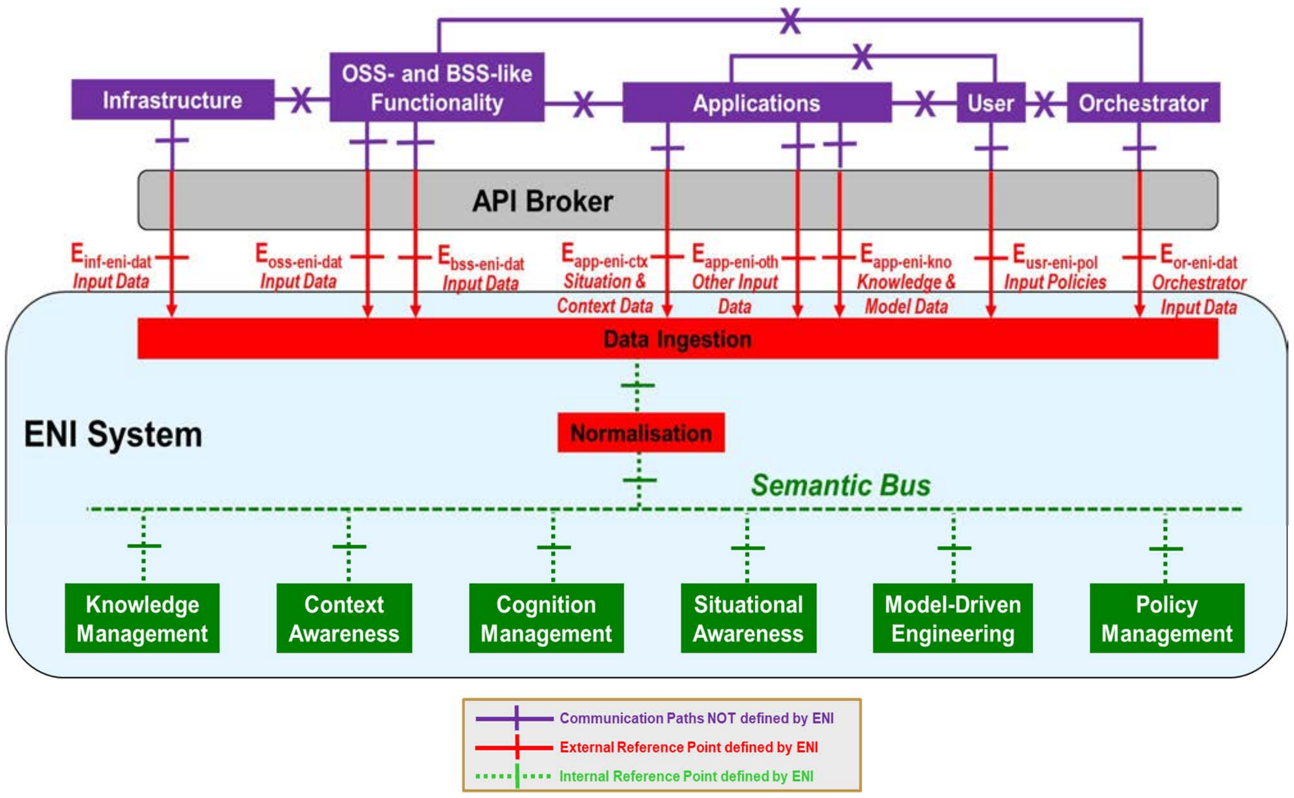

Infrastructure
OSS- and BSS-like Functionality
Applications
User
Orchestrator
API Broker
$E_{inf\text{-}eni\text{-}dat}$ Input Data
$E_{oss\text{-}eni\text{-}dat}$ Input Data
$E_{bss\text{-}eni\text{-}dat}$ Input Data
$E_{app\text{-}eni\text{-}ctx}$ Situation & Context Data
$E_{app\text{-}eni\text{-}oth}$ Other Input Data
$E_{app\text{-}eni\text{-}kno}$ Knowledge & Model Data
$E_{usr\text{-}eni\text{-}pol}$ Input Policies
$E_{or\text{-}eni\text{-}dat}$ Orchestrator Input Data
Data Ingestion
ENI System
Normalisation
Semantic Bus
Knowledge Management
Context Awareness
Cognition Management
Situational Awareness
Model-Driven Engineering
Policy Management
Communication Paths NOT defined by ENI
External Reference Point defined by ENI
Internal Reference Point defined by ENI

Figure 7-1: Overview of the ENI Internal Facing External Reference Points

Figure 7-2 shows an overview of the External Reference Points that provide externally facing External Reference Points (i.e. External Reference Points that communicate data and information from ENI to external entities.

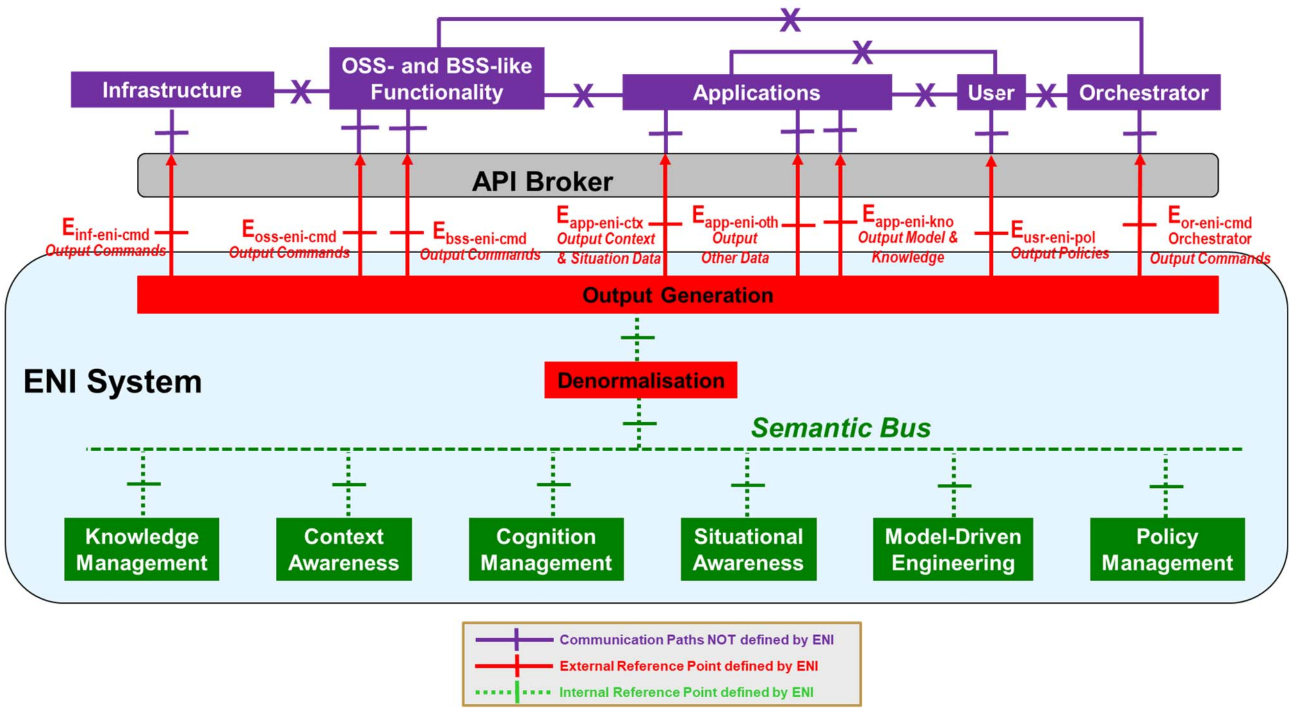

Infrastructure
OSS- and BSS-like Functionality
Applications
User
Orchestrator
API Broker
$E_{inf\text{-}eni\text{-}cmd}$ Output Commands
$E_{oss\text{-}eni\text{-}cmd}$ Output Commands
$E_{bss\text{-}eni\text{-}cmd}$ Output Commands
$E_{app\text{-}eni\text{-}ctx}$ Output Context & Situation Data
$E_{app\text{-}eni\text{-}oth}$ Output Other Data
$E_{app\text{-}eni\text{-}kno}$ Output Model & Knowledge
$E_{usr\text{-}eni\text{-}pol}$ Output Policies
$E_{or\text{-}eni\text{-}cmd}$ Orchestrator Output Commands
Output Generation
ENI System
Denormalisation
Semantic Bus
Knowledge Management
Context Awareness
Cognition Management
Situational Awareness
Model-Driven Engineering
Policy Management
Communication Paths NOT defined by ENI
External Reference Point defined by ENI
Internal Reference Point defined by ENI

Figure 7-2: Overview of the ENI External Facing External Reference Points

This figure shows ENI Reference Points for a single domain only.

