NMOP
Internet-Draft
Intended status: Informational
Expires: 2 September 2026

N. R. Davis
Ciena
C. Cardona
NTT
D. Lopez
Telefonica
M. Palmero
Independent
1 March 2026

Generalized Capability Principles
draft-davis-nmop-generalized-capability-principles-01

Abstract

This document introduces a framework for capability modeling based on the specification and refinement principles established in ITU-T G.7711 Annex G (also previously published as ONF TR-512.7. See latest G.7711 release) and the modeling boundaries work documented in draft-davis-netmod-modelling-boundaries. The framework defines how component宴都ystem capabilities can be explicitly described and refined via a process of pruning, refactoring, and occurrence formation.

These capability definitions can target detailed operational considerations, system interactions, licensing, abstract product declarations, or sales and marketing. The framework supports modular, layered, and fractal declarations of networked behavior, and provides a foundation for a suite of future IETF drafts aligned with ongoing work on photonic plug manifests, entitlement/licensing, IVY equipment modeling, energy/thermal considerations and related domains.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://github.com/marisolpalmero/draft-ietf-davis-generalized-capability-principles/blob/main/draft-davis-nmop-generalized-capability-principles-latest.md. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-davis-nmop-generalized-capability-principles/.

Discussion of this document takes place on the Network Management Operations mailing list (mailto:nmop@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/nmop/. Subscribe at https://www.ietf.org/mailman/listinfo/nmop/.

Source for this draft and an issue tracker can be found at https://github.com/marisolpalmero/draft-ietf-davis-generalized-capability-principles.

Status of This Memo

Copyright Notice

Table of Contents

1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT" "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in the document are to be interpreted as described in RFC2119}}.

The following terms abbreviations are used in this document:

* capability: What can be achieved by an individual item both alone and in assembly (using the component-system pattern)

* needs: Related to capability, this is what the item, either alone or in assembly, needs to achieve its capabilities

* manifest: A list of essential contents

* specification: A detailed definition of capabilities/needs in terms of opportunities/constraints including the arrangement of essential parts and their interconnectivity in assembly

* representation: An expression of structure and properties from a perspective

* component: A thing defined by a boundary where the internal structure within that boundary is not directly visible but is apparently visible through the behaviour exposed at that boundary

* port: A place on the boundary of a component where interaction with that component is possible

* component-system: A pattern that expresses each item as a component where components can be assembled into systems and where a system can be represented as a component where that assembly may be of real things or may be abstractions of the effect of real things.

* occurrence: A thing, the specification of which is a purposeful refinement partially constraining the definition of a broader thing, where a thing is a component, a specification etc.

* pruning: A process of narrowing of definition by reduction of capabilities

* refactoring: A process of rearranging, splitting and combining representation whilst maintaining semantic validity

* pruning & refactoring: The process that supports intentional progression of refinement from one level of structure of occurrences (e.g., system of components) to the next more specific level of structure of occurrences

2. Introduction

Currently, capabilities are mainly described loosely in human readable text, where that text is often incomplete, ambiguous or inconsistent. While people make these systems work in practice, the looseness result in errors, inefficiencies and limited reuse. As automation increases, there is a growing need to enable machine reasoning about the capabilities of network systems and components. While Large Language Models (LLMs) can interpret traditional documentation, there remains a strong need for greater formal rigor and structured representation to improve efficiency and precision. When asked, LLMs indicate that a rigorous model is preferable to loose ambiguous text. Existing IETF models predominantly focus on configuration, operational state, and telemetry. What is missing is a cohesive framework for expressing what a system _can_ do, i.e., its capabilities, in a declarative, structured, and reusable form. This document introduces the principles for a capability modeling framework grounded in the specification concept established in [ITU-T_G.7711] ([ONF_TR-512]). It applies these principles through the lens of the *component寰都ystem pattern* from [ONF_TR-512.A.2],

using the concept of *emergence through recursive narrowing, refactoring and occurrence formation*. These ideas are extended further by the modeling boundary principles described in [mobo]. The result is a standardized and extensible approach for expressing features, operational constraints, internal dependencies, etc. - separately from instance realizations. This approach supports capability modeling for any aspect of the controlled networking solution, and is designed to enable capability assembly, dynamic composition, licensing control, and integration with other IETF frameworks such as IVY equipment, photonic plug manifests, and entitlement interfaces. It also supports initiatives focussing on energy/thermal considerations where specific detailed capabilities and their power/thermal implications become critical considerations.

