Received: 17 August 2019 | Revised: 15 February 2020 | Accepted: 16 March 2020

DOI: 10.1002/eng2.12168

RESEARCH ARTICLE

WILEY

Using directed graphs to define viewpoints to keep a metamodel, an architecture framework and views using different modeling languages consistent

Nic Plum

Eclectica Systems Ltd., Cambs, UK

Correspondence
Nic Plum, Eclectica Systems Ltd., Cambs, UK.
Email: nic@eclectica-systems.co.uk

Abstract

Keeping an architecture description of a complex system consistent is difficult. View content is specified by viewpoints which draw upon an underlying metamodel. Viewpoints need to define consistency rules. This becomes increasingly hard as the metamodel size and the number of viewpoints increase. Use of an architecture description language (ADL) such as the Unified Modeling Language (UML) complicates this further because its metamodel may not be able to implement the triples in the AD metamodel and therefore one or more views requiring use of an additional ADL. Even for an architecture framework with a small footprint, such as TRAK, it is a time-consuming and error-prone process to keep the metamodel, viewpoint definitions, and implementation consistent. To tackle these TRAK has been modeled by creating views against five viewpoints. These viewpoints can be used to define a metamodel, define the viewpoints for an architecture framework, assess the implementation using one or more ADLs, and record model changes over time. The viewpoint definitions and views are wholly graph-based and provide a mechanism to define allowed and minimum view content. Views and viewpoints can be read as simple assertions requiring no technical knowledge and support query and analysis. The viewpoint definitions have been validated against a model of TRAK held in a Neo4J graph database. This showed that the TRAK metamodel contains c 750 triples not 500 as had been previously thought. It also identified metamodel elements missing from one of the UML profiles. Example views are provided together with CYPHER queries.

KEYWORDS

architecture description, graph, ISO 42010, mbse, metamodel, Neo4j, TRAK, viewpoint definition, viewpoint-based-modeling

1 | INTRODUCTION

TRAK[1] is an open source architecture framework aimed at systems engineers that complies with the international standard for architecture description, ISO/IEC/IEEE 42010:2011.[2] Originally released in 2010 it is specified by three documents—the metamodel,[3] the viewpoints[4] that use the tuples from the metamodel and the overall requirements.[1] These specifications are solution independent in keeping with systems engineering practice—they do not specify the content in terms of any particular architecture description language (ADL). The choice of the ADL is left to the user. There is also an implementation specification[5] that specifies how to implement the TRAK metamodel elements and two Unified Modeling Language (UML) model implementations; a general UML profile[6] and a specific one[7] for the Sparx Systems Enterprise Architect UML modeling tool.[8] There is also a centrally-held spreadsheet[9] that maps the UML metamodel elements used in the UML profile against the triples in the TRAK metamodel and therefore how much of each of the 24 TRAK viewpoints (specifications for views in accordance with ISO/IEC/IEEE 42010) can be represented using the particular UML profile implementation. This spreadsheet has over 500 rows (one for each TRAK metamodel triple) and 24 columns (one for each TRAK viewpoint) and is manually maintained. These artefacts are all interdependent (Figure 1) and maintaining consistency and assessing impact is essential to both the specification and implementation of the architecture framework. Figure 1 illustrates why these dependencies exist—for example the metamodel definition is the master source of metamodel elements which are used by the implementation specification and the spreadsheet that maps the TRAK metamodel elements against the UML metamodel elements selected for the UML profile for TRAK. These artefacts are produced and maintained separately and made available through separate project sites on Sourceforge.

The effort spent maintaining consistency across this set of artefacts depends on the size of the metamodel, the number of viewpoint definitions and the interconnectedness. TRAK has a small metamodel in terms of the number of metamodel node elements. The nodes are highly connected. It also has roughly half the number of viewpoints of an architecture

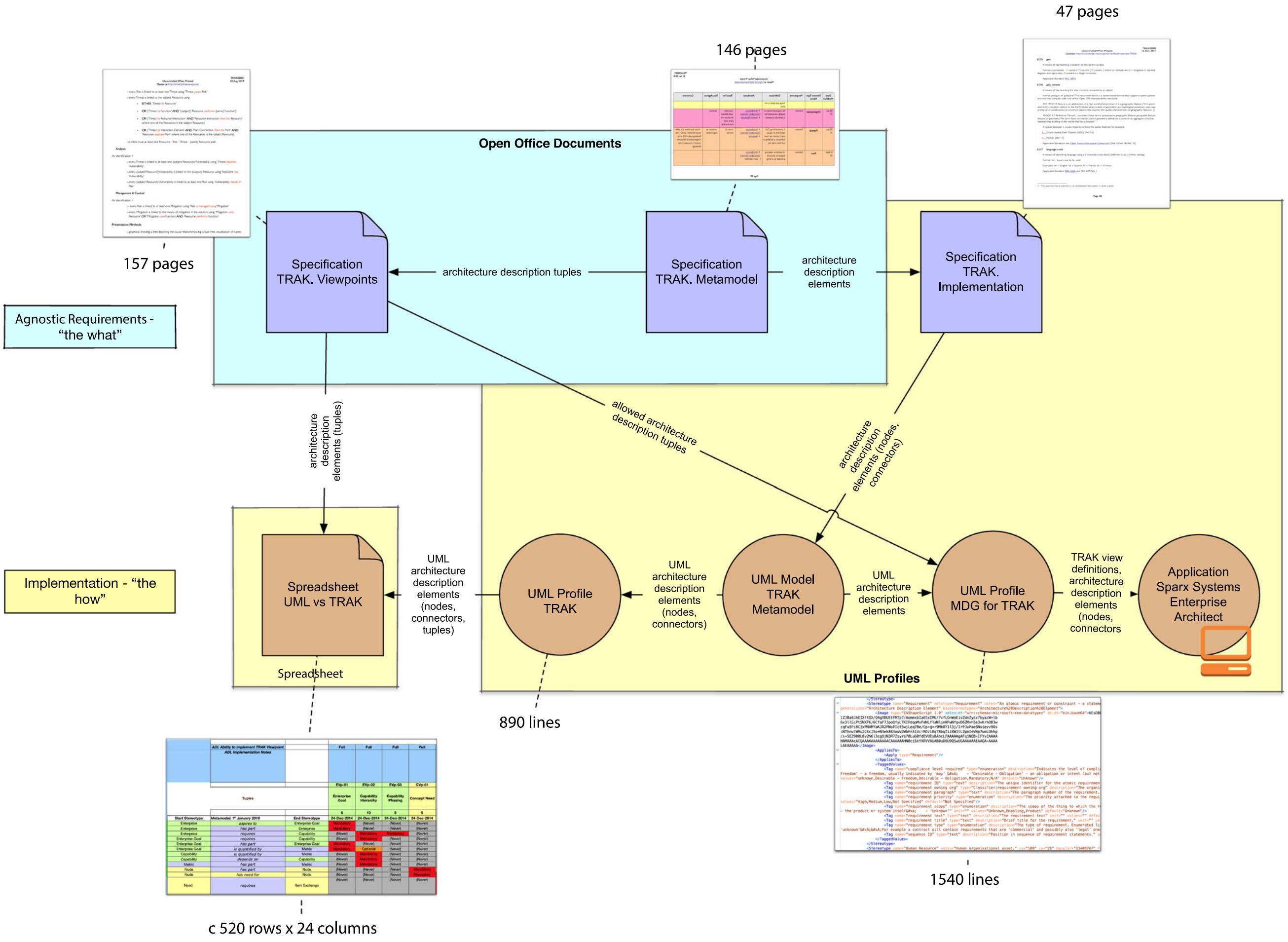

47 pages
146 pages
Open Office Documents
157 pages
Specification TRAK. Viewpoints
architecture description tuples
Specification TRAK. Metamodel
architecture description elements
Specification TRAK. Implementation
Agnostic Requirements - “the what”
allowed architecture description tuples
architecture description elements (tuples)
architecture description elements (nodes, connectors)
Implementation - “the how”
Spreadsheet UML vs TRAK
UML architecture description elements (nodes, connectors, tuples)
UML Profile TRAK
UML architecture description elements (nodes, connectors)
UML Model TRAK Metamodel
UML architecture description elements
UML Profile MDG for TRAK
TRAK view definitions, architecture description elements (nodes, connectors
Application Sparx Systems Enterprise Architect
Spreadsheet
UML Profiles
890 lines
1540 lines
c 520 rows x 24 columns

FIGURE 1 Need for consistency—dependencies between TRAK specification, implementation and centrally-published documents

framework such as the Ministry of Defence Architecture Framework (MODAF) now subsumed into the NATO Architecture Framework.[10] Even so manual maintenance is not only time-consuming but error prone and hence there is a need to integrate using a central model-based approach.

A model (architecture description) of the TRAK metamodel and the TRAK architecture viewpoints and the implementation as a UML profile is being created to formally integrate these artefacts and support analysis and their release. This has in turn required a set of five viewpoints to be created.[11] Two viewpoints describe a metamodel, one describes viewpoint definitions based on the metamodel, one describes the implementation of the metamodel and one describes changes to any model or metamodel element:-

1. Metamodel definition
    - Metamodel Element Structure Definition Viewpoint (5.2)
    - Metamodel Tuple Definition Viewpoint (5.3)
2. Viewpoint definition based on the metamodel
    - Viewpoint Definition Viewpoint (5.4)
3. Metamodel implementation
    - ADL Implementation Viewpoint (5.5)
4. Record changes to any model or metamodel element over time
    - Model Configuration and Change Viewpoint (5.1)

Views prepared against these viewpoint definitions in this article have been used to produce a complete description of the TRAK metamodel and two of the TRAK viewpoints as a graph using the Neo4J graph database. Whilst TRAK has been used as an exemplar or test piece the viewpoint definitions provided can be used to define any metamodel, viewpoint definitions using that metamodel and the implementation in a modeling language. The resulting model is able to be checked for errors in the definition of the metamodel, viewpoints and the implementation(s) of the metamodel. The Model Configuration and Change Viewpoint can be used with a metamodel and a model (see Section 3.6) and allows changes to be recorded within the respective model itself rather than using a separate versioning or "diffing" tool as is typically the case.

The model of TRAK is able to support verification of the extent to which view content using the UML profile addresses each stakeholder concern. It also allows checks to be made of the use of each metamodel triple in the TRAK viewpoint definitions to assess the viewpoint (view) overlap and coverage of the metamodel. A mechanism is provided to define the allowed and minimum view content and any consistency rules. The viewpoint definitions and the resulting view set and model as a directed graph can be easily read and support querying and analysis as a set of assertions.

Section 2 discusses the work undertaken in the context of related work.

Section 3 introduces architecture description—the primary standard, architecture frameworks, the architecture descriptions languages used to implement architecture frameworks and the tuple as the smallest unit of architecture description—itself a graph.

Section 4 provides a description of the structure of a viewpoint definition.

Section 5 provides an overview of each of the five viewpoint definitions produced:

1. Model Configuration and Change Viewpoint
2. Metamodel Element Structure Definition Viewpoint
3. Metamodel Tuple Definition Viewpoint
4. Viewpoint Definition Viewpoint
5. ADL Implementation Viewpoint

The Model Configuration and Change Viewpoint can be used to produce views that record changes to any model. The Metamodel Element Structure Definition Viewpoint followed by the Metamodel Tuple Definition Viewpoint enable a metamodel to be defined. This is then used by the Viewpoint Definition Viewpoint to define one or more viewpoints. The ADL Implementation Viewpoint then provides a means to systematically check an implementation using an ADL such as the UML and the suitability of the ADL to represent the viewpoints defined. The set of viewpoint definitions provide the means to systematically define a metamodel, viewpoints using the metamodel, and assess any implementation and then record the changes to these definitions over time.

The use of each of the viewpoints is presented in Section 6 with examples of views produced using graph queries run against a model created using the viewpoints presented of the TRAK architecture framework metamodel and TRAK viewpoint definitions. The results are returned as either graphs or tables and show how the views can be produced to describe any metamodel, describe viewpoints based on this metamodel, and check for potential errors. Once a model has been created it is then possible to assess how well suited a particular modeling language is to implement the (modeled) metamodel that is, to what extent does it contain and allow the necessary triples to be created that are required for a particular viewpoint?

Section 7 outlines how the viewpoints can be used to help define an architecture framework and a metamodel.

2 | RELATED WORK

Guychard et al[12] propose model federation to tackle the problem of trying to integrate domain-specific languages used within system design. They identify that multiviewpoint models and architecture frameworks have problems with consistency but provide no rules to manage it. They identify the lack "of any existing approach that provides ways to ensure consistency across heterogeneous modeling artifacts and tools." The use of the UML and extension using UML profiles requires expert knowledge and causes problems with semantic alignment and the symbols used to communicate meaning. The work presented in this article shows that directed graphs can be used to define viewpoints and form a model and views that are understandable and form a set of assertions that can be analyzed. Furthermore using graphs/architecture description triples it is possible to map different metamodels back to a single viewpoint and define consistency rules.

Fischer et al[13] identify that viewpoints will share model elements and therefore that there is an interdependency between viewpoints which requires consistency rules to be applied to views. In the terms used this work contains "an own viewpoint definition" (see the Viewpoint Definition Viewpoint) and "impact analysis features" (see the Model Configuration and Change Viewpoint). All of the viewpoints presented are based on a single metamodel (Appendix—Complete Metamodel) which provides "projection features."