Inter-domain Reference Points are the subject of future work (see clause 9). In addition, this version assumes that the OSS-like Functionality, Applications, BSS-like Functionality, and Orchestrator are all owned by the Operator. Applications that are not owned by the Operator are the subject of future work (see clause 9).

This version of the present document only defines External Reference Points. Work on communication between Functional Blocks using Internal Reference Points for future study (see clause 9).

NOTE: Currently, it is assumed that all applications are trusted. Work on non-trusted applications will start in a future Release.

Table 7-1 provides brief descriptions of the External Reference Points of ENI.

Table 7-1: ENI External Reference Point Overview

Name	Brief Definition	Interface Functions
$E_{oss\text{-}eni\text{-}dat}$	Defines data and information sent from the Assisted System (or its OSS-like functionality) to ENI; these data and information may be acknowledged by ENI.	Any data from the OSS-like functionality that ENI can use
$E_{oss\text{-}eni\text{-}cmd}$	Defines recommendations and/or commands sent from ENI to the Assisted System (or its OSS-like functionality); these recommendations and commands shall be acknowledged by the Assisted System.	The type of data sent by ENI depends on the mode that it is operating in
$E_{app\text{-}eni\text{-}ctx}$	Defines situation- and/or context-aware data and information exchanged between applications and ENI. Data and information received by ENI may be acknowledged by ENI. Similarly, data and information received by the Application may be acknowledged by the Application.	Contains input information for ENI to determine context and output context result data from ENI for Applications
$E_{app\text{-}eni\text{-}oth}$	Defines generic application data exchanged between Applications and ENI, which is neither situation- or context-aware data and also is not model or knowledge information. Data and information received by ENI may be acknowledged by ENI. Similarly, data and information received by the Application may be acknowledged by the Application.	Contains generic application data that may be useful to ENI, and generic ENI data that may be useful to applications
$E_{app\text{-}eni\text{-}kmo}$	Defines model and/or knowledge information and acknowledgements exchanged between Applications and ENI. Data and information received by ENI may be acknowledged by ENI. Similarly, data and information received by the Application may be acknowledged by the Application.	Limited to just DIKW and model information. (see note)
$E_{bss\text{-}eni\text{-}dat}$	Defines data and information sent by the BSS-like functionality to ENI; these data and information may be acknowledged by ENI.	Any data from the BSS-like functionality that ENI can use
$E_{bss\text{-}eni\text{-}cmd}$	Defines data and acknowledgements sent from ENI to the BSS-like functionality; recommendations and commands sent from ENI shall be acknowledged by the BSS-like functionality.	The type of data sent by ENI depends on the mode that it is operating in
$E_{usr\text{-}eni\text{-}pol}$	Defines policies exchanged between Applications and ENI that control behaviour (including services and resources) for a user (or an agent acting on behalf of the user). Policies and information received by ENI shall be acknowledged by ENI. Similarly, policies and information received by the Application shall be acknowledged by the Application.	These may be defined by the user or the operator. Inputs may be in a controlled language for Intent. Other input and output formats are for further discussion in Release 2 of the present document (see clause 9).
$E_{or\text{-}eni\text{-}dat}$	Defines data and acknowledgements sent from the Orchestrator to ENI; data sent from the Orchestrator may be acknowledged by ENI.	Any data from the Orchestrator that ENI can use
$E_{or\text{-}eni\text{-}cmd}$	Defines commands and acknowledgements sent from ENI to the Orchestrator; recommendations and commands sent from ENI shall be acknowledged by the Orchestrator.	The type of data sent by ENI depends on the mode that it is operating in

Name	Brief Definition	Interface Functions
$E_{inf-eni-dat}$	Defines data and acknowledgements sent from the infrastructure to ENI; data sent to ENI may be acknowledged by ENI.	Any data from the Orchestrator that ENI can use (e.g. log files, telemetry, alarms).
$E_{inf-eni-cmd}$	Defines recommendations and/or commands sent from ENI to the infrastructure; recommendations and commands sent from ENI shall be acknowledged by the Assisted System	The type of data sent by ENI depends on the mode that it is operating in.
NOTE: Requires extra security.		

7.3 Reference Point Definitions

7.3.1 Reference Point $E_{oss-eni-dat}$

This External Reference Point is used by the OSS-like Functionality of the Assisted System to provide data and information for ENI to ingest and process. This can include data, information, and metadata. ENI shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. This is a uni-directional External Reference Point, meaning that data shall only flow from the OSS-like Functionality to ENI.

ENI shall ingest the data and normalize it. If it is not possible to normalize the data, ENI shall report this to the OSS-like functionality of the Assisted System and ask for further information that defines the format, as well as expected characteristics and behaviour, of these data. Upon receipt of this information, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

7.3.2 Reference Point $E_{oss-eni-cmd}$

This External Reference Point is used by ENI to send recommendations and/or commands, as well as acknowledgements, to the OSS-like functionality. Recommendations and commands sent from ENI shall be acknowledged by the OSS-like functionality. ENI may include metadata to describe how the recommendations and/or commands are to be used by the OSS-like Functionality. This is a uni-directional External Reference Point, meaning that data shall only flow from ENI to the OSS-like Functionality.

If any recommendations or commands are not understood unambiguously by the OSS-like Functionality, then the OSS-like Functionality shall report this to ENI and ask for further information that defines the meaning and usage of the recommendation and/or command. A negotiation of data and information contained in the recommendation or command may need to occur to arrive at a final viable communication alternative. In this case, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

7.3.3 Reference Point $E_{app-eni-ctx}$

This External Reference Point is used to define situation- and/or context-aware data and information and acknowledgements exchanged between applications and ENI. ENI shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. Similarly, the Application shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. This is a bi-directional External Reference Point, meaning that context and situation data and information may flow from ENI to the Application or vice-versa.

Input context and situation aware data received by ENI shall be ingested and normalized. If it is not possible to normalize the data, ENI shall report this to the Application and ask for further information that defines the format, as well as expected characteristics and behaviour, of these data. Upon receipt of this information, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

ENI may include metadata to describe how the recommendations and/or commands are to be used by the Application. If any recommendations or commands are not understood unambiguously by the Application, then the Application shall report this to ENI and ask for further information that defines the meaning and usage of the recommendation and/or command. A negotiation of data and information contained in the recommendation or command may need to occur to arrive at a final viable communication alternative. In this case, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

7.3.4 Reference Point $E_{app-eni-oth}$

This External Reference Point is used to define generic application data and acknowledgements exchanged between applications and ENI that is neither situation- or context-aware data and also is not model or knowledge information. ENI shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. Similarly, the Application shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. This is a bi-directional External Reference Point, meaning that other types of data and information may flow from ENI to the Application or vice-versa.