3. Problem Statement

Network technologies and management-control frameworks increasingly rely on declarative data models to represent both configuration and operational state. However, these models often lack a principled way to describe the _capabilities_ of components and systems窶背hat they are able to support or provide, independent of any particular operational instance. This omission makes it difficult to reason about compatibility, constraint satisfaction, composition, or even basic intent feasibility. Clearly, many of these activities take place prior to the installation of the equipment and indeed determine which equipments are to be planned to be installed. In these cases it is not possible to interogate the actual equipment. Whilst knowing the YANG model for the equipment is beneficial, it is not sufficient as the YANG model essentially provides a space within which actual state etc. can be expresses, but it supports all possible combinations. The equipment will be very limited in comparison. Often it is desirable from a systems operation perspective to reduce the available capability through policy or other mechanisms due to the restrictions of a specific role. This becomes challenging if the base capability of a component is unclear and expressed in a chaotic form. In practice, five distinct concerns are often conflated, and also not fully expressed, within data models: - The *generic definition* of a model element or concept (e.g., a termination point) - this is expressed in YANG. It is a very broad definition encompassing all possible opportunities and ofthen many illegal state combinations etc. - The *capability definition* of a system or component, i.e., what it can support or expose (e.g., by a specific type or role of termination point). This is not expressed fully in YANG. There are both challenges with the expression of base capability and expression of the capability of combinations. This is especially sparce in representation - The users *policy definition* for system operation - the user may eliminate particular capabilities due to complexity, lack of trust,

regulation etc. and will not want them offered or may not want them offered under certain circumstances. The equipment will be expected to behave as if it does not have the capabilities as approproiate. - The *system combination* where an entity type may play several different roles and in each role may have specific distinct intentional limitations/restrictions. - The *operational instance*寰背hat is configured or active at a given time. Without a clear structural separation and with the sparseness of information on specific capabilities, it becomes challenging to formally describe feature constraints, support boundaries, or internal limitations. Implementers resort to informal documentation, code comments, yellow stickies, or out-of-band agreements to capture the intent behind model behavior. This reduces interoperability, increases integration effort, and undermines automation as a result of - *Ambiguity* between what a model element _is_ versus what a system _can support_. - *Redundancy* and inconsistency in the representation of common constraints (e.g., port types, layering, resource limits). - *Tooling difficulty* when extracting interoperable subsets of large models or generating technology-specific profiles. - *Incompatibility* between modular subsystems or plug-ins that must declare and verify their supported features. Furthermore, current models tend to assume a fixed taxonomy of types and features, rather than supporting a process of recursive refinement. This limits their ability to express how complex capabilities _emerge_ through constraint, composition, and modular pruning of more general-purpose constructs. What is needed is a modeling framework that: - Allows systems and components to be described in terms of their *capability boundaries*, including *capability interactions* separate from operational state, - Supports *refinement via pruning and refactoring to yield flexible structural transformation* rather than rigid inheritance or classification, - Enables *recursive occurrence formation*, where each level of pruning and refactoring produces a usable semantic structure, - Accommodates *multiple valid refinement paths*, supporting different levels of granularity and domain specificity, - Provides a *coherent trace* from abstract capability declarations down to deployable or licensable configurations. This draft introduces such a framework by building on the refinement logic of [ITU-T_G.7711] ([ONF_TR-512]) in general and especially the *specification pattern* structures of ITU-T G.7711 Annex G (ONF TR-512.7) which provides a means of expressing bounded capability envelopes through a formal refinement of generic model elements. This also provides grounding in the recursive occurrence model informed by the component寰都ystem pattern [ITU-T_G.7711] ([ONF_TR-512.A.2] and modeling boundaries approach [mobo]. This document leverages the foundations laid by [ITU-T_G.7711] ([ONF_TR-512]).

The same expression challenges appear in statements of intent. The process of formulating intent through negotiation and resultant gradual refinement has a similar feel to the degrees of pruning and refactoring of the specification.

4. Specification in terms of the Model

The specification of capability should be presented in terms of the terminology of the problem space and hence in terms of the appropriate model. The challenge is determining which model is the "appropriate" model.

An area of the problem space can be described in different ways depending upon what the intention of the model is. There are many ways of representing a semantic space/

Prior to embarking on evaluation of specification of capability, it is important to consider the specific model and how it is structured.