Holt and Perry[14] define six viewpoints to define an architecture framework of which one viewpoint describes a metamodel and two viewpoints describe a viewpoint. It provides no mechanism to define rules for consistency or minimum view content. Baroni et al[15] use a semantic wiki in collaboration with a model-driven engineering (MDE) model to address the problem identified by Malavolta et al[16] that ADL need to be simple and intuitive to allow effective communication yet formal and structured to allow analysis. The work presented in this article shows that by defining viewpoints and view content using solely triples and holding the model as a graph in a graph database that it is possible to achieve both readability of view content whilst maintaining a semantic model that is amenable to analysis and ad hoc queries.

In 2006 Wenzel and Kelter describe a means to compare two models and characterize the differences as a graph.[17] Differences are characterized as "attribute difference," "reference difference," "move difference," and "structural difference." This makes the changes evident but it is left to the user to deduce what the differences are. It does not link the changes to any rationale because in comparing changes after the fact this may not be obvious. The Model Configuration and Change Viewpoint presented in this article provides the means to produce views that explicitly describe changes to model elements linked to a version of a model and a set of change requests. It also provides a means to describe the complete lifecycle of any model element.

The work shows that viewpoint definition and therefore view content and the addressing of stakeholder concerns can only be achieved by using tuples or triples—using nodes produces an ambiguous result. The viewpoint definitions presented in this article and in TRAK are distinct in using only triples as the means of specification of view content and consistency rules.

3 | ARCHITECTURE DESCRIPTION

3.1 | Standards—ISO/IEC/IEEE 42010:2011

ISO/IEC/IEEE 42010:2011[2] is the current standard that applies to the description of a system architecture. It defines architecture as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution" and an architecture description as "work product used to express an architecture."

It is based on the idea that systems are complex and their description requires multiple overlapping views, each describing a particular aspect that addresses a particular set of concerns. This separation of concerns is used to define one or more viewpoints, each of which is a specification against which a view is prepared and interpreted. The use of "viewpoint" as a specification against which a view is created and implemented is fundamental to the means to define view content and consistency rules. It is also distinct from the use of "viewpoint" to refer to a collection of view definitions. Indeed one of the objectives of any standard such as ISO/IEC/IEEE 42010 is to standardize terminology and reduce this inconsistency where the same term is used to describe different concepts.

ISO/IEC/IEEE 42010:2011 specifies requirements that apply to architecture descriptions, viewpoints, and architecture frameworks. It does not specify any particular means to achieve them. The structure of the viewpoint definitions in Section 4 is based on the structure used for the viewpoint definitions in the TRAK architecture framework and is designed to address the requirements of ISO/IEC/IEEE 42010:2011.

3.2 | Architecture descriptions

ISO/IEC/IEEE 42010:2011 defines an architecture description as the "work product used to express an architecture." They can be produced using a wide variety of modeling languages or notations. In the systems engineering domain they can be quite specific such as a set of fault trees or a set reliability block diagrams to describe the robustness or dependability of a system. Equally an architecture description can be used to describe a business, its organizational parts, and the processes and resources that support this. The diagrams represent partial views of the underlying model and indeed ISO/IEC/IEEE 42010:2011 identifies this idea where a complex system is described using many views, each of which addresses a particular set of concerns as "separation of concerns."[18]

As the architecture description gets larger and more views are produced it becomes harder to maintain consistency across the collection. Modeling tools can help, for example by allowing the user to present the same object in many views so that any changes to the object appear in all of these views. If a general-purpose modeling language is used it can be difficult to ensure that each modeler in a team uses the same type of element from the modeling language to represent the same type of real-world entity. General purpose modeling languages allow a wide variety of content to be shown on any particular view which makes it difficult to produce consistent views without defining some rules. A viewpoint provides the mechanism to define rules for content and consistency. If each modeler has freedom to choose the relationship between elements this is another potential source of inconsistency. This is one of the reasons why architecture frameworks are sometimes used—they define viewpoints that deliberately restrict the choices available.

3.3 | Architecture frameworks

ISO/IEC/IEEE 42010:2011 defines an architecture framework as "conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders." They typically define a set of shapes (nodes) and connector elements that can be used to describe a system and its context and they define view content. In effect they provide a controlled grammar because the user is restricted to using only the defined set of elements in the combinations allowed. Many modeling languages are often very general and offer so much choice that different modelers make different choices in the selection of elements to create views. This is a source of inconsistency that bedevils this area. An architecture framework in defining a restricted set of elements is helpful because it limits choice. The elements may therefore reflect the language of the domain in which the architecture framework originated, for example the MODAF[19] includes military-related elements such as Actual Post, Capability, Enduring Task, and Mission. Similarly The Open Group Architecture Framework (TOGAF),[20] which originated within the IT sector, includes Actor, Application Component, Business Service, and Value Stream.

An architecture framework is often defined in a way which is agnostic of the modeling language (Section 3.4). Done this way it provides the end user with the ability to choose which modeling language best suits their needs and the modeling tools at their disposal.

The architecture framework used to provide sources against which the viewpoint definitions in this article (Section 5) were tested is TRAK.[1] TRAK is an open source architecture framework that originated from work within the systems engineering discipline at London Underground Limited although it contains no rail-specific architecture description elements. TRAK complies with ISO/IEC/IEEE 42010:2011—a compliance assessment of TRAK as an architecture framework, its viewpoints and any TRAK-conforming architecture description is provided to support the claim.[21]

TRAK defines 24 viewpoints split amongst five perspectives: Enterprise, Concept, Procurement, Solution, and Management. The perspectives provide an overarching subject area, for example the Procurement Perspective covers includes project, project activity, and milestone and relationships. Each has a "xVp-nn" identifier, which identifies the perspective, for example, "CVp-01 Concept Need," "SVp-01 Solution Structure Viewpoint," and "MVp-04 Assurance Viewpoint" where the first letter refers to the perspective that is, "C" = Concept Perspective, "M" = Management Perspective. The viewpoint definitions use the TRAK metamodel to define the allowed and minimum acceptable view content. In effect the TRAK metamodel defines a controlled grammar as a set of assertions (see Section 3.5) and each viewpoint definition is formed from a subset from which the user can produce architecture views to describe their particular system of interest.

3.4 | Architecture description languages

ISO/IEC/IEEE 42010:2011[2] defines an ADL as "any language for use in an architecture description." In practical terms a modeling tool, such as an enterprise architecture tool provides the user with the means to use a particular ADL. Many enterprise architecture tools are based on a particular ADL such as the OMG UML.

Where an architecture framework such as TRAK, is defined in a "solution-agnostic" manner this means that the definition is independent of any particular ADL. The metamodel and viewpoint content are not defined using a particular ADL. This separation of "the what" (problem) from "the how" (design response) is in accordance with typical systems engineering practice (ISO/IEC/IEEE 15288).[22] Separating the notation used to specify the metamodel and viewpoints from the notation used for implementation as view content (which might be text, directed-graph, UML, and so on) avoids any possible common-mode failure where an error or gap in a notation affects both the requirement and the design response to the requirement. Again, this represents good system design practice.

Using a notation or modeling language to implement a viewpoint definition involves selection of elements from the modeling language's metamodel. Inevitably this involves trade-offs—it might be that the chosen elements are semantically correct but the modeling language definition does not permit them to be combined to represent the tuples in the metamodel being implemented. It might be that the modeling language specifies behavior that makes modeling easier for the architect/modeler but only for specific combinations of elements that do not properly represent the required semantics. It might be that the modeling language just does not include the necessary metamodel elements to form the tuples required. For any one particular modeling language there are always many possible ways of implementing the required metamodel.

A UML modeling tool can be used to produce a view that conforms to a viewpoint specification. A common way is to define a UML profile which defines a set of node and connector elements which are based on the UML metamodel. The UML modeling tool loads the UML profile and provides the user with palettes of elements from which they can produce the views. As a set of customized UML elements a UML profile does not define view content nor what connector can be joined to what node(s) beyond the behavior in the modeling tool. This behavior is defined by OMG UML specifications that define the UML.

3.5 | The architecture description tuple—a graph

ISO/IEC/IEEE 42010:2011[2] defines the relationships between a system of interest, its architecture, and architecture description within a conceptual model as the context for understanding architecture description (Figure 2).

Figure 2 presents two statements or assertions—"System EXHIBITS Architecture" and "Architecture Description EXPRESSES Architecture." The arrows define the directions in which each statement is to be read. Each of these is formed from two nodes and a connector. This representation is a mathematical graph.[23] In graph theory a node is termed a vertex and the connector is termed an edge.

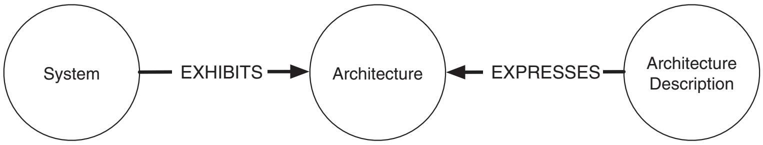


FIGURE 2 Relationships between system, architecture and architecture description from ISO/IEC/IEEE 42010:2011

In mathematics a tuple is an ordered sequence of elements.[24] The node—connector—node notation is a graph and in addition each statement forms an ordered subject—predicate—object tuple (sometimes referred to as a triple because it contains three items). Triples are the basis of the Resource Description Format (RDF)[25] which is used to represent information on the World Wide Web. RDF triples provide the means for machine-understandable assertions to be made and construction of the Semantic Web.[26]

Since direction is defined, Figure 2 is a directed graph. Directed graphs can be used to represent a model or a metamodel (Figure 4). They provide a very simple way to describe the relationships between things.

Triples/graphs are useful because they can be read as simple textual statements and therefore no particular technical knowledge is needed. Each triple forms an assertion. An architecture view formed solely from triples can be represented using graphs/RDF and has the potential to be machine-understandable. All of the architecture views provided consist wholly of triples. This is unusual because many notations such the UML or SysML allow containment, proximity, and docking of elements to infer relationships and therefore views to be created that are only partial graphs.

Since ISO/IEC/IEEE 42010[2] defines architecture in terms of relationships between a system, its parts, and the environment it follows that the smallest unit of architecture description must include a relationship and is therefore node—connector—node (Figure 3). There is therefore a natural fit with a triple and its presentation as a graph.

3.6 | Metamodeling

Bézivin[27] states that in MDE "a particular view (or aspect) of a system can be captured by a model and that each model is written in the language of its metamodel." When applied to architecture description of a real-world entity using an architecture framework such as TRAK there are three levels of model (Table 1) from a model element (M1) describing or representing a real world thing (M0) to the nodes and connectors from which a metamodel (M2) is built.

Models and metamodels with a graphical presentation always include node elements and connector elements. The graphic presentation may vary and typical examples of models used within systems engineering include:

- Fault tree—where the node elements represent events and the Boolean logic applied the combination of child events needed for the parent event to occur.[28]

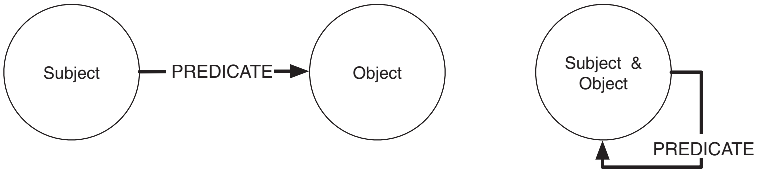


FIGURE 3 Tuple (triple)—smallest unit of architecture description

TABLE 1 Abstraction—model levels and characteristics

Model level	Entity	Characteristics	Typical stakeholder	Example
M0	Real World Entity		Driver, Maintainer.	Autonomous Vehicle
M1	Model element representing real world entity	The model may contain hundreds or thousands of model elements.	System Designer/Architect, System Engineer	A TRAK System element in an architecture description (representing the autonomous vehicle)
M2	(Meta) model element representing the type of element in the model of the Real World Entity	Population of the set of metamodel nodes and connectors—the number of tuples in the metamodel. Description of particular elements allowed in a model.	Metamodel Developer, Architecture Framework Maintainer	The TRAK metamodel System element
M3	A node—used to represent the metamodel element.	Small population of elements needed to describe a metamodel element	Metamodel Developer, Architecture Framework Maintainer	Node (used to describe the TRAK metamodel System element)

- Reliability Block Diagram—where the node elements represent each part of the system of interest that must work for the system of interest to achieve its purpose. The connectors describe functional connectivity and any parallel connectivity where there is redundancy.[29]
- Assurance model—where the node elements represent Claims, Arguments, and Evidence and the connectors show how Arguments support/oppose Claims and Evidence supports/opposes Arguments. A typical claim might be that a design satisfies a specified requirement. The arguments together with supporting evidence might be summarized as a compliance matrix.

Each model has an underlying metamodel that defines the node and connector elements and how they connect.

For ADL such as the UML or the SysML the (M2) metamodel is specified in its respective specification document.[30,31] In systems engineering terms these are design specifications against which one of the design responses is the respective UML profile used by a UML modeling tool to provide the user with the node and connector elements from which they can create their (M1) model. Another part of the design response against the respective specification sits within the modeling tool itself that implements the required tool behaviors and visualization.

3.7 | The TRAK metamodel

The TRAK Metamodel[3] used to test the metamodel viewpoint definitions in Section 6 of this article is defined using a simplified entity—relationship (graph) notation supported by tables that define each TRAK metamodel node and connector element and their properties. Figure 4 provides an example of the presentation of part of the TRAK metamodel. As a set of node—connector—node elements it demonstrates how a set of tuples (triples) can be used to form a metamodel. It shows the relationships associated with the Claim—Argument—Evidence elements used to describe the assurance of a system of interest.