Input application data received by ENI shall be ingested and normalized. If it is not possible to normalize the data, ENI shall report this to the Application and ask for further information that defines the format, as well as expected characteristics and behaviour, of these data. Upon receipt of this information, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

ENI may include metadata to describe how the recommendations and/or commands are to be used by the Application. If any recommendations or commands are not understood unambiguously by the Application, then the Application shall report this to ENI and ask for further information that defines the meaning and usage of the recommendation and/or command. A negotiation of data and information contained in the recommendation or command may need to occur to arrive at a final viable communication alternative. In this case, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

7.3.5 Reference Point $E_{app-eni-kno}$

This External Reference Point defines model and/or knowledge information and acknowledgements exchanged between applications and ENI. ENI shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. Similarly, the Application shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. This is a bi-directional External Reference Point, meaning that knowledge and model information may flow from ENI to the Application or vice-versa. This Reference Point shall contain extra security measures to protect the sensitive nature of these types of data and information. These data and information shall require a special protocol to exchange their content. That protocol will be defined in Release 2 of the present document (see clause 9).

Input application data received by ENI shall be ingested and normalized. If it is not possible to normalize the data, ENI shall report this to the Application and ask for further information that defines the format, as well as expected characteristics and behaviour, of these data. Upon receipt of this information, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

ENI may include metadata to describe how the recommendations and/or commands are to be used by the Application. If any recommendations or commands are not understood unambiguously by the Application, then the Application shall report this to ENI and ask for further information that defines the meaning and usage of the recommendation and/or command. A negotiation of data and information contained in the recommendation or command may need to occur to arrive at a final viable communication alternative. In this case, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

7.3.6 Reference Point $E_{bss-eni-dat}$

This External Reference Point is used by the BSS-like functionality to send data to ENI. ENI shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. This is a uni-directional External Reference Point, meaning that data shall only flow from the BSS-like Functionality to ENI.

Input application data received by ENI shall be ingested and normalized. If it is not possible to normalize the data, ENI shall report this to the BSS-like Functionality and ask for further information that defines the format, as well as expected characteristics and behaviour, of these data. Upon receipt of this information, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

7.3.7 Reference Point $E_{bss\text{-}eni\text{-}cmd}$

This External Reference Point is used by Defines data and acknowledgements exchanged between the BSS-like functionality and ENI. Recommendations and commands sent from ENI shall be acknowledged by the BSS-like Functionality. ENI may include metadata to describe how the recommendations and/or commands are to be used by the BSS-like Functionality. This is a uni-directional External Reference Point, meaning that data shall only flow from ENI to the BSS-like Functionality.

If any recommendations or commands are not understood unambiguously by the BSS-like Functionality, then the BSS-like Functionality shall report this to ENI and ask for further information that defines the meaning and usage of the recommendation and/or command. A negotiation of data and information contained in the recommendation or command may need to occur to arrive at a final viable communication alternative. In this case, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

7.3.8 Reference Point $E_{usr\text{-}eni\text{-}pol}$

This External Reference Point is used to define policies exchanged between external entities and ENI that control behaviour (including services and resources) for a user (or an agent acting on behalf of the user). ENI shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. Similarly, the external entity shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. This is a bi-directional External Reference Point, meaning that knowledge and model information may flow from ENI to the external entity or vice-versa.

Input policies received by ENI shall be ingested and parsed. If the parsing produces any errors or warnings, ENI shall report this to the entity that sent the policy. ENI may record the types of errors and/or warnings detected and later analyse them to determine if the grammar can be made more clear. ENI shall not continue work on the input policy that did not parse completely until it has been fixed by the external authoring entity.

ENI may include metadata to describe how policies are to be used by the external entity. If any policies generated by ENI are not understood unambiguously by the external entity, then the external entity shall report this to ENI and ask for further information that defines the meaning and usage of the policy. A negotiation of data and information contained in the policy may need to occur to arrive at a final viable alternative. In this case, ENI shall record this in its knowledge base and correct the format and content of future policies sent to this entity.

7.3.9 Reference Point $E_{or\text{-}eni\text{-}dat}$

This External Reference Point is used to define data and information sent from the Orchestrator to ENI. ENI shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. This is a uni-directional External Reference Point, meaning that data shall only flow from the Orchestrator to ENI.

Input application data received by ENI shall be ingested and normalized. If it is not possible to normalize the data, ENI shall report this to the Orchestrator and ask for further information that defines the format, as well as expected characteristics and behaviour, of these data. Upon receipt of this information, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

7.3.10 Reference Point $E_{or\text{-}eni\text{-}cmd}$

This External Reference Point is used to define recommendations and commands sent from ENI to the Orchestrator. Recommendations and commands sent from ENI shall be acknowledged by the Orchestrator. ENI may include metadata to describe how the recommendations and/or commands are to be used by the Orchestrator. This is a uni-directional External Reference Point, meaning that data shall only flow from ENI to the Orchestrator.

ENI may include metadata to describe how policies are to be used by the external entity. If any recommendations or commands are not understood unambiguously by the Orchestrator, then the Orchestrator shall report this to ENI and ask for further information that defines the meaning and usage of the recommendation and/or command. A negotiation of data and information contained in the recommendation or command may need to occur to arrive at a final viable communication alternative. In this case, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

7.3.11 Reference Point $E_{inf\text{-}eni\text{-}dat}$

This External Reference Point is used by Defines data and acknowledgements exchanged between the infrastructure and ENI. ENI shall acknowledge the receipt of data and information that it has requested, and may acknowledge the receipt of unsolicited data. This is a uni-directional External Reference Point, meaning that data shall only flow from the Infrastructure to ENI.

Input application data received by ENI shall be ingested and normalized. If it is not possible to normalize the data, ENI shall report this to the infrastructure and ask for further information that defines the format, as well as expected characteristics and behaviour, of these data. Upon receipt of this information, ENI shall record this in its knowledge base.

7.3.12 Reference Point $E_{inf\text{-}eni\text{-}cmd}$

This External Reference Point is used to define recommendations and/or commands sent by ENI to the infrastructure. Recommendations and commands sent from ENI shall be acknowledged by the infrastructure or an agent acting on behalf of the infrastructure. This is a uni-directional External Reference Point, meaning that data shall only flow from ENI to the Infrastructure.

ENI may include metadata to describe how policies are to be used by the external entity. If any recommendations or commands are not understood unambiguously by the infrastructure, then the infrastructure shall report this to ENI and ask for further information that defines the meaning and usage of the recommendation and/or command. A negotiation of data and information contained in the recommendation or command may need to occur to arrive at a final viable communication alternative. In this case, ENI shall record this in its knowledge base and correct the format and content of future recommendations and commands sent to this entity.

8 Interacting with Other Standardized Architectures

8.1 Introduction

The ENI System uses policy management, knowledge engineering, and cognition to enhance the operation and performance of the monitoring, configuration, management, and orchestration processes of the Assisted System. The ENI System will also offer the automation of these processes. The ENI System does not require any changes to Reference Points or their interfaces with existing Systems, including NFV MANO, SDN Controllers, or the MEF-LSO architecture. Rather, ENI uses those definitions and provides adaptation and mediation with external standardized architectures. However, changes made to existing Assisted Systems to comply with ENI External Reference Points and ENI APIs will benefit ENI as well as increase the semantics and understanding of interaction between the ENI System and these existing Assisted Systems.

ENI interacts with architectures from other SDOs using a subset of External Reference Points defined in clause 7.

An Assisted System may contain zero or more SDO Systems. All previous information defined in clauses 3 to 7 still apply to the Assisted System regardless of how many SDO Systems it contains.

Every clause that follows represents initial thinking of the ENI ISG. Each interaction with each external SDO is pending future collaboration and liaison work with that SDO.

Clause 8.2 describes a generic architecture that enables ENI to interact with one or more SDO Systems. Clause 8.4 provides preliminary thoughts about how the generic architecture may be applied to NFV MANO. Further work on interaction with NFV MANO, as well as with other SDO Systems, are for further study in Release 2 of the present document (see clause 9).