* Focus: Semantic area covered at centre and periphery
* Specialization: Specific detailed focus on an area with rich structure, e.g., PCE, problem analysis, etc.
* Granularity: the 窶ize窶of the semantic units (including the depth of recursion of fractal representations)
* Phase: The positioning of the semantic boundaries
* Richness: The detailed coverage within a semantic unit
* Fidelity: Precision v approximation
* Abstraction: Closeness to actual detail
* Maturity: Lifecycle development stage. How stable the model is likely to be. This is primarily about semantics, but also covers syntax.
* Omission: Gaps and missing parts

5. Generalized Modeling via ComponentSystemSpecification Refinement

This framework moves away from rigid classification schemes and instead adopts a dynamic, refinement-based approach to modeling. Traditional classification attempts to impose fixed categories onto a system, but this often obscures nuance, variation, and the emergence of intermediate structures that carry operational or architectural significance.

We begin instead with the concept of a *universal component*—a general-purpose structure with maximal capability potential. Through the process of *pruning & refactoring* (constraint-driven refinement), this semantic volume is gradually refined, yielding intermediate structures with more sharply defined roles and properties. These refined artifacts are not pre-classified entities, but *emergent forms* that arise naturally at specific “sweat spots” in the refinement trajectory, where the remaining capabilities align with a recognizably useful or interoperable function.

Each such emergent form is treated as an *occurrence*. Occurrences appear at every stage of meaningful refinement including at the level of final implementation instances. At all stages of use the application of properties is via the idea of intent where even the tightest constraint of a single value is essentially a statement of intent (as it is impossible to guarantee that a property will be set). This intent consideration will be dealt with further later in this document.

An LTP (Logical Termination Point) in [ITU-T_G.7711] ([ONF_TR-512]), for example, is not a primitive class but a pattern that arises from pruning and constraining the universal component until only the semantic envelope of an LTP remains. A TerminationPoint from RFC8345

To support variation, reusability, and convergence across implementations, each component or system is described not by a single fixed class, but by a *specification*: a constrained and possibly pruned refinement of a more general and broader model element. This allows the model to express bounded capabilities without requiring full instantiation, enabling tools and orchestrators to reason about compatibility, substitution, and support constraints before deployment. The specification describes the capabilities of an occurrence in terms of occurrences achieved via similar pruning. A system spec is a pattern assembly of subtly specialized occurrences at a particular level of specialization arranged in a meaningful structure that yields a relevant behaviour. The specification of an occurrence is itself a system spec.

The combination of the *componentsystem pattern* with the *specification refinement pattern* enables a modeling architecture where:

* Systems are recursively composed of components,

* Specifications constrain and refine capabilities at each level,

* Occurrences are layered realizations of specs applied to specific contexts or configurations.

This approach supports *gradual realization*, where capability declarations can progressively transition from abstract to concrete, through intermediate spec refinements and pruning. Each layer of model realization adds specificity— structurally (via system composition), behaviorally (via constraints), and operationally (via mapping to configuration/state models).

A specification may provide explicit definifinition of a property as discussed above but it may also refer to one or more other specification(s). For example a specification may include a set of properties specified elsewhere. It may also define a property that is an enumeration of literals or identifies where those literal values or identify values are actually references to other specifications that provide deeper detail.

In an ideal environment, there is an ecosystem of specificactions each providing interrelated detail to fully define the semantics. The ecosystem would include specifications from standards bodies providing the definition of a network protocol that can be interpreted by an AI component such that the abstracted effect on the solution can be fully understood and simulated/emulated. Any detected conditions would be understood in terms of the protocol and hence the implications of the condition detected in terms of the carried signal can be fully understood.

In this ideal environment, the specification would fully capture all non-failure case behaviours of a component (and potentially some common failure cases) and the component would be designed internally to "guarantee" these behaviours (it would be engineered with appropriate control structures that would bound its behaviour). These specifications, although abstractions would often be highly complex (consider the specification of a CPU for example), but would be less overwhelming in detail and stated in terms of intentional behaviour as opposed to behaviour of the parts. The specification is a statement of the effects of the assembly of detailed parts (see definition of component).

The specification of capability provides a stabilising layer reducing the reasoning required to build a solution as a result of not having to assess the full detail of behaviour of all assemblies to the finest detail. The specification of capability will have a unique identifier that anchors the definition and allows it to be accessed. This reflects the same principle that gives rise to labels in a taxonomy, where the label recalls the abstract definition removing the need to understand the effect of the parts from first principles.