Table 2 provides an example of the presentation of the definition of two of the metamodel elements—Argument and Capability—showing how a metamodel element and its properties are defined.

4 | ARCHITECTURE VIEWPOINT STRUCTURE

In ISO/IEC/IEEE 42010 architecture viewpoints are products that define requirements for the construction, interpretation, and use of architecture views. In accordance with Reference 2, each architecture viewpoint frames a defined set of stakeholder concerns about a system of interest. In this case the system of interest is a metamodel, viewpoint definitions using the metamodel and any modeling language(s) used to create views against the viewpoint definitions. Viewpoint definitions are defined using triples from the respective metamodel.

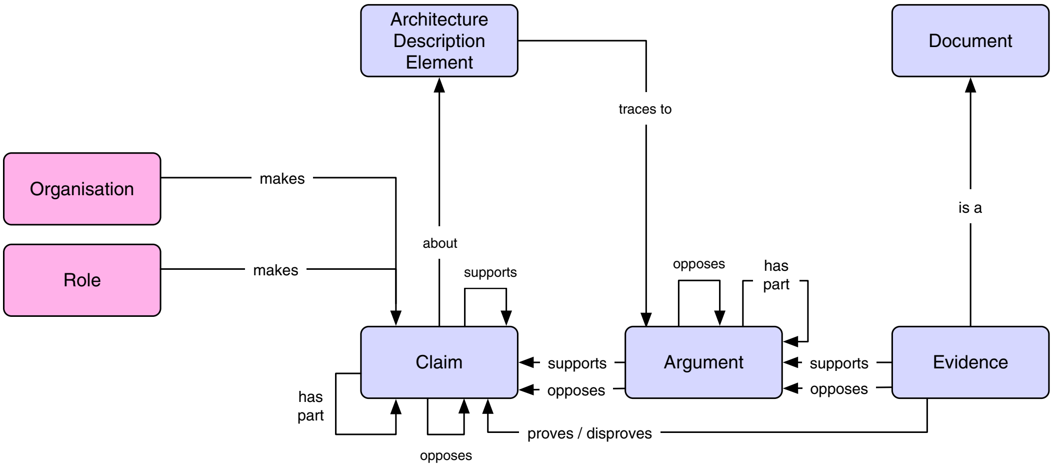


FIGURE 4 Part of the TRAK metamodel (M2 level) showing simplified presentation

TABLE 2 Part of the TRAK metamodel—definition of element and properties

Date modified	Element type name	Perspective	Definition	Attributes	Tests for	Tests against	Comment
				• viewpoint identifier			
23, July, 15	**Argument**	Management	A connected series of statements or reasons intended to establish a position (and, hence, to refute the opposite); a process of reasoning argumentation	+Architecture Description Element			• OED http://www.oed.com/view/Entry/10663 (accessed August 8, 2013)
27, Jan, 11	**Capability**	Enterprise	The ability to undertake a particular kind of action or the extent of someone's or something's ability.	+Architecture Description Element	Makes sense when used with "Enterprise requires … " easy to reuse	Includes technology includes a "thing"	• Keep short • Use active (doing) voice • Should be able to be reused easily across ADs/models

For the viewpoints in this article the metamodel at Appendix A is used and the relevant extracts are included in each viewpoint definition in Section 5 Architecture Viewpoint Definitions. Examples of triples from this metamodel include:

- "Architecture Description Tuple addresses Concern"
- "Architecture Description Tuple starts at Node"
- "Architecture Description Tuple uses Connector"
- "Architecture Description Tuple ends at Node"
- "Implementation uses Node to represent Node"
- "Implementation uses Architecture Description Tuple to represent Architecture Description Tuple"

Examples of triples taken from the TRAK Metamodel used to test the viewpoint definitions in this article[3] include:

- "Physical contains System"
- "Concern about Requirement"
- "Argument supports Claim"
- "Threat poses Risk to Software"
- "Organization plays Role extends to System"—to describe the extent of responsibility

Each viewpoint presented in this article is fully defined in a separate document.[11]

Since ISO/IEC/IEEE 42010:2011 defines a viewpoint as a specification against which a view is prepared and interpreted the description of such a viewpoint must therefore be able to define requirements and be able to describe rules against which a view is verified. This is why there are sections for Subject Tuples, Optional Tuples, Well-Formedness, and Consistency Rules in the viewpoint structure and why the Viewpoint Definition Viewpoint definition includes tuples such as:-

- "Architecture Viewpoint requires at least Architecture Description Tuple"—to describe mandatory minimum view content
- "Architecture Viewpoint allows Architecture Description Tuple"—to describe optional view content, for example to add additional context or narrative.

and provides the means to describe sequencing and concatenation of tuples.

The structure used to define each architecture viewpoint is the same as that used in the TRAK Viewpoint specification[4] consisting of:

- Version number
- Date created
- Date modified.
- Concerns addressed. These are the stakeholder concerns that define the purpose of the viewpoint and hence the tuples needed to address them.
- Anti-concerns. These are used to resolve potential confusion with other viewpoints by defining any concerns specifically not addressed by a viewpoint but typically addressed by another.
- Subject tuples. These are the core tuples needed to address the concerns. They form the basis of the well-formedness definition. They are also used to assess overlap between viewpoints—as the overlap increases is becomes harder for the modeler to select the most appropriate view and the more likely it is that different modelers will make different and therefore potentially inconsistent choices.
- Optional tuples. These are tuples that provide additional context and allow the reader to navigate between views. Often these are tuples that may appear in many views.

- Well-formedness. These define the minimum acceptable content for a view. This is needed to ensure consistency in presentation. It also aids the reader because there is an expectation of the content of a particular view that is, the affordance of a view in human factors terms. Rules are also needed to enable the coverage of the viewpoint definition set against the metamodel providing the tuples to be checked. Any metamodel tuple not required to appear in any viewpoint and therefore view needs to be verified since this is a potential error. The larger the metamodel and more interconnected the greater the cost of maintaining the metamodel and viewpoint definitions. In a metamodel it is likely that two nodes may be connected by two or more relationships, for example "System is configured with Physical" and "Physical contains System." Tuples therefore have to be used to provide an unambiguous definition of view content by specifying a unique path through the metamodel. Tuples have the benefit of being able to be read as sentences, provide an explicit set of assertions and support ad hoc querying of models and whole repositories by path-following queries.
- Presentation methods
- Examples
- Consistency rules. Correspondence rules are required by ISO 42010. Consistency rules include all forms of consistency—the need to include the same particular element (correspondence) and also consistent logic and semantics. The rules are defined between elements across a collection of views in order to keep the architecture description consistent. This might involve specifying that if an interface between two systems is characterized that the description of this must include the same pair of system elements used to identify the interface beforehand. They may involve the sequence in which tuples or assertions are created to avoid users circumventing a natural order for example, asserting that a claim is proven before making any arguments or providing any evidence.
- Comments

An example of one of the viewpoint definitions produced—the Viewpoint Definition Viewpoint (Section 5.4) is provided in Appendix B at B.1. An example of a particular TRAK Viewpoint, the TRAK MVp-04 Assurance Viewpoint, used as a test article is provided in Appendix B at B.2.

5 | ARCHITECTURE VIEWPOINT DEFINITIONS

Five viewpoints have been defined. The Model Configuration and Change Viewpoint can be used with both a model (M1) and a metamodel (M2). The Metamodel Element Structure Definition Viewpoint and the Metamodel Tuple Definition Viewpoint define a metamodel. Its implementation using an ADL such as the Universal modeling Language (UML)[30] is defined using the ADL Implementation Viewpoint. The outline characteristics of each are discussed below. The application of the viewpoints to a real-world example describing parts of TRAK with example view outputs and the queries to produce them is discussed in Section 6.

5.1 | Model configuration and change viewpoint

The concerns addressed are:

- What changes have been made since a specified date or version?
- Has anything been added or deleted?
- What changes have been made to a specified architecture description element?

Figure 5 shows the partial metamodel defining the viewpoint tuples.

The Model Configuration and Change Viewpoint can be used to record changes to any model—a representation of the real world (M1) or a metamodel (M2). It therefore differs from the other viewpoints which apply to describing metamodels.

In the context of recording changes to a metamodel this could include adding, removing or modifying a property or attribute of an element.

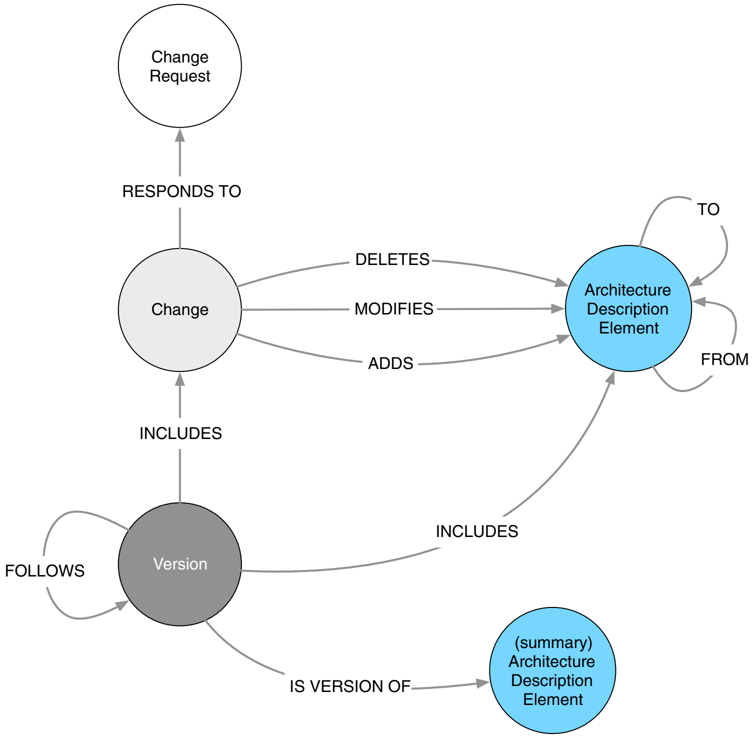


FIGURE 5 Model configuration and change viewpoint tuples

5.2 | Metamodel element structure definition viewpoint

The concerns addressed are:

- What are the entities and relationships?
- What are the properties and allowed property values for the entities and relationships?
- Are there properties or relationships that can be shared?

Figure 6 shows the partial metamodel defining viewpoint tuples.

The Metamodel Element Structure Definition Viewpoint is used to define the entities and relationships and their properties. The Metamodel Tuple Definition Viewpoint is then used to define the architecture description tuples formed from these elements.

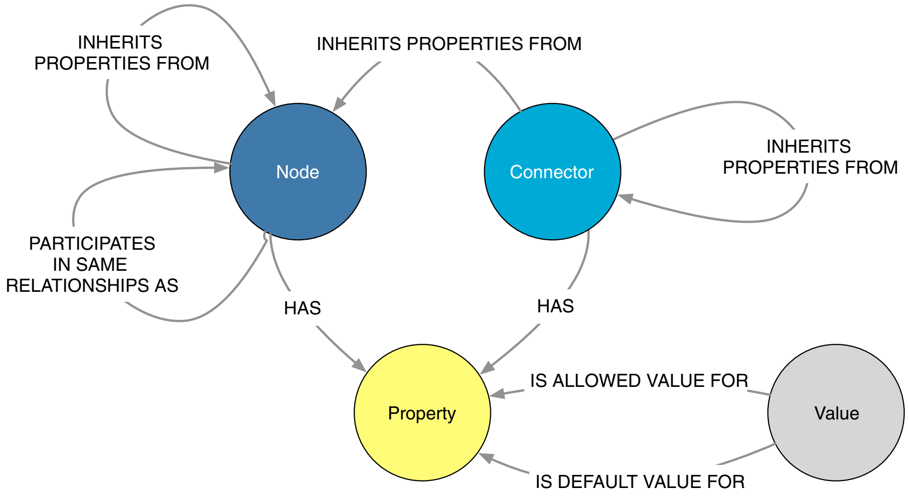


FIGURE 6 Metamodel element structure definition viewpoint tuples

5.3 | Metamodel tuple definition viewpoint

The concerns addressed are:

- What are the tuples within the metamodel?
- What is the direction of the tuple?
- Are all of the metamodel node and connector elements involved in tuples that is, are there any that are not used? This represents a quality check since it is an error to have orphan elements.
- How many tuples does each node and connector element appear in?

Figure 7 shows the partial metamodel defining the viewpoint tuples.

The Metamodel Tuple Definition Viewpoint takes the nodes and connectors defined in the Metamodel Element Structure Viewpoint and creates the tuples that comprise the metamodel (an architecture description cannot by definition include orphan nodes since ISO/IEC/IEEE 42010 defines architecture in terms of relationships:

“fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.”

The smallest permissible unit of architecture description is therefore a triple. A triple has to have an explicit direction. This is true if a connecting line is drawn because it needs a start point and end point. It is also needed to make sense of the assertion that results from the tuple itself for example,“Threat EXPLOITS Vulnerability” has only one direction in which the statement makes sense with Threat as the start point and Vulnerability as the end point. This is described using the “Architecture Description Tuple STARTS AT … or FINISHES AT … ” tuples. Since a tuple always involves at least one Connector this is asserted using “Architecture Description Tuple USES CONNECTOR Connector” tuple.

This describes a single tuple—a triple. It is possible that a longer assertion comprising multiple triples needs to be defined. This is described using “Architecture Description Tuple STARTS WITH (first) Architecture Description THEN (second) Architecture Description Tuple” and so on. The advantage of this is that this then provides the basis for any consistency or well-formedness rules for a viewpoint definition.