8.2 Generic Architecture

Figure 8-1 is a high-level diagram that illustrates how the ENI System may interact with an Assisted System that includes one or more SDO Systems (e.g. NFV MANO, MEF-LSO, BBF-CloudCO, as well as others that may become available in the market that are of interest to the ENI ISG) or their Designated Entities. In particular, it shows how ENI may interact at different levels (e.g. physical, resource, service, customer) of the combination of the SDO System(s) and the Assisted System or its Designated Entity. There may be zero or more instances of an SDO System in an Assisted System.

The SDO System layer(s) are arbitrarily shown between the Service and Resource layers. This need not be construed as a requirement that any particular SDO Component logically exists at that layer. In addition, an SDO System may affect one or more of the Physical, Resource, Service, or Customer Layers. An SDO System may introduce a new logical layer as well.

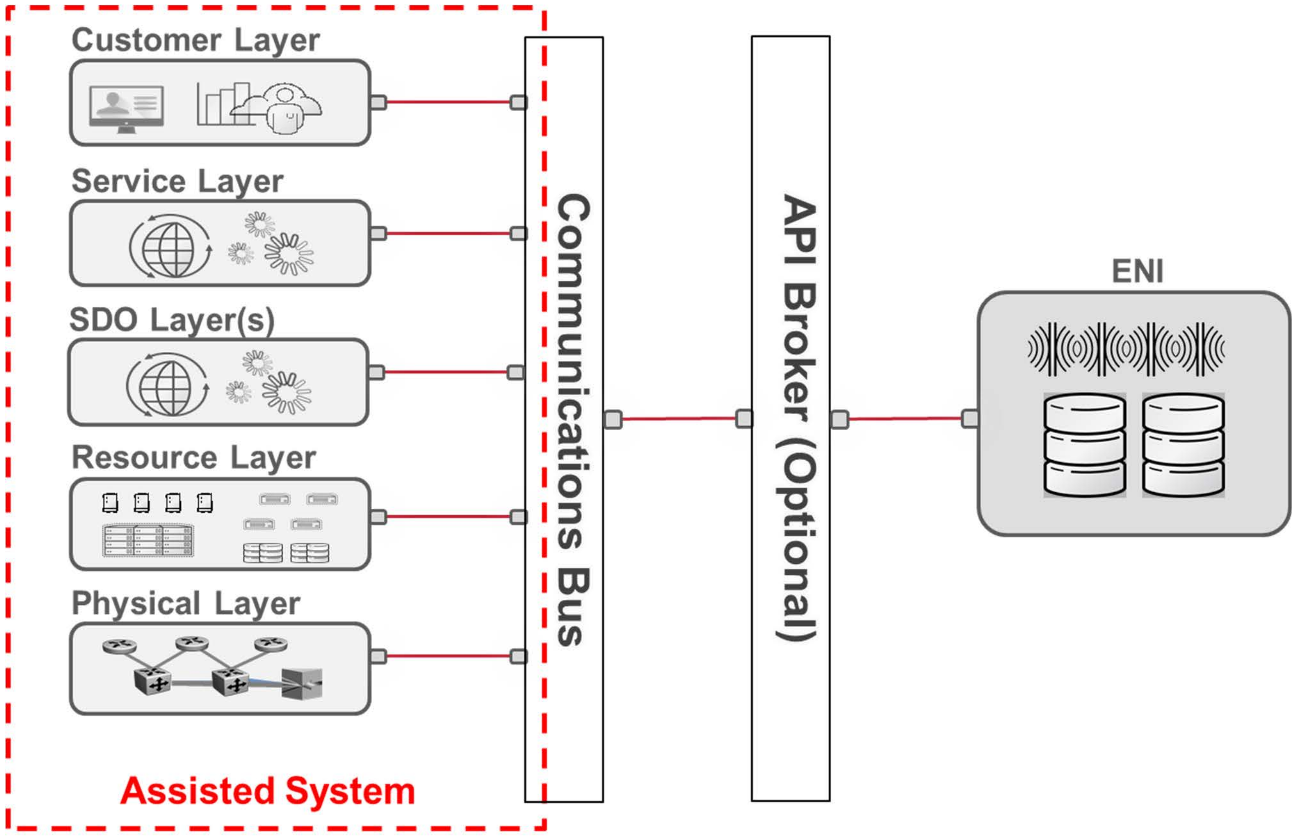


Figure 8-1: Overall view of System Interaction of ENI

Definitions of the above layers are summarized as follows:

- **Customer Layer:** Customer portal, and status notifications of their services and system health. This interface is needed as ENI is potentially changing this layer's behavior.
- **Service Layer:** Logical constructs that combine multiple elements to deliver a single service (example: A VPN service is a combination of components, including a firewall service, authentication service and routing services, blended into a single purpose construct).
- **SDO Layer:** Functionality provided by a System defined by an SDO. This appears as a set of Functional Blocks to ENI.
- **Resource Layer:** Data from all physical and virtualization telemetry (e.g. CPU, storage, and network).
- **Physical Layer:** All hardware resources supporting the infrastructure. Information on system health, usage and resource utilization will contribute to understanding the current state of hardware, and aid in predicting its future state. For example, this will aid in the optimization of the physical layer when using machine learning mechanisms.

This release of the ENI System architecture is limited to proposing how telemetry data from an Assisted System that contains one or more SDO Systems may interact. Release 2 and beyond will expand this interaction to other Layers shown in Figure 8-1, as well as new layers (if appropriate). Specific functional diagrams with Functional Blocks are discussed in clauses devoted to the interaction of ENI with a particular Assisted System (or its Designated Entity).

NOTE: This topic will be further specified in Release 2 of the present document (see clause 9).

8.3 Generic SDO Interaction Architecture

8.3.1 Introduction

An API Broker may be used to facilitate the interaction between Assisted Systems and the ENI System. This interaction may include one of two options:

1) direct communication between the ENI System and the Assisted System (or its Designated Entity); and/or

2) direct communication between the ENI System and the SDO System.

The difference is that in the first option, the Assisted System (or its Designated Entity) is responsible for communicating between ENI and the SDO System, whereas in the second option, the ENI System communicates directly with the SDO System. In the second option, the SDO System may inform the Assisted System (of which it is a part) of its interaction with the ENI System. In either of these options, the interaction shall not extend, or require the extension of, the functionality of the SDO system, or any of its components, beyond what the SDO System has currently defined in its specifications.

The interaction between the ENI and SDO Systems will use a set of common scenarios to investigate what types of interactions are possible. These scenarios are not unique to any one particular SDO, and some scenarios may not be applicable to all SDOs. They are provided to define a common framework to encourage conversation on interaction between ENI and each SDO System that wants to interoperate with ENI.

A common theme in each scenario is the functionality provided by the SDO System compared to the Assisted System that contains the SDO System. More specifically, the following five exemplary scenarios examine:

- Passive notification to the SDO System, no interaction with the rest of the Assisted System
- Data analysis between the ENI and SDO Systems, no interaction with the rest of the Assisted System
- Active assistance to the SDO System, no interaction with the rest of the Assisted System
- Active assistance to the entire Assisted System for only those functions that are contained in both the SDO System and the Assisted System
- Active assistance to the entire Assisted System, regardless of whether a function is only in the Assisted System or the SDO System

The above five scenarios are defined in order of increasing complexity. These are for further study in Release 2 of the present document (see clause 9).

8.4 Interaction with NFV MANO

8.4.1 High Level description of the NFV MANO - ENI Interaction

NOTE 1: The contents of this clause requires a liaison and agreement with the NFV MANO ISG, and is for further study in Release 2 of the present document (see clause 9).