Today’s solution at best have a coded form of the semantic interpretation that may not reflect the formal definition due to inaccuracies of interpretation. Many semantics are reduced to inconsistent labels that a user has to interpret. Whilst an LLM can do a reasonable job at interpretation of chaotic data, it will benefit a rigorous model traceable through formal definitions to fundamentals.

6. Specifications and LLMs

As discussed briefly above, LLMs can take advantage of specifications of capability. The LLM reasoning load can be reduced by working with the guaranteed behavioural abstraction provided in the specification for a component as opposed to working at the finest of details (it does not always need to understand the environment using string theory!).

The LLM can develop system solutions by assembling components of understood capability (using normal engineering and design processes) knowing that the behaviour of the components are internally controlled to be within the bounds of the specification. The LLM can then describe the behaviour of the system at its boundaries, i.e., of the component(s) that that system can realize. Hence the LLM can develop the specification for the components it produces.

For components not produced by a specific LLM (produced by another LLM or by a human), the LLM can assess the internal workings of the component (by reviewing the actual code/circuitry) at fine-grained detail. LLM reasoning can:

* extract the essential behaviour and abstract that to form a specification

* consider whether that abstracted behaviour defined in the specification appears beneficial in the formation of relevant systems and where not, propose simplifications

* evaluate the robustness of that essential behaviour and propose enhancements to ensure that the component operates within the desired bounds

* review existing specifications to determine whether other components already do a similar job

* etc.

7. Some specification examples

This section provides some simple examples and will reference the equipment capability draft and other future drafts.

7.1. A temperature sensor

Consider a simple temperature sensor. The physical sensor will have an operational range, a precision, an accuracy, etc. It will provide output in particular units and may be able to indicate out of range. The sensor is itself a small system of components. It will be sensitive to power supply behaviour, humidity and other environmental factors.

All of the above will be included in the hardware specification of that physical component. That component when designed into a system will contribute to the system behavior.

For this example we will assume that the output for that sensor is available via a control solution and is presented at an externally accessible interface. We will assume that the presentation is in JSON and that presentation was defined in YANG.

In a the imagined application for this sensor, lets assume that the temperature is relevant only to whole degrees and is required to be in Celsius so an integer is used to represent the temperature.

With this level of coarseness the fine grained precision and accuracy of the actual component can probably be ignored (although the component may be pushed close to its limits and hence there may be an accuracy consideration etc.), but the operational range is potentially still relevant and environment effects that cannot be eliminated still need to be understood.

There may also be known failure modes that cause detectable incorrect readings that need to be accounted for.

So, considering the component alone, simply stating integer in the YANG model is not sufficient.

Going further, the temperature sensor has a particular role in the context of the equipment it is monitoring. There may be several temperature sensors on that single equipment. Traditionally they would have had distinct labels (although these were often potentially misleading). Whilst this may have been sufficient in a basic operations environment, much more can be done and is probably necessary current and future solutions.

Having an identifier is clearly necessary, but that should lead to an accurate and fully interpretable representation of the positioning of that component in the equipment in isolation and in the broader solution as a whole.

For example, the detector may be at the top of a circuit pack that is placed in an assembly with convection cooling where that detector is provided to measure the temperature of the airflow leaving the top of the circuit pack and hence feeding to the next equipment above.

For a full understanding of the implications of a measurement provided by that detector, a detailed understanding of its positioning and purpose is necessary. It is intended that the specification model provide such detail.

The specification model will be generalized such that the details provided can be used in any relevant application. It will not describe detailed per instance cases. Hence the specification will be used in conjunction with the actual instance arrangement to allow understanding of any reading in context.

Traditionally, with ad-hoc formatting and variable accuracy of definitions etc., only a well experienced SME would have a chance of determining the relevance of a detected value.

In a modern and future solutions we can do and have to do better. The intention is that the specification approach using the generalised specification definition structure set out in this document will provide a basis for LLM assisted specification generation and interpretation.

8. Recursive pruning and refactoring

This builds on the example sketches and formalizes the process of recursive pruning and refactoring.

The essential process involves defining a general abstracted thing at some intermediate point in the progression of refinement (e.g., a temperature sensor), setting out a reasonable derivation path from the most generalized component and then refining that general abstract thing by recursive pruning and refactoring to arrive at the necessary specialization.

The following subsections take some generalised cases to illustrate the process.