The “Node CANNOT BE JOINED TO Connector” tuple provides a means to record where a combination is not allowed. This supports an assessment of the implementation of a metamodel using an ADL such as the UML where, depending on the elements chosen to represent the solution-agnostic metamodel elements, it may prove impossible to represent the required tuple.

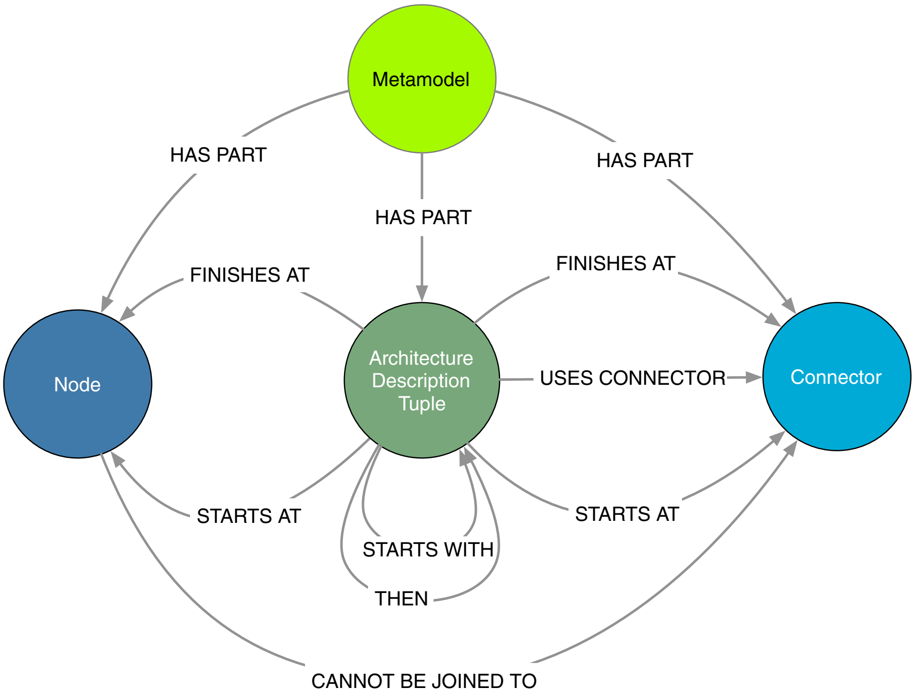


FIGURE 7 Metamodel tuple definition viewpoint tuples

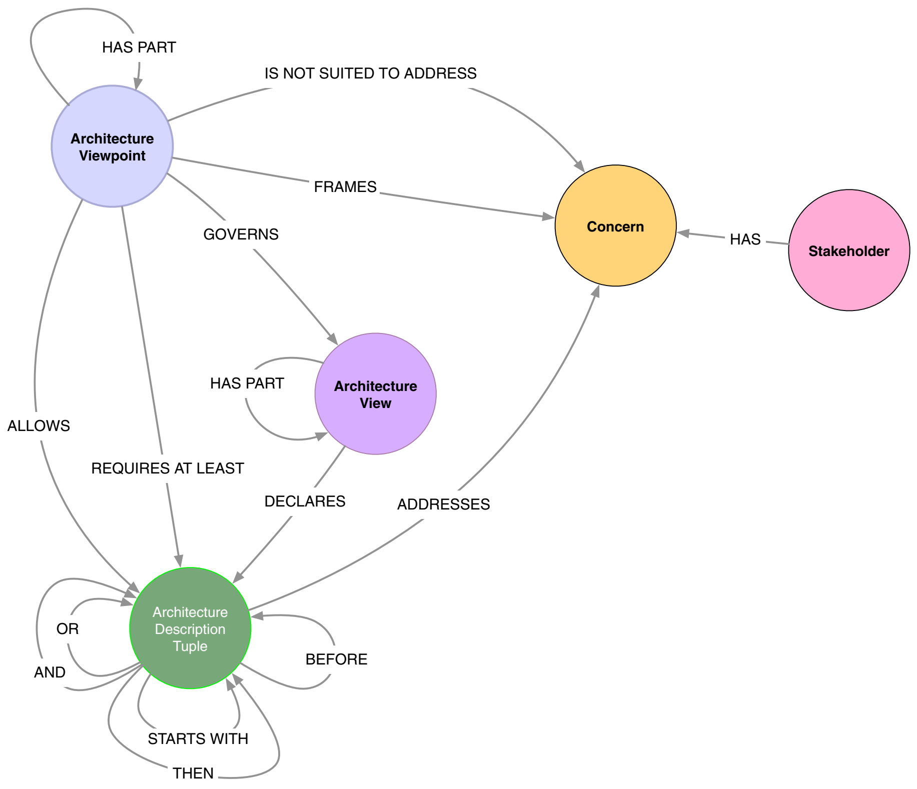


FIGURE 8 Viewpoint definition viewpoint tuples

5.4 | Viewpoint definition viewpoint

The concerns addressed are:

- What concerns does the viewpoint frame?
- Which stakeholders hold these concerns?
- What metamodel tuples are needed to address these concerns?
- What is the acceptable view content?

Figure 8 shows the partial metamodel defining the viewpoint tuples.

The "Architecture Viewpoint—FRAMES -> Concern," "Stakeholder—HAS -> Concern" and "Architecture Viewpoint—GOVERNS -> Architecture View" are taken from the conceptual model in ISO/IEC/IEEE 42010:2011.

The Viewpoint Definition Viewpoint is split into three parts—first defining the purpose of a viewpoint, then defining which tuples address the concerns identified and finally defining the tuples forming the minimum required and allowed view content.

The consistency of the viewpoint definition can be checked using the "Architecture Viewpoint—FRAMES -> Concern <- ADDRESSES—Architecture Description Tuple <- REQUIRES AT LEAST—Architecture Viewpoint" path. This provides a check that the tuples addressing the viewpoint concerns form part of the minimum acceptable view content.

5.5 | ADL implementation viewpoint

The ADL Implementation Viewpoint provides the means to assess how well any particular implementation of the metamodel is able to represent the set of tuples in the metamodel. This can only be achieved by comparing the tuples in the metamodel with those used in the ADL metamodel.

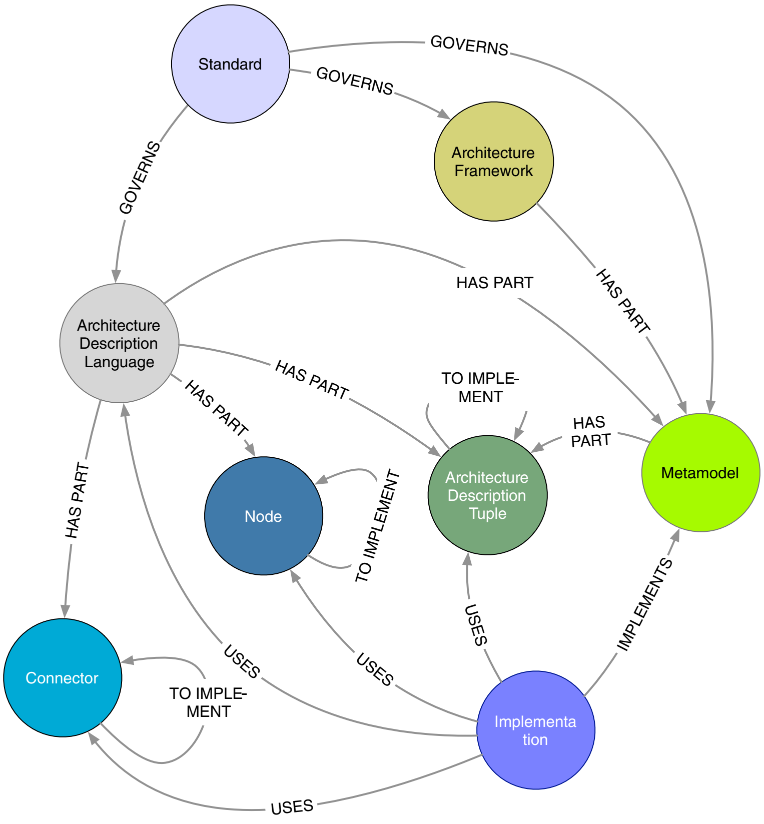


FIGURE 9 Architecture description language (ADL) implementation viewpoint tuples

The concerns addressed are:

- What ADL is used to implement the architecture framework?
- What element of the ADL is used to represent the architecture description element of the architecture framework?
- To what extent can architecture description tuples in the architecture framework be represented by the ADL?

Figure 9 shows the metamodel fragment defining the viewpoint tuples.

It is possible that more than one ADL is used and that there might in fact be more than one ADL used for a single viewpoint. This might occur, for example, because the primary ADL is unable to implement some metamodel tuples and as a result a stakeholder concern cannot be fully addressed by view content using this ADL. If the concern is essential to the architecture description task an additional ADL could be chosen to fill in the gap. There are, however, some practical problems that might be introduced if multiple ADLs are used. The ADLs might be implemented in different tools making integration of the architecture description difficult. It is also then a lot harder to check for completeness and enforce consistency rules across the architecture description.

This viewpoint provides a method to make an explicit mapping between the (agnostic) metamodel and the ADL metamodel to systematically assess the impact of gaps on the ADL ability to support viewpoints and therefore views. In providing a means to identify whether a suitable connector and nodes exist in an implementation separate from a tuple it provides a means to identify where the combination is not allowed in the ADL. The metamodel provides “Node CANNOT BE JOINED TO Connector” to make this explicit. In the metamodel definition there is a “rationale” property for the CANNOT BE JOINED TO relationship to record the specific reason.

6 | USE OF THE VIEWPOINTS TO DESCRIBE (MODEL) A METAMODEL, VIEWPOINTS, IMPLEMENTATION, AND CHANGES TO THE MODEL

The examples in this section include views produced against the viewpoints outlined in Section 5—Architecture Viewpoint Definitions. The views are produced using CYPHER queries and the resulting query is output as graphical or tabular views. Figure 10 shows an example of a graphical output. The CYPHER code is included in Appendix C so that replication of the results is possible.

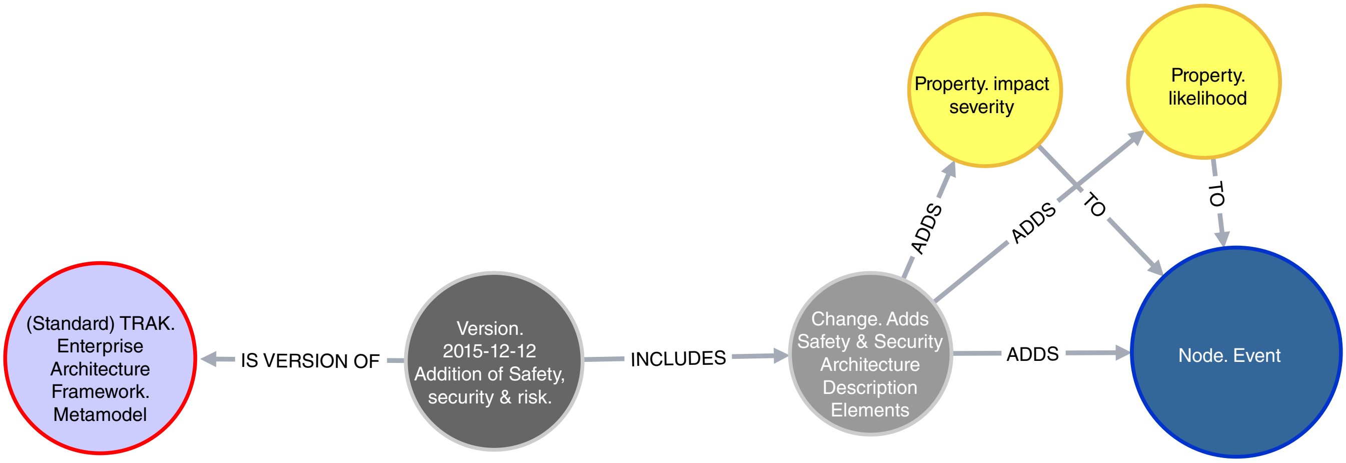

Property. impact severity
Property. likelihood
ADDS
ADDS
TO
TO
(Standard) TRAK. Enterprise Architecture Framework. Metamodel
IS VERSION OF
Version. 2015-12-12 Addition of Safety, security & risk.
INCLUDES
Change. Adds Safety & Security Architecture Description Elements
ADDS
Node. Event

FIGURE 10 Example view from CYPHER query (lifecycle of TRAK risk metamodel element)

A description of the TRAK metamodel has been produced using the Neo4J graph database.[32] In addition two of the TRAK viewpoints (MVp-04 Assurance Viewpoint and SVp-13 Solution Risk Viewpoint) have been modeled to validate the metamodel viewpoint definitions.

The advantages of using Neo4J are that the TRAK Viewpoints are themselves defined using directed graphs and that the model views can be defined as CYPHER[33] queries. No technical expertise is required to read the output because the graphs form natural English statements. CYPHER provides an easy to use mechanism to direct path traversals through the model to retrieve information and make consistency checks, for example, path sequences or path closure.

Having a wholly graph-based model at both the M1 (model) and M2 (metamodel) levels is a significant benefit because it allows the models/descriptions to be easily interrogated. One of the advantages of using graphs is that the description can include graphs presenting models at all three levels in the one description. This is in contrast with a UML model where there is a maximum of two levels if a class is instantiated (the UML metaclasses are part of a separate model that produces the UML profile providing the classes).