The interaction of ENI with an NFV MANO-based architecture is shown in the following diagram. Note that the functionality provided by NFV MANO is only a part of what service providers work with in orchestration. For example, NFV MANO does not directly monitor or manage some of the components in the Physical Layer, as well as the most functionality of the Service and Customer Layers. All of these components need to be coordinated. This coordination may be done by the ENI System.

The High Level NFV MANO - ENI Interaction Architecture Framework is shown in Figure 8-2, where the ENI System is external to the NFV MANO framework:

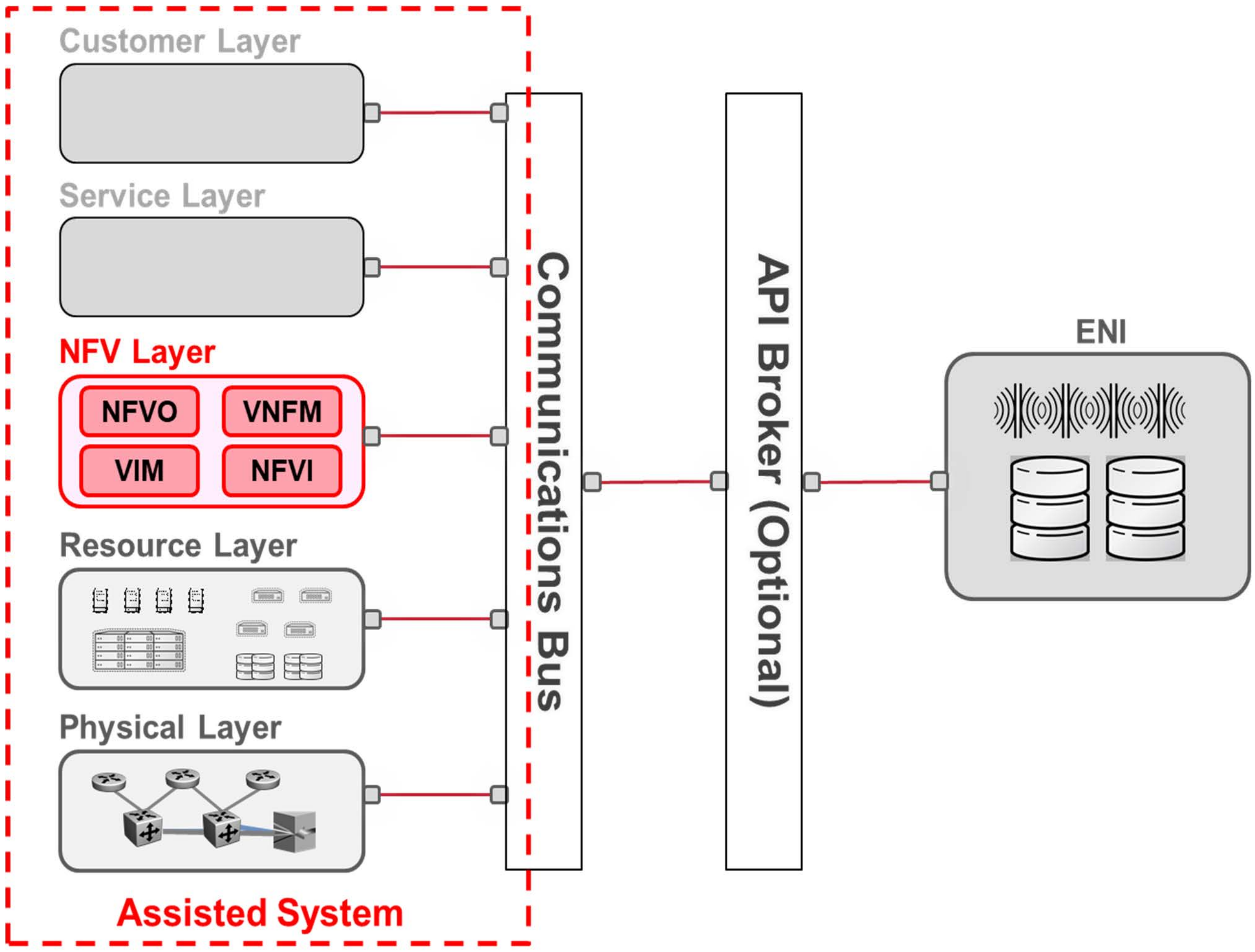


Figure 8-2: Interaction between the NFV MANO and ENI

The ENI System shall connect to the NFV MANO framework through specific External Reference Points in the following four functional areas:

- NFVO (NFV Orchestrator)
- VNFM (VNF Manager)
- VIM (Virtualized Infrastructure Manager)
- NFVI (NFV Infrastructure)

This will enable the collection of appropriate telemetry for ENI to analyse. However, it is unclear, at this point in time, how ENI can provide recommendations and/or commands to the NFV MANO Assisted System (or its Designated Entity), as the existing NFV MANO Reference Points are not suitable for conveying such information.

NOTE 2: Appropriate Reference Points are not currently defined by NFV MANO. Hence, this will be work for Release 2 of the present document (see clause 9), after a liaison has been created between ISG ENI and ISG NFV.

NOTE 3: Additional detailed interaction may be possible. For example, the ENI System may also be able to connect directly beyond these four areas into more specific subcomponents of the SDO System (e.g. the VNFM) or components that the NFV MANO System the Assisted System share (e.g. the EMS). This may enable ENI deterministic latency to be reduced.

The purpose of the NFV MANO System is to provide orchestration, management, and control functions. In the envisaged interaction, the ENI System may support and help to improve these NFV MANO functionalities. ENI may also enhance the effectiveness of NFV MANO by providing AI-based functionalities as well as context- and situation-awareness. There are several options for NFV MANO and ENI to interact. One example is using the API Broker to translate between ENI and NFV MANO APIs. However, equivalent External Reference Points will first need to be defined by the NFV MANO System for at least the NFV Orchestrator, VNF Manager, and VIM Functional Blocks. The desire to implement this interaction could also imply the addition and/or modification of External ENI Reference Points to facilitate this interaction. The advantage of this approach is that the API Broker decouples the ENI and NFV MANO Systems, so that either can change without directly affecting the other.

The NFV MANO System has specific management responsibilities that are different in scope than those of the ENI System. The interaction between the NFV MANO and the ENI Systems may take on a number of permutations; exemplary scenarios are described in the following clauses. The behaviour of both Systems to the messages that are exchanged in each scenario being proposed shall be defined in clauses 4.4.2.1 and 4.4.3 of the present document.

8.4.2 Initial proposals for interaction scenarios

8.4.2.1 Introduction

The following clauses describe four scenarios of interaction between the ENI and Assisted Systems, where the latter has an NFV MANO System embedded in it. The first three scenarios limit the ENI System to interacting with the NFV MANO part of the Assisted System only, while the last scenario removes these restrictions and allows the ENI System to interact with the entire NFV MANO System.

8.4.2.2 Scenario 1: Passive Notification to NFV MANO

This scenario assumes that interaction between ENI and the Assisted System shall be limited to ENI and NFV MANO.

The ENI System observes the interaction between the Assisted System and the NFV MANO System. The ENI System may then place information describing the results of analysis performed by ENI in a repository shared by the NFV MANO and ENI Systems for review and future decision support. In this option, the ENI System shall not send recommendations or commands to either the Assisted System or the NFV MANO System. The ENI System can only send notifications to the NFV MANO System when it adds or changes data or information in the shared repository.

8.4.2.3 Scenario 2: Active Data Analysis for NFV MANO

This scenario assumes that interaction between ENI and the Assisted System shall be limited to ENI and NFV MANO.

The ENI System may take on the role of augmented data analysis, and enhance the MANO subsystem via delivery of decision support to the NFV MANO orchestrator in real time. In this option, the ENI System only sends recommendations or commands to the NFV MANO System that affect data analysis and related functions, such as trend prediction. It may also send repository change notifications to the NFV MANO System.