8.1. Thing to component

In this approach a thing has all possible functions and capabilities of anything imaginable. Moving to component via pruning and refactoring involves recognition of the concept of boundary of a thing and then facet of a boundary, i.e., a surface that can "interface" with the surface of another thing. From facet, we can derive port which is a specific place on the surface where an interface can be formed. The idea of port is fundmental in the essence of a component as it is the place where the component capabilities are accessed.

The same essential approach can be used to move from assembly of things being a thing to the more formalised component system pattern.

A component can be physical or abstract functional. All components have some active influence on their environment (unlike a specification which is an informational thing and is inherently passive). The generalized abstract functional component is a pruned form of the generalized component. It includes all possible behaviours. It is still too general to apply meaningfully and requires further pruning.

8.2. Component to specific termination point

A termination point as per [RFC8345] is a specific pruned functional component that offers at its ports a defined subset of all possible functions. It does not offer the capability to forward information over great distances but does offer the ability to provide access to a flow of information at a specific place. In other standards [ITU-T G.7711] the LogicalTerminationPoint has roles including in one direction processing an incoming flow determining timing and framing and extracting the content "payload".

The termination point is still general and requires refinement to represent what is really feasible and useful in a network deployment context.

Up to this point refinement was carried out via pruning and refactoring where each level resulted in an explicit relabelling Thing -> Component -> TerminationPoint. Traditionally, the same orientation of progression was considered as a process of classification where properties were added as opposed to removed and the process continued beyond this point to highly specialised classes.

In the approach realized via [RFC8345] and [ITU-T G.7711], further refinement is carried out by augmentation. Here augmentation essentially exposes properties that were already encompassed by the definition of the thing being augmented. It is not an extension, it is an exposure of underlying properties.

So a termination point that processes photons is represented via an augmentation of the generalised termination point. Likewise, the termination point that process Ethernet is represented via an equivalent augmentation. Clearly, an augmentation of a termination point with photonic and Ethernet properties is not rational.

This is where the specification becomes critical. Each specific realization of a termination point type in software or hardware will be distinct. Just because it is an Ethernet termination point type does not mean it is the same as all other Ethernet termination point types. Of course, there will be many many instances of the type and they will have identical functional capability.

Setting out the distinct capabilities of the type is the role of the specification. The specification will be constructed by assembling pruned and refactored specifications of more complete definitions. So, for example, the Ethernet standard may define MEP and MIP capabilities, but the type of termination point may only support MEPs and there may be 7 levels of MEP in the standard, but this termination point type may only support 1 level where the measurements available to the MEP may be limited and a specific measurement constrained in range. All of this detail is available via the specification.

Armed with the specification a controller can determine precisely how the termination point can be applied in a solution and the range of opportunities available.

Whilst designing a solution, the controller may use a specific type of termination in a restricted form. For example, the Ethernet termination, although capable of supporting a MEP may be required to not provide that capability. The design of the pattern of use of terminations in a system may utilise the same type several times in the pattern where each occurrence in the pattern has a distinct further narrowing of the capability of the type. This is discussed further in Specification of an assembly (add reference to section).

Eventually the pattern will be realized in a network. This will first be designed with no real instances in place. This will be represented with further specific narrowed termination point occurrences. Finally, there will be real instances in the network. These can also be considered as occurrences.

8.3. Further examples

-Thing to Component to physical thing to equipment to specific equipment type to use of that equipment to instance of equipment -A plug example Circle back and relate this more rigorous section to the specification examples.

9. Specification of an assembly

Build on the examples and the recursive pruning and refactoring to explain the subtle narrowings in a system/scheme spec. Describe the essential process. Use examples to illustrate the progression: - Same examples as recursive pruning and refactoring but focus on role and subtle specializations in role List other examples.

10. Generalization of the specification

Build a specification structure from the examples and show the references and reuses. Explain how the specification relates to the things in the problem space. Lay out the specification structure.

11. Characteristics of a language of specification

The language needs inherent capabilities (as opposed to after the fact bolt-on warts) Extract key characteristics from above and from mobo - narrowing requires specific redefine (relate to pruning) - occurrence is an assembly of constrained type and specific values - need to reference other specs as reusable parts - refactoring, minor specialization and assembly - interrelationship and influence - uncertainty and preferences (Need to review mobo and TR-547 spec, component-system etc.)

12. Specification language options

Landscape of languages... does anything do this? Take YANG and enhance (as discussed in mobo)

13. Building a specification structure

Tooling and support to build and interrelate. Catalogue/library of specs Deep application... machine interpretable structure in all standards Use of AI to reverse engineer specs with guidance... peer review and testing cycle

14. A specification evolution example

Discuss how a spec may change as understanding emerges and how it may be refactored.