Another problem with using the UML is that whilst the UML metamodel (M2) is wholly a graph, the M1 UML models that systems engineers produce have diagrams that may contain partial graphs. An example is where the UML allows node elements to be connected together (eg, Class, Port) or placed within another element (Package, Package) without a visible relationship within the model itself (Figure 11).

This is visually compact but makes finding relationships difficult because it at least relies on exploring within the tool. For someone without the tool there is not the opportunity. The UML specification is several hundred pages long and aimed at the implementers of the UML not the users. It means that many users either do not realize that there is a hidden relationship between the elements, for example because the UML allows elements to be placed in a boundary rectangle which has the appearance of containment but has no relationship to the contained elements, or they have differing opinions on what the relationship might be. The users may therefore have different understandings of what the implicit relationship actually is. This does not help consistency in interpreting the semantics of the model.

In contrast keeping everything solely as a directed graph allows every tuple or assertion to be explicitly expressed and therefore queried. In this sense the presentation is kept consistent with the underlying model. This is a significant benefit for a repository. Exploitation of model tuples justifies the expense of creating the model. The model has therefore to support ad hoc queries of arbitrary path length.

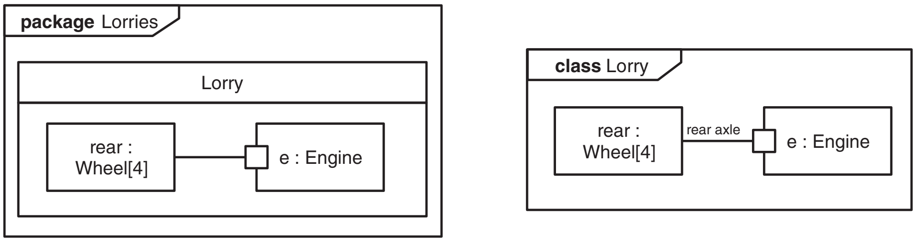


FIGURE 11 UML diagrams allow nongraph notation/implied relationships—UML class diagram and composite structure diagram

6.1 | Model element versions, lifecycle, and change

The Model Configuration and Change Viewpoint has been used to describe the changes to the TRAK Metamodel recorded in versions of the TRAK Metamodel specification. An example is a record of the addition of the "geographic extent" attribute to the Resource metamodel element (Figure 12). The CYPHER code to return Figure 12 is listed separately in Section C.2.

Since each tuple or assertion represents a path traversal it is possible using a graph query in CYPHER to display all of the changes made to the model element during its life. The query result (Figure 13) describes the addition of the Evidence element and its modification in subsequent versions of the metamodel in the sequence of the metamodel specification. The CYPHER code to return Figure 13 is listed separately in Section C.2.

Notice that the "submission date" property was added first but later deleted from the Evidence element hence there are both "ADDS … TO… " and "DELETES … FROM… " relationships with the Evidence element. In order to allow multiple additions and deletions of the same element the viewpoint metamodel includes a "change identifier" property for the ADDS … TO, DELETES … FROM and MODIFIES relationships to tie them to a particular Change identifier and provide the basis for a conditional query.

6.2 | Describing metamodel elements and properties

The Metamodel Element Structure Definition Viewpoint enables explicit declarations to be made in terms of inheritance of properties and participation in the same relationships as another element.

An example of this is the description of the TRAK System metamodel element. This is an Architecture Description Element as all elements that may appear in TRAK architecture views are. It is also a Resource as are Software, Physical, Organization, Role, and Job. These are the primary structural elements in TRAK architecture views describing solutions. In the TRAK metamodel all Resource elements and the Function element can be given a safety-significance, for example by setting a safety integrity level. There is therefore a Safety-Monitored Element from which such properties can be inherited. The Resource and Safety-Monitored Elements are both abstract—they cannot appear in an Architecture View and hence are not themselves Architecture Description Elements. Their purpose is simply to provide sources of common properties. In effect they are elements that are used to help manage the metamodel.

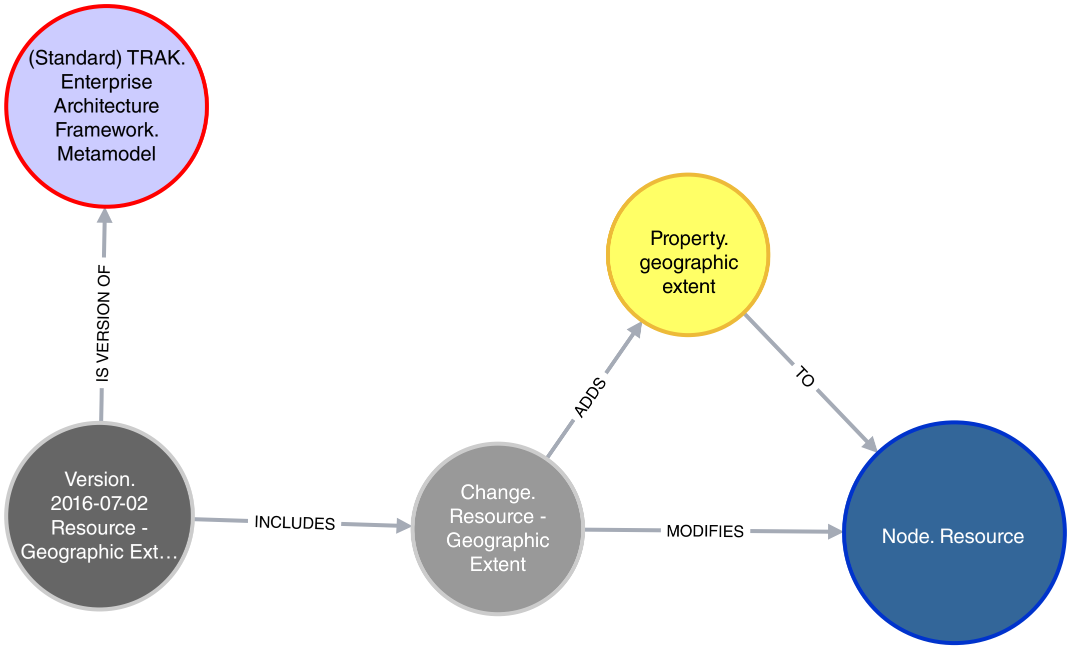

(Standard) TRAK. Enterprise Architecture Framework. Metamodel
IS VERSION OF
Property. geographic extent
ADDS
TO
Version. 2016-07-02 Resource - Geographic Ext…
INCLUDES
Change. Resource - Geographic Extent
MODIFIES
Node. Resource

FIGURE 12 Change made to TRAK resource metamodel element at specified version

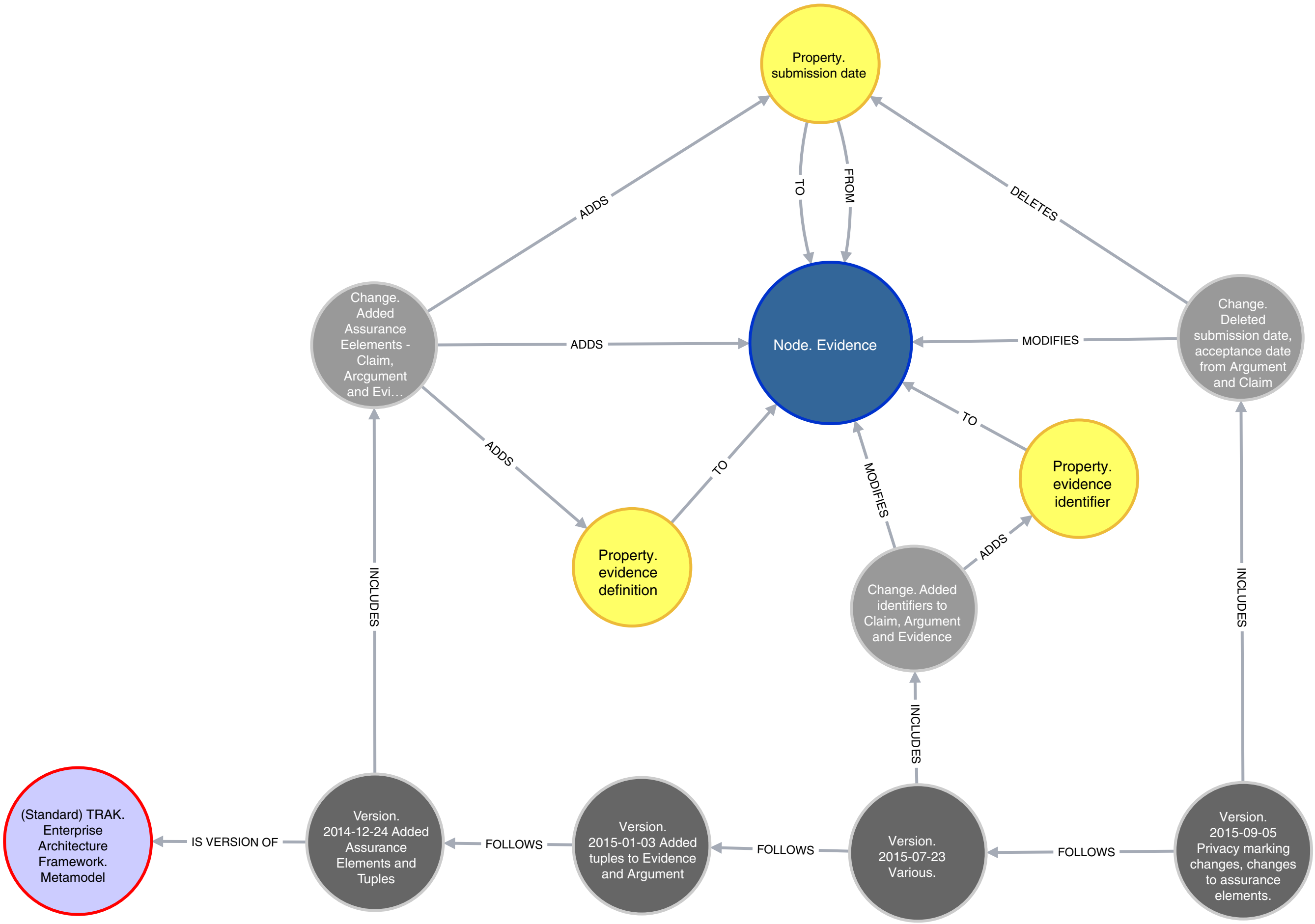

Property. submission date
TO
FROM
ADDS
DELETES
Change. Added Assurance Eelements - Claim, Arcgument and Evi…
ADDS
Node. Evidence
MODIFIES
Change. Deleted submission date, acceptance date from Argument and Claim
ADDS
TO
TO
Property. evidence identifier
MODIFIES
Property. evidence definition
ADDS
Change. Added identifiers to Claim, Argument and Evidence
INCLUDES
INCLUDES
INCLUDES
(Standard) TRAK. Enterprise Architecture Framework. Metamodel
IS VERSION OF
Version. 2014-12-24 Added Assurance Elements and Tuples
FOLLOWS
Version. 2015-01-03 Added tuples to Evidence and Argument
FOLLOWS
Version. 2015-07-23 Various.
FOLLOWS
Version. 2015-09-05 Privacy marking changes, changes to assurance elements.

FIGURE 13 Description of change lifecycle of the evidence TRAK metamodel element

The Metamodel Element Structure Viewpoint is able to represent these distinctions. Figure 14 is the result of the CYPHER query to display the description of the System metamodel element.

It is also possible to then produce metamodel element definitions in the tabular form which returns part of table 4.2 in the TRAK Metamodel specification (Table 3):

Using the tuples defined by the Metamodel Element Structure Definition Viewpoint it is possible to produce a characterization of the nodes, connectors, and tuples forming the metamodel (Table 4).

Table 4 shows that there are almost 50% more tuples within the TRAK metamodel than had been previously identified by manually unpacking the visual representation of the inheritance relationships within the TRAK metamodel. This is a direct benefit of a model.

6.3 | Describing and checking metamodel tuple definition

The Metamodel Tuple Definition Viewpoint can be used to describe single step (1-tuple) and multiple step (*n*-tuple) paths. *N*-Tuple paths are often needed to support consistency rules in an architecture description for example where there is a natural order that needs to be enforced to prevent the system engineer taking shortcuts. An example of this is not allowing a Claim to be asserted as proven before establishing the claim, supporting arguments, and supporting evidence.