8.4.2.4 Scenario 3: Active Assistance to the NFV MANO System

This scenario assumes that interaction between ENI and the Assisted System shall be limited to ENI and NFV MANO.

The ENI System may take on an expanded role, assisting the NFV MANO System where appropriate. In this option, the ENI System sends recommendations and commands to the NFV MANO System for all functions that both the ENI and NFV MANO Systems have. However, the ENI System will not send recommendations and commands to the NFV MANO System or the Assisted System that affect functionality that NFV MANO does not currently have in its published specifications. It may also send repository change notifications to the NFV MANO System.

8.4.2.5 Scenario 4: Active Assistance to the Assisted System

This scenario assumes that interaction between ENI and the Assisted System, including NFV MANO, shall be permitted.

The ENI System may take on a further expanded role, assisting the NFV MANO System where appropriate as well as providing functionality that the NFV MANO System does not offer to the rest of the Assisted System. In this option, the ENI System sends recommendations and commands to the NFV MANO System for all functions that both the ENI and NFV MANO Systems have as well as recommendations and commands to the Assisted System that are from functions that only the ENI System has. This scenario shall use explicit communication, via ENI External Reference Points, to ensure that the ENI System, the Assisted System, and the NFV MANO System shall agree on which control, management operations are done by the ENI System, and which are done by the NFV MANO System. Similarly, the ENI System, the Assisted System, and the NFV MANO System shall agree on which operations require which type of messaging protocol (e.g. request-response, or one-way notification, or something else).

This scenario is applicable to situations where there is an evolution in the network that requires new functionalities that NFV MANO does not currently address. As an example, this may happen for situations where future network resource models will require more granular and multi-faceted analysis (mobile devices as cloud resource nodes, etc.), and will require a complex management capability, which is ideally augmented via ENI.

The ENI System may inform the NFV MANO System of recommendations and commands that it has made to other portions of the Assisted System, as this may influence the decision process of the NFV MANO System. It may also send repository change notifications to the NFV MANO System.

NOTE: Work on how decisions based on the closed loop architecture of ENI and the open loop architecture of NFV MANO is for study in Release 2 of the present document (see clause 9).

9 Areas for Future Study

9.1 Introduction

The following clauses define areas for further study in Release 2 of the present document.

9.2 Future Study for the Assisted System

Assisted Systems often need to add different types of descriptive and/or prescriptive information to data and information communicated to ENI (e.g. for better understanding or to provide preferential processing). This may require additional study.

It is likely that Class 3 system communication evolves in future releases. Therefore, additional study may be required to understand the needs of these communication mechanisms, including how External Reference Points, Interfaces, and the functionality of ENI adapt to meet those needs.

Currently, it is assumed that ENI works as its own discrete System external to the Assisted System. The alternative (ENI working as an embedded System within the Assisted System) is currently not doable, because the appropriate APIs, models, and other required artefacts do not yet exist for ENI. This may be investigated in a future Release of the present document.

9.3 Models

9.3.1 Future Study for the Information Model

Release 2 of the present document shall explore various information models based on the conclusions of ETSI GR ENI 003 [i.16]. A single information model shall be used internally in ENI.

Release 2 of the present document may study how different information model fragments are exchanged between ENI and the Assisted System (or its Designated Entity). This has important security ramifications, since this information represents critical assets of both ENI and the Assisted System.

9.3.2 Future Study for Data Models

Release 2 of the present document shall explore how to create different data models using MDE mechanisms. This may include the following topics:

- Is there a single preferred data model that is used by all Functional Blocks, or is there a single preferred data model that is used by a particular Functional Block, or is there neither of these?
- If there are multiple data models used, how are their differences reconciled?
- The above questions apply to both input ingestion and normalization and denormalization and output generation.
- Some data models (e.g. of policies) may pose extra security risks, and hence, require additional security measures. While this existing document specifies a dedicated External Reference Points for exchanging model ($E_{app-eni-kno}$) and policy ($E_{usr-eni-pol}$) data, further refinement may be necessary.

9.4 Input and Output Processing

9.4.1 Future Study for Data Ingestion and Normalization

The topics of data and information ingestion and normalization shall require further study in Release 2 of the present document. This may include the following topics:

- The use of MDE mechanisms to simplify and aid the input ingestion and normalization processes.
- The use of metadata (with or without MDE mechanisms) to simplify and aid the input ingestion and normalization processes.
- The use of different non-AI algorithms, such as structure, pattern, and feature matching, as well as computational linguistics.
- The use of AI-based algorithms instead of or in addition to the above non-AI algorithms.
- The use of formal logic instead of or in addition to the above AI and/or non-AI algorithms.
- The use of ontologies to extend the use of formal logic.

9.4.2 Future Study for Denormalization and Output Generation

Release 2 of the present document shall investigate whether denormalization and output generation can be modularized as the inverse functions of normalization and data ingestion, respectively. This may simplify their operation, and enable them to take advantage of inherent efficiencies present. For example, metadata present and/or added to the normalization process may be used for the denormalization process.

9.5 Knowledge Processing

9.5.1 Future Study for Knowledge Management

The subjects of semantics (in general) and formal semantics (in particular) shall require further study in Release 2 of the present document. This may include the generation and use of semantic networks.

The subject of knowledge representation, including comparing different representation approaches, extensibility, and efficiencies, shall require further study in Release 2 of the present document.

The processes that turn data into information, information into knowledge, and knowledge into wisdom, shall require further study in Release 2 of the present document. This may include the description of operational flows of these processes.

The managing of knowledge shall require further study in Release 2 of the present document. This may include description and exemplary operational flows of:

- How changes to existing data and information are made.
- How conflicts between existing data and information versus newly ingested and/or discovered data and information are resolved.
- How knowledge is distributed to other ENI Functional Blocks.

9.5.2 Future Study for Context Management

The detailed modelling of context awareness shall require further study in Release 2 of the present document.

The interaction between the cognition model and context shall be further specified in Release 2 of the present document.

9.5.3 Future Study for Cognition Management

At least one formal cognition model for ENI shall be further specified in Release 2 of the present document.

More detailed description and specification of various cognition processes that may be used by ENI requires further study in Release 2 of the present document.

Symbolic, connectionist, and hybrid cognition architectures shall be further specified in Release 2 of the present document.

9.5.4 Future Study for Situation Awareness

The roles that different types of memory play in situational awareness shall be further specified in Release 2 of the present document.

The interaction between the cognition model and situations shall be further specified in Release 2 of the present document. This may include:

- How the cognition model enables the current situation to be determined.
- How the cognition model enables future situations to be predicted.
- How different AI algorithms can be used to leverage historical situation processing.
- How different AI algorithms can be used to leverage knowledge to achieve and protect system goals.

9.6 Future Study for MDE Processing

MDE Processing shall be further specified in Release 2 of the present document. Exemplary operational flows may be described to help explain:

- How the (single) ENI information model is maintained.
- How one or more ENI data models are constructed and maintained.
- How policies are created and managed.

9.7 Future Study for Policy Management and Processing

Policy Management and Processing shall be further specified in Release 2 of the present document. This may include exemplary descriptions and flows of the lifecycle of a policy.

The mechanism(s) for generating and managing each type of policy shall be further specified in Release 2 of the present document. This may include:

- Comparison of parser, interpreter, and compiler approaches for each type of policy that is constructed.
- Comparison of different techniques for policies to be input to ENI (e.g. using a Controlled Language or DSL).
- Study of different structures and languages for use by ENI internally.

9.8 Future Study for Reference Points

Existing External Reference Points may need to be enhanced, and new External Reference Points may need to be added, to accommodate increased functionality of ENI. This shall require further study in Release 2 of the present document.

Internal Reference Points shall be defined in Release 2 of the present document.