15. A system specification example

Take the language considerations and set out system specs in a more formal way

16. Broader Application of the Language

Negotiation Refinement of planning Development of standards Expression of uncertainty and pattern

17. Conclusion

Mindset Change Language challenges Use of AI Target is an ecosystem of specs driving agentic components...

18. Security Considerations

TBD

19. IANA Considerations

This document has no IANA actions.

20. References

20.1. Normative References

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

20.2. Informative References

[BaseInventory] Yu, C., Belotti, S., Bouquier, J., Peruzzini, F., and P. Bedard, "A Base YANG Data Model for Network Inventory", Work in Progress, Internet-Draft, draft-ietf-ivy-network-inventory-yang-14, 5 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-ivy-network-inventory-yang-14>.

[ITU-T_G.7711] "Generic⋯.", 31 August 2022, <https://www.itu.int/rec/T-REC-G.7711/recommendation.asp?lang=en&parent=T-REC-G.7711-202202-I)>.

[ivy] "ivy", 31 August 2022, <https:// 3.pdf>.

[LF_TAPI] "Transport API", n.d., <https://github.com/Open-Network-Models-and-Interfaces-ONMI/TAPI-Home>.

[mobo] "draft-davis-netmod-modelling-boundaries", 31 August 2022, <https:// 3.pdf>.

[ONF_TR-512] "TR-512 Core Information Model (CoreModel) v1.5", n.d., <https://opennetworking.org/wp-content/uploads/2021/11/TR-512_v1.5_OnfCoreIm-info.zip>.

[ONF_TR-512.7] "TR-512.7 Specification", n.d., <https://opennetworking.org/wp-content/uploads/2021/11/TR-512_v1.5_OnfCoreIm-info.zip>.

[ONF_TR-512.8] "TR-512.8 Control", n.d., <https://opennetworking.org/wp-content/uploads/2021/11/TR-512_v1.5_OnfCoreIm-info.zip>.

[ONF_TR-512.A.2] "TR-512.A.2 Appendix: Model Structure, Patterns and Architecture", n.d., <https://opennetworking.org/wp-content/uploads/2021/11/TR-512_v1.5_OnfCoreIm-info.zip>.

[plug] "plug", 31 August 2022, <https:// 3.pdf>.

Appendix A. Appendix A: Interpretive Notes on Refinement and Occurrence

A.1. A.1 No Single Refinement Path

In this modeling approach, there is no single correct way to refine a universal component. The refinement process supports multiple valid paths, each representing a different semantic purpose, level of granularity, or domain context. What emerges depends not on a fixed taxonomy, but on the alignment of constraints, intent, and reuse patterns.

This enables: - Coexistence of multiple specification layers derived from the same abstract element, - Domain-specific “semantic phases” that are meaningful within a particular stack (e.g., optical vs packet), - Purpose-driven modeling: e.g., one path for plug manifests, another for logical topology.

A.2. A.2 Occurrence at Every Layer

Occurrences are not limited to final instances. Each meaningful stage of refinement produces an occurrence—an intent-aligned, constrained projection of the universal component. Even so-called “instances” are not full realizations, but expressed intent within a given operational context.

A.3. A.3 Sweating Out the Shape

Useful structural forms (e.g., an LTP) are not pre-classified primitives. They _emerge_ from the pruning process when remaining capabilities reach a “sweat spot” of balance—enough constraints to be meaningful, but not so much as to be frozen. This allows the model to remain adaptive while still supporting mapping, reasoning, and automation.

A.4. A.4 Classification Considered Harmful

Rigid classification schemes tend to obscure natural emergence and lead to artificial separations. This model rejects top-down typing in favor of bottom-up capability surfacing, grounded in refinement logic. Semantic rigor replaces taxonomic rigidity.

Appendix B. Acknowledgments

This document has been made with consensus and contributions coming from multiple drafts with different visions. We would like to thank all the participants in the IETF meeting discussions.

Contributors

Nigel Davis
Ciena
Email: ndavis@ciena.com

Authors' Addresses

Nigel Robert Davis
Ciena
Email: ndavis@ciena.com

Camilo Cardona
NTT
Email: camilo@gin.ntt.net

Diego Lopez
Telefonica
Email: diego.r.lopez@telefonica.com

Marisol Palmero
Independent
Email: marisol.ietf@gmail.com