In the TRAK metamodel a 2-tuple is needed to describe a Resource Interaction between two Resources (Job, Role, Organization, Physical, System, Software) where the Resource Interaction is represented as a node because a defined direction is also needed that is, from the source Resource element to the destination Resource element. In order to describe a

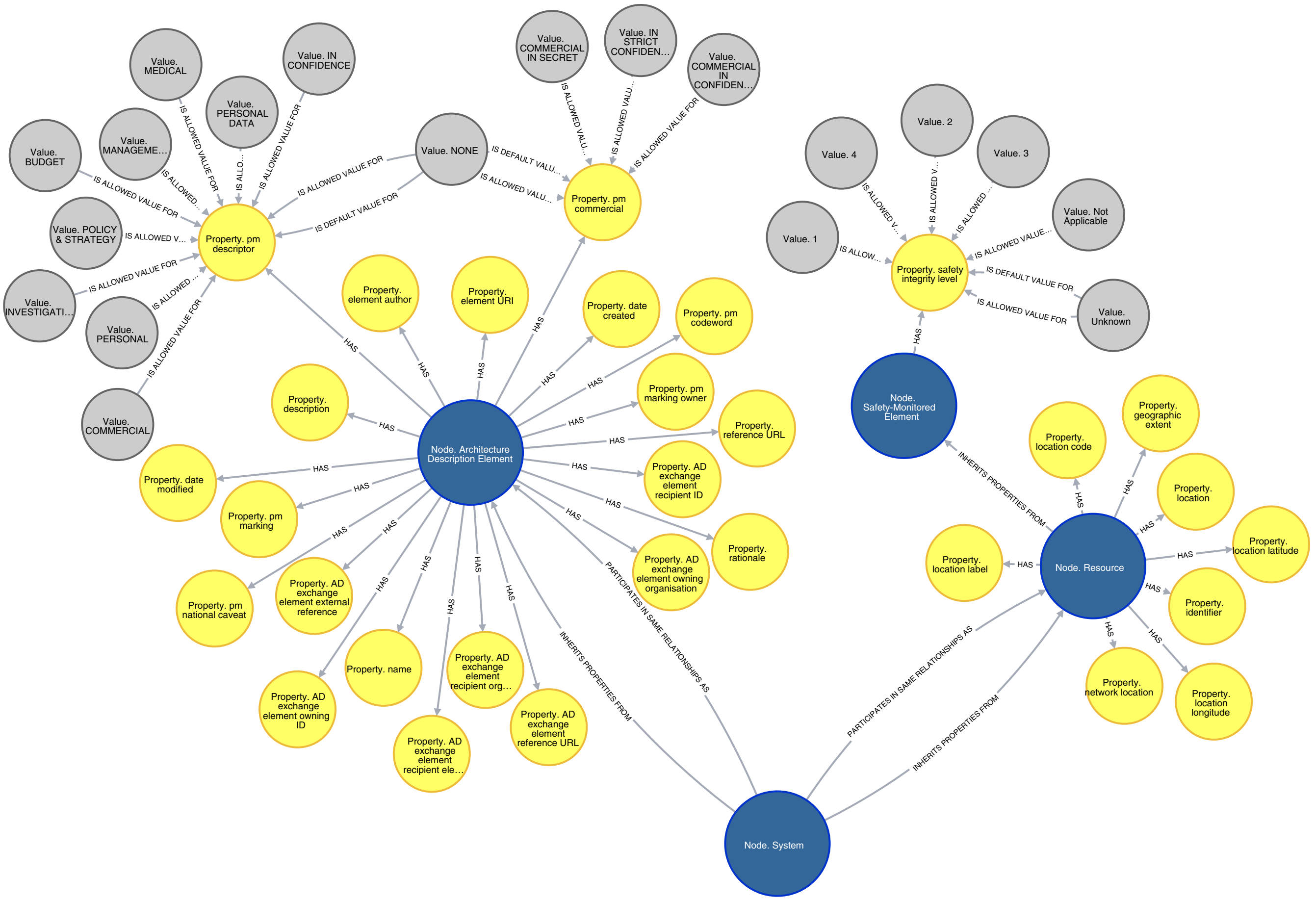


FIGURE 14 TRAK "System" metamodel element description

TABLE 3 Tabular description of role metamodel element

Date created	Date modified	Metamodel element (block)	Perspective	Definition	Inherits properties from	Tests for	Tests against	Comments
February 15, 2010	July 23, 2015	Role	Solution	The duties or status assumed or part played by a person or organization.	+Architecture Description Element +Human Resource +Resource +Safety-monitored element	many roles map to a single job		If a single role occupies most of a job-holder's time it may result in the job being redefined around the role.

TABLE 4 Overall TRAK metamodel statistics

Type	Count
No. metamodel architecture description tuples	748
No. metamodel node elements (architecture description element)	38
No. metamodel node elements (not architecture description element)	9
Metamodel node elements (total)	47
No. metamodel connector elements (total)	73
Properties (all)	242
Values	75

Resource Interaction from a Software element to a System element we need a sequence of the 1-tuples "Resource Interaction FROM Software" and "Resource Interaction TO System" (Figure 15). The CYPHER query to produce this is in Section C.4.

Using a directed graph query it is then possible to differentiate an 1-ary tuple from a *N*-ary tuple by testing for the STARTS WITH relationship that is, NOT "(adt) -[STARTS WITH]->()" or return the path length.

It is also possible to check for path closure in a model to validate the model structure. For example a TRAK architecture description used for design assurance or safety assurance would include a TRAK MV04 Assurance View. In this view there is a path "Evidence -SUPPORTS->Argument -SUPPORTS->Claim" which may later be closed using "Evidence -PROVES->Claim" which can be checked to ensure that the same Evidence and same Claim are involved.

It is possible to check for errors such as a Node or Connector not appearing in any metamodel tuple (Table 5).

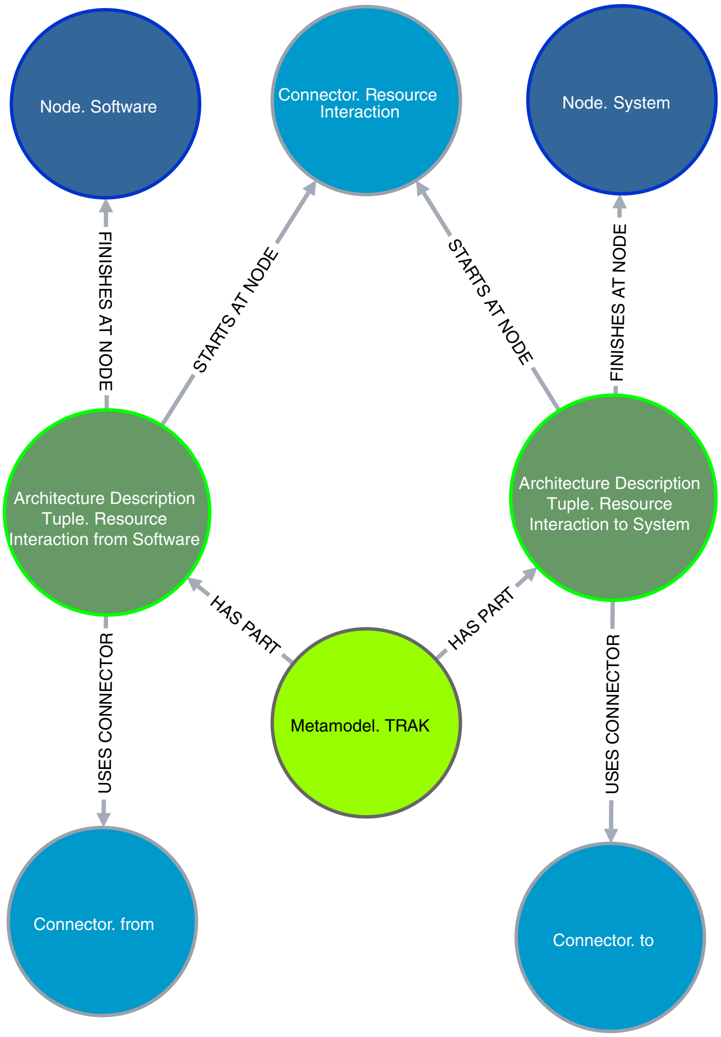


FIGURE 15 Description of 2-Tuple "Resource Interaction from Software to System"

TABLE 5 Quality check—metamodel connectors not used in metamodel tuple definition

Metamodel connector element not used in metamodel tuple definition
allows
groups related
presents

TABLE 6 Quality check—metamodel tuples not having a defined start node

Tuple	Warning
Architecture description issued by organization	No STARTS AT NODE relationship
Architecture view issued by organization	No STARTS AT NODE relationship
Architecture viewpoint issued by organization	No STARTS AT NODE relationship
Contract issued by organization	No STARTS AT NODE relationship
Evidence issued by organization	No STARTS AT NODE relationship

It is also possible to check for a malformed tuple, for example, one that is missing a definition of the start point (Table 6).

6.4 | Describing and checking architecture viewpoint definitions

The Viewpoint Definition Viewpoint provides the means to define the concerns addressed by the viewpoint and then metamodel tuples needed to address them. The construction of the tuples is provided by the Metamodel Tuple Definition Viewpoint. The "Architecture Description Tuple -ADDRESSES-> Concern" tuple provides the rationale for each metamodel tuple. This provides the basis for a quality check to identify Concerns for which there is no metamodel tuple or metamodel tuples for which there is no Concern.

Figure 16 describes the purpose of the TRAK MVp-04 Assurance Viewpoint. The CYPHER query used to produce this is in Section C.5.

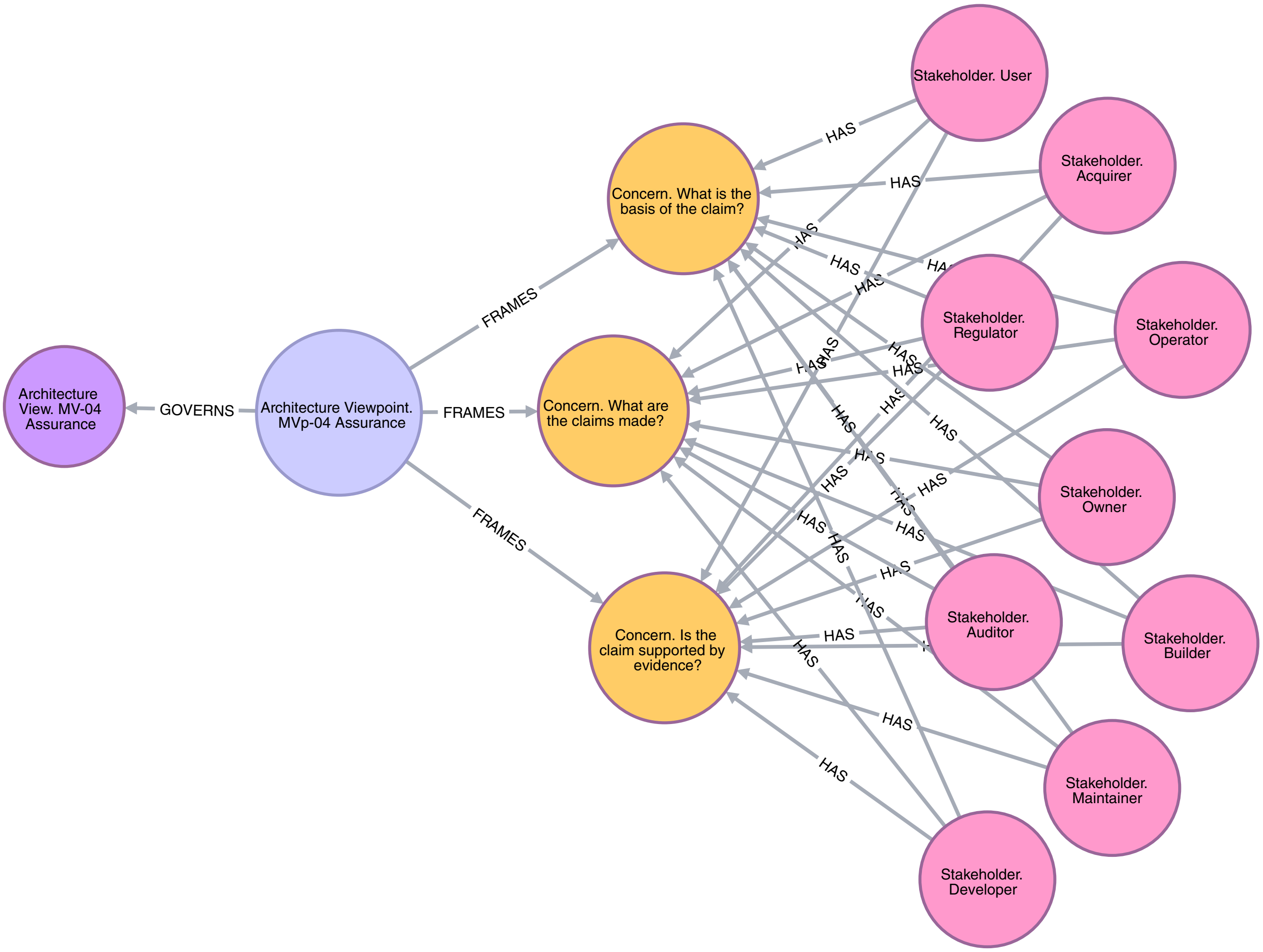


FIGURE 16 Description of viewpoint purpose and governance—MVp-04 assurance viewpoint

TABLE 7 Viewpoint concerns and tuples addressing them for MVp-04 assurance viewpoint

Viewpoint	Viewpoint concern	Addressed using
MVp-04 Assurance	Is the claim supported by evidence?	Evidence disproves Claim
		Evidence has part Evidence
		Evidence opposes Argument
		Evidence proves Claim
		Evidence supports Argument
MVp-04 Assurance	What are the claims made?	Claim about Architecture Task
		Claim about Argument
		Claim about Capability
		Claim about Claim
		Claim about Competence
		Claim about Concept Activity
		Claim about Concern

TABLE 8 Quality check—metamodel tuples missing from well-formedness definition

Tuples missing from well-formedness definition
Evidence has part Evidence
Claim has part Claim
Argument has part Argument
Argument opposes Argument

The tuples needed to address these Concerns are shown in Table 7.

The well-formedness rules can be checked against these tuples identified by verifying the completeness of the Architecture Viewpoint -FRAMES -> Concern <– ADDRESSES- Architecture Description Tuple <-REQUIRES AT LEAST—(same) Architecture Viewpoint path (Table 8).

6.5 | Describing the implementation of a metamodel using an ADL

The classic systems engineering process[34] starts with a solution agnostic specification against which a design or design specification is produced. This allows "the what" to be separated from "the how" as often there are many possible designs that may meet the solution agnostic specification. The same process can be applied to an architecture framework and its metamodel to separate the architecture framework from any one particular implementation or design using an ADL. Using an independent notation to specify from that used in the implementation avoids common mode failures in the notation that would otherwise manifest in both the specification and the design response in the ADL.