New Internal and External Reference Points may be added to Figure 4-1 in Release 2 of the present document.

Further study may be required, in Release 2 of the present document, of different roles being played by a Reference Point (e.g. producer vs. consumer). This further study includes considerations for separating data, control, and management from being communicated on the same interface. This is one of the reasons that the present document has differentiated between Reference Points that end with "-dat" and Reference Points that end with "-cmd".

Reference Point Use Cases shall be specified in Release 2 of the present document.

The following requirements shall be specified in Release 2 of the present document:

- Reference Point Configuration Requirements
- Reference Point Monitoring Requirements
- Reference Point Administration Requirements
- Reference Point Notification Requirements
- Reference Point Heartbeat Requirements
- Reference Point Security Requirements

The $E_{app\text{-}eni\text{-}kno}$ External Reference Point shall require a special protocol to exchange knowledge and model content. That protocol shall be defined in Release 2 of the present document.

Inter-domain Reference Points (i.e. internal ENI Reference Points that enable distributed ENI Functional Blocks and ENI (sub)systems to communicate) are the subject of future work.

9.9 Future Study for Interfaces

Hardware and Software Interfaces shall require further study in Release 2 of the present document. This may include how they are used to support API operations at one or more particular Reference Points.

Interaction between ENI Interfaces and the Interfaces of another SDO is for further study in Release 2.

9.10 Future Study for APIs and the API Broker

The API Broker shall be defined as a Functional Block, using the Reference Points and APIs defined by ENI, in Release 2 of the present document. This may include exemplary operational flows describing how the mediation is performed.

The subject of using an API Broker to mediate between ENI and the Assisted System (or its Designated Entity) when ENI External Reference Points are not available in the Assisted System (or its Designated Entity) shall require further study in Release 2 of the present document.

Some API calls can be resource-intensive when they operate, which enables attacks similar in nature to a DOS attack to be implemented by repeatedly calling the API. This requires further study in Release 2 of the present document.

9.11 Future Study for New ENI Functional Blocks

Additional Functional Blocks may be required for Release 2 of the present document. Examples include:

- Modeling different types of Repositories as one or more Functional Blocks.
- Modeling communication mechanisms, such as a publish-and-subscribe bus, as a Functional Block.
- Modeling Lifecycle Management as a Functional Block.
- Applications, OSS- and BSS-like Functionality, and the Orchestrator are all owned by the Operator. Applications that are not owned by the Operator are the subject of future work.

9.12 Future Study for Interaction with Other Organizations

Further work on interaction with other SDO Systems are for further study in Release 2 of the present document. All interactions that are documented in Release 2 of the present document shall require a liaison between ENI and the SDO System that is interacting with ENI.

This release of the ENI System architecture is limited to proposing how telemetry data from an Assisted System that contains one or more SDO Systems may interact. Release 2 and beyond will expand this interaction to other Layers shown in Figure 8-1, as well as new layers (if appropriate). Specific functional diagrams with Functional Blocks are discussed in clauses devoted to the interaction of ENI with a particular Assisted System (or its Designated Entity).

Currently, there are five interaction scenarios, ordered in increasing ENI involvement and interaction complexity, that are planned. Not all scenarios may be applicable for a given SDO System. These five scenarios are:

- Passive notification to the SDO System, no interaction with the rest of the Assisted System.
- Data analysis between the ENI and SDO Systems, no interaction with the rest of the Assisted System.
- Active assistance to the SDO System, no interaction with the rest of the Assisted System.
- Active assistance to the entire Assisted System for only those functions that are contained in both the SDO System and the Assisted System.
- Active assistance to the entire Assisted System, regardless of whether a function is only in the Assisted System or the SDO System.

9.13 Future Study for Security

ENI security is to be considered in the broader context of the Assisted Systems and Operator's environment. ENI security is subject to the security policies governing the aforementioned context and will match or exceed them.

As ENI exposes new interfaces, new APIs and Reference Points may expose new attack surfaces. ENI shall provide mitigation against attacks from those new attack surfaces. For example:

- Access to new interfaces may be subject to Role Based Access Control (RBAC) [11], Attribute Based Access Control (ABAC) [12], and/or finer granularity access control mechanisms.
- Access control for communications may be further protected with security protocols, such as TLS [13].
- Identity is a fundamental foundation for establishing security between parties. Credentials with identity information may be exchanged during any interaction with the ENI System.
- Policy may indicate the desired security level to be used. Authenticated and authorized policy changes shall be stored by ENI and may impact mechanisms used for ENI interaction.

- Confidentiality and integrity mechanisms, common in networking wire protocols, shall be used to mitigate man-in-the-middle attacks and to provide authentication as required by the proper interactions with ENI.

Auditing tools to determine which entity performed which operation at what time may be provided using programmatic interfaces.

ENI may provide additional security to applications as well as the Assisted System (or its Designated Entity) by more closely integrating with existing security systems through programmatic interfaces exposed over appropriate External Reference Points. This may provide multiple layers of defence (e.g. providing security on top on unsecure APIs, Systems and networks).

For all other potential security threats (e.g. already existing exposures for TCP/IP, virtualization, orchestration Systems etc.), reference to appropriate industry standards and mitigation techniques relevant to those environments may be provided.

Annex A (informative): SDO and Open Source Interactions

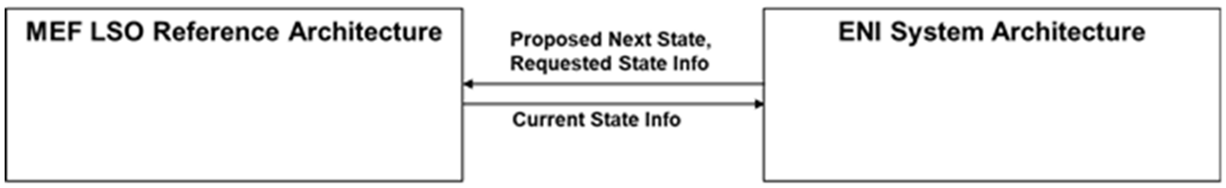
A.1 Integration with Other SDOs and Open Source Communities

A.1.1 Introduction

This informative annex will describe preliminary thoughts on integrating ENI with MEF-LSO and the BBF CloudCO architectures. This work, along with work on other architectures (e.g. ZSM), is for further study in Release 2 of the present document.

A.1.2 Interaction with MEF-LSO

The interaction between the High Level ENI Architecture and the MEF-LSO Reference Architecture is indicated in Figure A-1.


MEF LSO Reference Architecture
Proposed Next State, Requested State Info
Current State Info
ENI System Architecture

- ENI ISG collaborates with MEF on liaison basis
- ENI insights can improve the consumer experience by improving the inter-operator interaction
- Of potential interest can also be LSO extensions to include ENI ⇔ ENI interaction and collaboration

Figure A-1: Interaction between the MEF LSO and ENI

NOTE: Revise figure to be consistent with Figure 8-1 in a future version of this Release.

The MEF's LSO Reference Architecture defines a set of Management Interface Reference Points that specify the logical points of interaction between different functional management entities. Each Management IRP is defined by an Interface Profile, supported by a common information model and a set of interface-specific data models, and implemented by a set of interface-specific APIs.

The ENI Engine sits outside the MEF LSO RA functional elements, and augments their behaviour. The ENI Engine interacts with the Functional Blocks of the LSO RA via the existing APIs defined by the LSO RA; this avoids altering the MEF LSO RA architecture.

A.1.3 Interaction with BBF CloudCO

The BBF CloudCO (Cloud Central Office) reference architecture is a Central Office Domain that is (1) leveraging SDN and NFV techniques, (2) running on a cloud-like infrastructure deployed at Central Offices and (3) accessed through a Northbound API, allowing Operators, or third parties, to consume its functionality, while hiding how the functionality is achieved from the API consumer.