A metamodel may be represented by an ADL such as the UML, Systems modeling Language (SysML),[31] or ArchiMate.[35] The suitability of this representation depends on the concepts within the ADL and the tuples that exist in the ADL metamodel. It also depends on whether the concept is represented by a node or connector since a tuple is formed from two nodes and a connector (node—connector—node) or a single node with a relationship to itself (eg, Physical HAS PART Physical). The comparison of tuples can then be used to establish how much of each viewpoint the ADL is able to represent. The tuples that can be represented using the ADL affect whether particular viewpoint concerns can be addressed (using the "Architecture Description Tuple ADDRESSES Concern" established in the Viewpoint Definition Viewpoint). If particular concerns are essential to the architecture description process and the ADL is partially or completely unable to represent one or more tuples addressing these concerns a decision has to be made whether to choose an alternative ADL or use multiple ADLs. The viewpoint provides a basis for an objective assessment.

As an example Table 9 lists 10 random UML metamodel elements from the UML profile for TRAK and the TRAK metamodel elements represented by them. The CYPHER query used to return this is in Section C.6.

TABLE 9 Description of UML metamodel elements used to represent TRAK metamodel elements

Implementation	ADL element	ADL parent	ADL element type	TRAK element	TRAK element type
UML profile for TRAK	trakumlprofile:: Milestone	UML:: Class	Node, UML	Milestone	TRAK, Node
	trakumlprofile:: Metric	UML:: Class	Node, UML	Metric	TRAK, Node
	trakumlprofile:: Threat	UML:: Class	Node, UML	Threat	TRAK, Node
	trakumlprofile:: triggers	UML::Dependency	Connector, UML	triggers	TRAK, Connector
	trakumlprofile:: impacts on	UML:: Association	Connector, UML	impacts on	TRAK, Connector
	trakumlprofile:: caused by	UML:: Dependency	Connector, UML	caused by	TRAK, Connector
	trakumlprofile:: Risk	UML:: Event	Node,UML	Risk	TRAK, Node
	trakumlprofile:: Human Resource	UML:: ActivityPartition	Node,UML	Human Resource	TRAK, Node
	trakumlprofile:: addresses	UML:: Dependency	Connector,UML	addresses	TRAK, Connector
	trakumlprofile:: Job	UML:: Class	Node,UML	Job	TRAK, Node

Note that even if the ADL contains two suitable nodes and a connector it may still be unable to represent the required metamodel tuple if the ADL forbids the connection of the node elements using the connector. This is one of the reasons why there is an explicit "Implementation USES->Architecture Description Tuple -TO IMPLEMENT-> Architecture Description Tuple" path separate from the "Implementation USES-> Node -TO IMPLEMENT-> Node" and "Implementation USES-> Connector -TO IMPLEMENT-> Connector" paths. Any difference can then be made explicit.

This viewpoint has been used to directly check the output of a UML model and check the implementation against the description of the TRAK metamodel. The UML profile for TRAK is created using Sparx Enterprise Architect UML modeling tool and exported as an XML Metadata Interchange (XMI)[36] file. Using an eXtensible Stylesheet Language Transformation[37] sheet it is possible to convert this to a Comma-Separated Variable[38] file or connect directly to the Neo4J database by a Java Database Connectivity connector[39] which can then import and link elements in the one operation and check against the description of the architecture framework metamodel. Note that the forthcoming version 15 of Sparx Enterprise Architect incorporates an updated JavaScript engine[40] that supports JavaScript Object Notation (JSON)[41] which allows import of both nodes and connectors.

Table 10 lists metamodel elements not present in the UML profile.

TABLE 10 TRAK metamodel elements missing from the implementation in the UML profile for TRAK

Metamodel version 2018-01-31		
TRAK architecture description element	**Type(s)**	**Implementation**
from	TRAK, Connector	Missing from "UML profile for TRAK"
is attached to	TRAK, Connector	Missing from "UML profile for TRAK"
physically supports	TRAK, Connector	Missing from "UML profile for TRAK"
satisfies	TRAK, Connector	Missing from "UML profile for TRAK"
to	TRAK, Connector	Missing from "UML profile for TRAK"

7 | USE IN PRODUCING A METAMODEL OR ARCHITECTURE FRAMEWORK

Depending on what is needed there are different sequences in which the viewpoints can be used. To simply define a metamodel create a view against each of the viewpoints:

- Metamodel Element Structure Definition Viewpoint
- Metamodel Tuple Definition Viewpoint

If the metamodel is to be implemented using an ADL use the ADL Implementation Viewpoint to assess how many of the metamodel triples are able to be implemented.

If the aim is to define an architecture framework consisting of one or more viewpoints the order is slightly different because a concern-led approach is used to first identify the concerns addressed by each viewpoint before defining the metamodel that provides the tuples that address the concerns for each viewpoint.

- Viewpoint Definition Viewpoint -Viewpoint Purpose & Governance section. One for each viewpoint needed.
- Metamodel Element Structure Definition Viewpoint
- Metamodel Tuple Definition Viewpoint
- Viewpoint Definition Viewpoint—to tie one or more viewpoint tuples from the metamodel to each viewpoint concern using the Viewpoint Implementation section.
- For each ADL used—ADL Implementation Viewpoint—to assess how well the ADL is able to represent the triples required.

At suitable points the Model Configuration and Change Viewpoint can be used to define a baseline for the metamodel or architecture framework being defined.

8 | FURTHER WORK

Further work is ongoing to describe the remaining TRAK viewpoints. This may cause the metamodeling viewpoint definitions to be refined. Once all of the TRAK viewpoints have been described the model will be used to generate the mapping of the UML tuples to the TRAK metamodel tuples in each of the TRAK viewpoints.

One of the potential difficulties is interpreting the UML specification in order to identify the rules that apply to the allowed combinations of the UML node and connector elements to form the allowed UML tuples. It is not simply enough to look at the elements in isolation. Whilst a UML profile only contains the definitions of the node and connector elements in isolation the enforcement behavior is implemented in the UML modeling tool. The resulting behavior and ease of use or visibility affects the choice of element. This then creates the mappings between the UML metamodel tuples and the TRAK architecture description tuples in its metamodel. Having the mappings fully described will allow the impact of any trade-off in implementation to be made explicit to the users of TRAK.

Once the model is able to create the hand-crafted mappings in the current spreadsheet, the TRAK metamodel element definitions and the TRAK viewpoint definitions will be maintained using the model as a single source of truth.

Having a wholly graph-based model of the TRAK metamodel now provides the means to create a semantic description using RDF or OWL. It also provides the means to analyze combination paths which can be used to define consistency rules.

9 | CONCLUSIONS

Five viewpoint definitions have been created to support metamodel and viewpoint definition and the changes made over time. The viewpoint definitions have been designed to conform to ISO/IEC/IEEE 42010. Use of tuples is necessary to specify view content unambiguously. The Viewpoint Definition Viewpoint provides a mechanism to specify allowed and minimum view content and consistency rules.

The viewpoints presented have enabled the TRAK metamodel, viewpoint definitions, and the metamodel implementation in the UML to be described by graphs. This can be generalized to metamodels in general. The viewpoints can also be used to define a new architecture framework and show verify the means of addressing stakeholder concerns by linking them to metamodel triples.

The Model Configuration and Change Viewpoint enables changes made to a model element in response to a change request to be recorded within a model. In TRAK it is being used to maintain the record the versions of the TRAK metamodel specification but it can be applied to any model.

A single graph model is able to support metamodel definition, associated viewpoint definitions and implementations of the metamodel. This reduces inconsistency errors and as an integrated model enables impact of proposed changes to a viewpoint, metamodel element or triple or the implementation in a particular ADL to be assessed.

Views presented using directed graphs provide a method of describing architecture that is both understandable as a set of assertions by a nontechnical audience and is amenable to path-following query and analysis—it supports the needs of both the human and the machine. This is an important step towards providing a single architecture description that supports semantic analysis and transformation.

Using a graph database allows queries to be run to both generate and check definitions of the metamodel the associated viewpoint definitions and implementations of the metamodel. The visualization of results enables errors to be more easily spotted. Modeling TRAK showed that the manual unpacking of the metamodel triples had only identified 500 of the 750 triples. It also identified metamodel elements missing in the UML profile implementation by comparison of the Sparx Enterprise Architect XMI output with the model of the TRAK metamodel. Using CYPHER to create queries that follow paths through the model enables complex queries to be created to support error-checking. In particular the Awesome Procedures on CYPHER for Neo4J plug-in module[42] produces an output of the element labels (similar to types) based on the model content enabling a quick visual check of the expected content and relationships to be made.

ACKNOWLEDGEMENTS

I would like to thank Stephen Moran for the conversation at National Grid that introduced me to graph databases and Neo4J.

PEER REVIEW INFORMATION

Engineering Reports thanks the anonymous reviewers for their contribution to the peer review of this work.

CONFLICT OF INTEREST

The authors declare no potential conflict of interest.

ORCID

Nic Plum https://orcid.org/0000-0003-3058-5022

REFERENCES

1. Plum N. TRAK00004 TRAK Enterprise architecture framework. TRAK00004; 2016. https://sourceforge.net/projects/trak/.
2. ISO/IEC/IEEE 42010: 2011: Systems and software engineering, architecture description. International Standards Organisation; 2011.
3. Plum N. TRAK00002 TRAK Enterprise architecture framework. Metamodel. TRAK00002; 2017. https://sourceforge.net/projects/trakmetamodel/.
4. Plum N. TRAK00001 TRAK Enterprise architecture framework. Viewpoints. TRAK00001; 2018. https://sourceforge.net/projects/trakviewpoints.
5. Plum N. TRAK00005 TRAK Implementation. Architecture description elements. TRAK00005; 2017.
6. TRAK UML Profile. *SourceForge*. https://sourceforge.net/projects/trakumlprofile/. Accessed August 16, 2019.
7. Sparx EA - MDG for TRAK. *SourceForge*. https://sourceforge.net/projects/mdgfortrak/. Accessed August 16, 2019.
8. Sparx systems enterprise architect UML modelling tool. https://sparxsystems.com/products/ea/index.html. Accessed August 09, 2019.
9. Plum N. TRAK00007 TRAK implementation suitability assessment UML. TRAK00007; 2016. https://sourceforge.net/projects/trak/files/Suitability_of_Architecture_Description_Languages/UML/.
10. NATO architecture framework Version 4. Architecture Capability Team; 2018.
11. Plum N. Architecture description viewpoints: metamodel description, implementation and model changes. Eclectica Systems Ltd, 3736126–001; 2019. https://doi.org/10.13140/RG.2.2.14037.99049.
12. Guychard C, Guerin S, Koudri A, Beugnard A, Dagnat F. Conceptual interoperability through models federation; 2013.
13. Fischer K, Panfilenko D, Krumeich J, Born M. Viewpoint-based modeling—towards defining the viewpoint concept and implications for supporting modeling tools; 2012.
14. COMPASS D21.5b: definition of the COMPASS architectural Ffamework. COMPASS Project Consortium, D21.5b, September, 2014. http://www.compass-research.eu/Project/Deliverables/D21.5b.pdf.