Figure A-2 shows the interaction between the High-Level ENI Architecture and the BBF CloudCO architecture, see BBF TR-384 [2]. All the functional components that are shown in Figure A-2 are defined in BBF TR-384 [2] and ETSI GS NFV-MAN 001 [3]. The reference point between the ENI System and the BBF CloudCO architecture is being specified by ETSI ISG ENI. All the other reference points/interfaces are specified in BBF TR-384 [2] and ETSI GS NFV-MAN 001 [3].

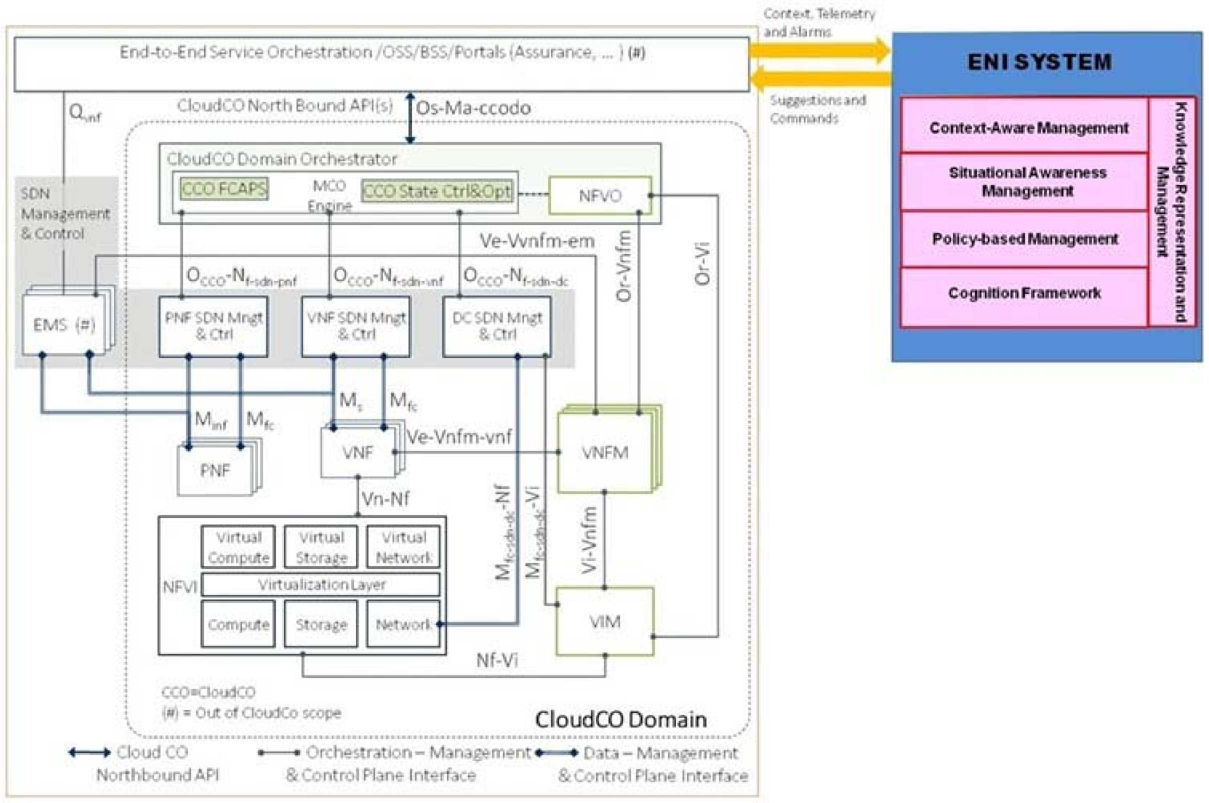


Figure A-2: Interaction between the BBF CloudCO and ENI

NOTE: Revise figure to be consistent with Figure 8-1 in a future version of this Release.

In Figure A-1, the shown End to End Service Orchestrator coordinates all client interfaces with regard to their privileges and views, and resolves any contentions that may arise. Furthermore, the E2E Service Orchestrator coordinates multiple CloudCO Domain Orchestrators, deployed on specific nodes. The CloudCO Domain Orchestrator manages, controls and orchestrates each CloudCO Domain that spans over portions of the whole access/edge network. Part of the E2E Service Orchestrator function is to as well coordinate user plane handoffs when services cross domain boundaries.

In this scenario the ENI System interacts with the CloudCO End-to-End Service Orchestration/OSS/BSS/Portals functional component via a reference point that is being specified by ISG ENI. The ENI System collects the necessary information from the CloudCo Domain Orchestrator on topics such as:

1) supported tenants;

2) supported use cases;

3) used and available resources per tenant and use case; and as well

4) information related to lifecycle management of the virtual environment.

The output of the ENI System can be used for several purposes such as acting on the service/policy/resource module and engineering rules, recipes for various actions, policies and processes.

A.2 Interaction with Open Source Communities

The interaction between the ENI System and various open source communities is for further study in Release 2 of the present document.

Annex B (informative): ENI Architectural Evolution

B.1 ENI Architecture Evolution Motivation

Technology is evolving especially when it comes to Cognition, near natural language processing, deep reasoning and even situation awareness and context awareness.

Industry agreement as to Policy semantics (not DSL) and data normalization and tools is still WIP, at best.

Multiple PoC are emerging at ETSI ENI and elsewhere in the industry. These PoC are using off the shelf technology with limited use of afore mentioned advanced technologies.

Need to coordinate architecture with related SDOs and Industry Consortia.

Need to coordinate Reference Points with other SDOs before finalizing the model. Additional challenges.

Data ingestion: use different tools today, need to be normalized into one common and COORDINATED data with agreed representation!

Infrastructure (a.k.a "Existing System") is still silo'ed hard to accomplish coordinated configuration, control and management of Network with Compute + Storage.

B.2 ENI Architecture Evolution Proposal

- Define few evolutional phases.
- Align PoC, Open Source and other SDO and industry consortia to ENI phases.
- Interact with open source communities!
- Update ENI Architecture based on industry progression.
- The above is designed to make ENI the focal of ML/AI evolution for the Telco industry and for its use of various Orchestration and Cloud Operation Environments (and network industry?) by ensuring tight coupling.
- Time Limit: limit evolution and publication of successive ENI phases, to no more than 18 months after the first publication of the ENI Architecture.

B.3 Proposed Definition of the ENI Phases

An "ENI Phase" is defined as compliance with a set of ENI Reference Points and its specific version, as set by the ENI WG from time to time.

For clarity, it is expected that a Reference Point may be versioned as the ENI architecture evolves.

The architecture document will continue as currently planned and will not slow down or change course.

For each "ENI Phase", ENI will consider inputs from its documents and members, ongoing PoC, interaction and liaison with SDO and collaboration with open source communities to determine the set of Reference Points required to be supported.

An ENI implementation, MAY claim compliance for a given ENI Phase when it is in compliance with a set of Reference Points set by the ENI WG, as aforementioned.

It is recommended that the Architecture WG will provide specific recommendation of the phases using Reference Points.

Annex C (informative): Authors & contributors

The following people have contributed to the present document:

Rapporteur:
Dr. John Strassner, Huawei Technologies

Other contributors:
Antonio Gamelas, PT Portugal SGPS SA

Dr. Uri Elzur, Intel Corporation (UK) Ltd

Haining Wang, China Telecommunications

Yue Wang, Samsung

Dr. Raymond Forbes, Huawei Technologies

Francisco da Silva, Huawei Technologies

Luca Pesando, Telecom Italia

Dr. Liu Shucheng, Huawei Technologies

Yannan Bai, China Telecommunications

Fred Feisullin, Verizon UK Ltd

Li Weiyuan, China Mobile

Huang Bingming, China Unicom

History

Document history		
V1.1.1	September 2019	Publication