15. Baroni A, Muccini H, Malavolta I, Woods E. Architecture description leveraging model driven engineering and semantic Wikis. Paper presented at: Proceedings of the 2014 IEEE/IFIP Conference on Software Architecture; 2014:251-254.
16. Malavolta I, Lago P, Muccini H, Pelliccione P, Tang A. What industry needs from architectural languages: a survey. *IEEE Trans Softw Eng*. 2013;39(6):869-891. https://doi.org/10.1109/TSE.2012.74.
17. Wenzel S, Kelter U. Model-driven design pattern detection using difference calculation. Paper presented at: Proceedings of the 1st International Workshop on Pattern Detection for Reverse Engineering (DPD4RE), co-located with 13th Working Conference on Reverse Engineering (WCRE'06); 2006; Benevento, Italy.
18. Dijkstra E. On the role of scientific thought. *Selected Writings on Computing: A Personal Perspective*. New York, NY: Springer-Verlag; 1982:60-66.
19. MODAF M3 Version 1.2.004; 2013. https://www.gov.uk/guidance/mod-architecture-framework.
20. C182 the TOGAF® Standard Version 9.2. The Open Group. https://publications.opengroup.org/standards/togaf/c182.
21. Plum N. TRAK00015 TRAK: architecture description summary conformance assessment - ISO/IEC/IEEE 42010:2011. TRAK00015; 2015.
22. ISO/IEC/IEEE 15288:2015. ISO. http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/37/63711.html. Accessed March 15, 2019.
23. Graph (Discrete Mathematics). *Wikipedia*. Wikimedia Foundation. https://en.wikipedia.org/wiki/Graph_(discrete_mathematics). Accessed December 15, 2017.
24. Tuple. *Wikipedia*; January 02, 2020. https://en.wikipedia.org/w/index.php?title=Tuple&oldid=933707383. Accessed January 04, 2020.
25. RDF 1.1 concepts and abstract syntax. World Wide Web Consortium (W3C). http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.
26. Berners-Lee T. Semantic web roadmap. October 14, 1998. https://www.w3.org/DesignIssues/Semantic.html. Accessed January 05, 2020.
27. Bézivin J. On the unification power of models. *Softw Syst Model*. 2005;4(2):171-188. https://doi.org/10.1007/s10270-005-0079-0.
28. IEC 61025:2006 fault tree analysis. IEC 61025:2006; 2006. https://webstore.iec.ch/publication/4311.
29. IEC 61078:2016 reliability block diagrams. IEC 61078:2016; 2016. https://webstore.iec.ch/publication/25647.
30. OMG Unified Modeling Language (OMG UML™) Version 2.5.1. The Object Management Group, Formal/17–12-05; 2017. http://www.omg.org/cgi-bin/doc?formal/10-05-06.pdf.
31. OMG Systems Modeling Language (OMG SysML™) Version 1.4. The Object Management Group, Formal/June 3, 2015; 2015. http://www.omg.org/spec/SysML/1.4/PDF.
32. Neo4j Graph Platform. Neo4j Graph Database Platform. https://neo4j.com/. Accessed August 09, 2019.
33. Cypher Graph Query Language. eo4j Graph Database Platform. https://neo4j.com/cypher-graph-query-language/. Accessed August 09, 2019.
34. INCOSE Systems Engineering Handbook. 4th. INCOSE-TP-2003-002-04; 2015
35. ArchiMate ® 3.0.1 Specification. The Open Group, C179. August; 2017.
36. XML Metadata Interchange (XMI) Specification. Version 2.5.1. The Object Management Group, Formal/June 7, 2015, June 2005; https://www.omg.org/spec/XMI/2.5.1/PDF.
37. XSL Transformations (XSLT) Version 3.0. https://www.w3.org/TR/xslt-30/. Accessed August 09, 2019.
38. Shafranovich Y. RFC 4180 Common Format and MIME Type for Comma-Separated Values (CSV) Files. IETF RFC 4180 October; 2005. https://tools.ietf.org/html/rfc4180.
39. Neo4j JDBC Driver Documentation. https://neo4j-contrib.github.io/neo4j-jdbc/. Accessed August 10, 2019.
40. History - Enterprise Architect 15.0 | Sparx Systems. https://sparxsystems.com/products/ea/history.html. Accessed August 09, 2019.
41. Bray T. RFC 7159 the JavaScript object notation (JSON) data interchange format. *IETF RFC*. 2014;7159:1-15. https://tools.ietf.org/html/rfc7159.
42. Awesome procedures on cypher for Neo4j. Neo4j Contrib; 2019.
43. Chapter 3: Cypher the Neo4j developer manual v3.2. https://neo4j.com/docs/developer-manual/3.2/cypher/. Accessed August 16, 2019.

How to cite this article: Plum N. Using directed graphs to define viewpoints to keep a metamodel, an architecture framework and views using different modeling languages consistent. *Engineering Reports*. 2020;2:e12168. https://doi.org/10.1002/eng2.12168

APPENDIX A. COMPLETE METAMODEL

For completeness it is always necessary to be able to present a metamodel as a whole. This allows checks of paths through the model. It also enables coverage checks to be made of the tuples allowed/required in any viewpoint definition against the whole metamodel to assess the degree of overlap and check for any coverage gaps. This helps mitigate against errors where there is too much overlap between viewpoints indicating that the concerns are not sufficiently distinct or parts of the metamodel that appear to serve no purpose because they do not appear in any viewpoint definition or the balance between the coverage by the set of minimum well-formedness tuples vs the allowed tuples.

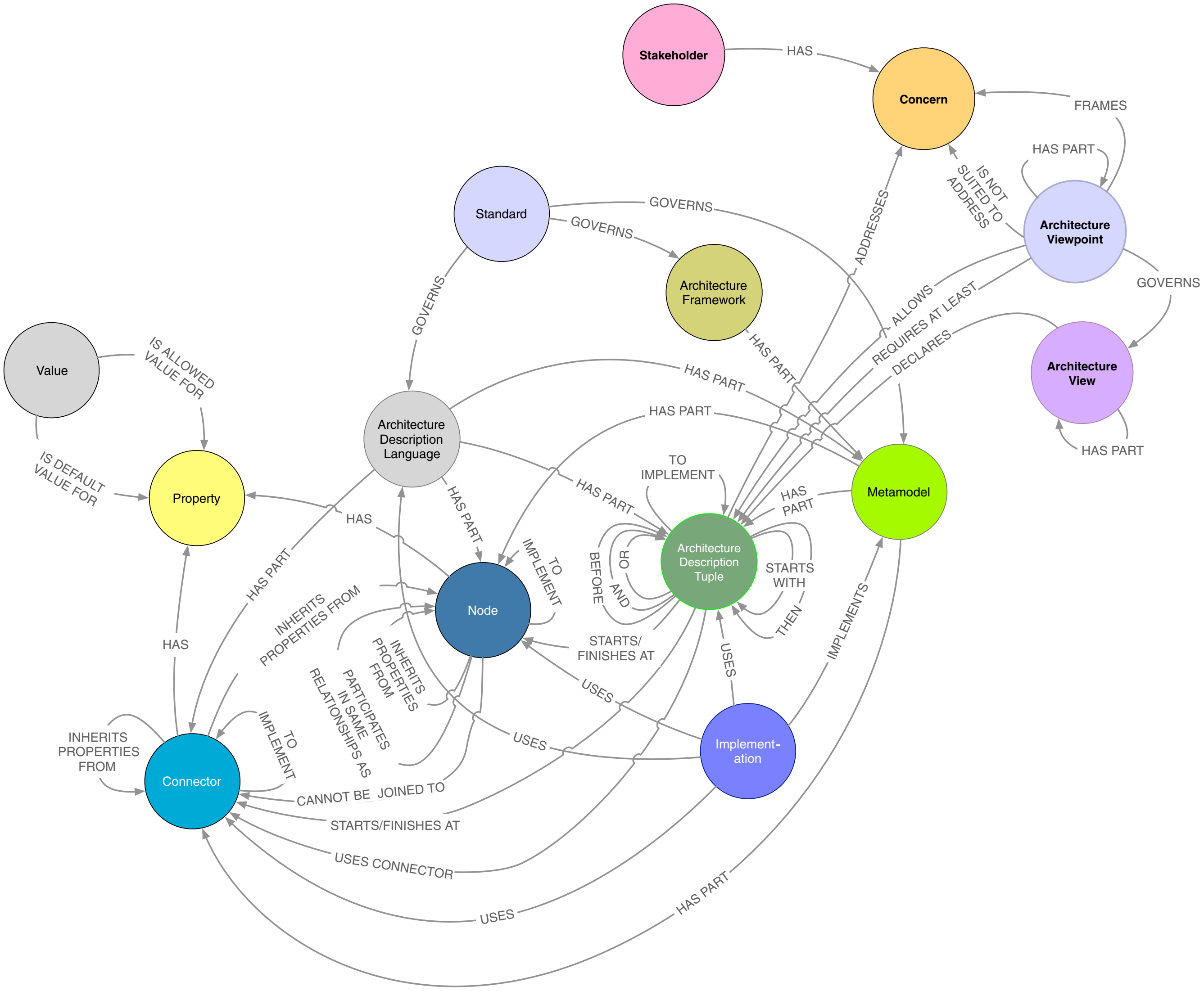


FIGURE A1 Complete metamodel on which viewpoint definitions based

Figure A1 presents the nodes and relationships forming the complete metamodel behind the viewpoint definitions (and therefore the basis for the CYPHER queries). The metamodel is fully defined including the element properties in Reference 11.

Note that the metamodel for the Model Configuration and Change Viewpoint in Figure 5 can be connected to any of these elements to describe changes to the metamodel over time.

APPENDIX B. VIEWPOINT DEFINITION

B.1 Viewpoint definition viewpoint

The Viewpoint Definition Viewpoint in Reference 11 is defined as follows. This provides an example of how a viewpoint can be defined using triples and form a specification against which views are produced. It follows the structure outlined in Section 4.

An example of an actual viewpoint from TRAK, the MVp-04 Assurance Viewpoint, follows this at B.2 to show the content of a viewpoint as it appears in the TRAK Viewpoint specification.[4]

VIEWPOINT DEFINITION VIEWPOINT.
VERSION NUMBER.
1

DATE CREATED.
DATE MODIFIED.
10. Nov. 2019
DESCRIPTION.

Describes the definition of an Architecture Viewpoint. This viewpoint provides the basis of defining what tuples are needed in the metamodel in order to address the specific concerns framed by each viewpoints.

CONCERNS ADDRESSED

- What Concerns does the viewpoint frame?
- Which Stakeholders hold these Concerns?
- What metamodel tuples are needed to address these Concerns?
- What is the acceptable view content?

ANTICONCERNS.
-
SUBJECT TUPLES.
Figure B1 shows the tuples used to define the viewpoint content.

[Viewpoint Purpose & Governance]

- Stakeholder—HAS → Concern
- Architecture Viewpoint—ADDRESSES → Concern
- Architecture Viewpoint—IS NOT SUITED TO ADDRESS → Concern

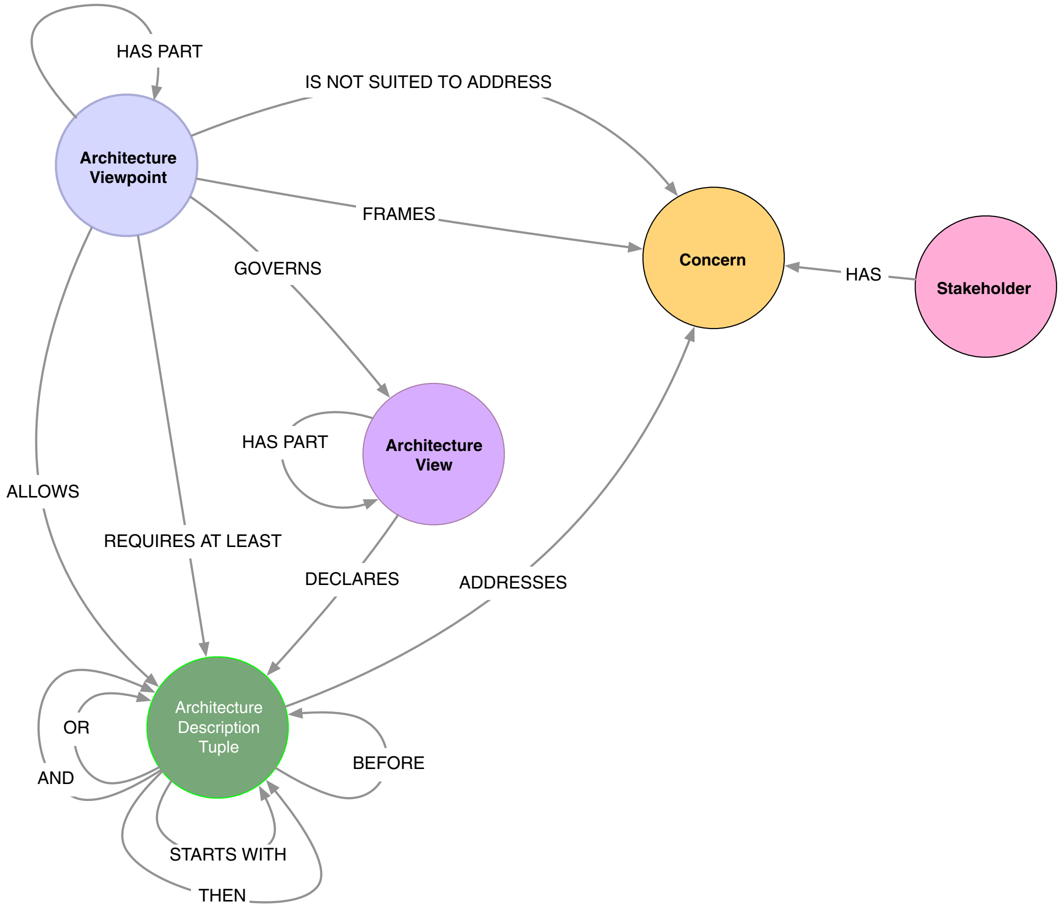


FIGURE B1 Viewpoint definition viewpoint tuples

- Architecture Viewpoint—GOVERNS → Architecture View
- Architecture Viewpoint—HAS PART → Architecture Viewpoint
- Architecture View—HAS PART → Architecture View

[Viewpoint Implementation]

- Architecture Description Tuple—ADDRESSES → Concern
- Architecture Viewpoint—REQUIRES AT LEAST → Architecture Description Tuple
- Architecture Description Tuple—ALLOWS → Architecture Description Tuple
- Architecture Description Tuple—BEFORE → Architecture Description Tuple
- Architecture Description Tuple—OR → Architecture Description Tuple

where a path/multiple tuple can be represented:

[2..N-Tuple]

- Architecture Description Tuple—STARTS WITH → Architecture Description Tuple
- Architecture Description Tuple—THEN → Architecture Description Tuple

OPTIONAL TUPLES.
-
WELL-FORMEDNESS.
There is at least one Architecture Viewpoint—the subject of the viewpoint definition.

[Viewpoint Purpose and Governance].
At least one Architecture Viewpoint governs one Architecture View:

- Architecture Viewpoint—*GOVERNS* → Architecture View.

Each Architecture Viewpoint frames at least one Concern:

- Architecture Viewpoint—*FRAMES* → Concern.

Each Concern has at least one identified Stakeholder using:

- Stakeholder—*HAS* → Concern.

The set of Concerns addressed by each Architecture Viewpoint is distinct from those addressed by any other Architecture Viewpoint—otherwise it is identical in purpose and what is needed to address the concerns.

[Viewpoint Implementation].
There is at least one Architecture Viewpoint—the subject of the viewpoint definition.
Each Architecture Viewpoint requires at least one Architecture Description Tuple using:

- Architecture Viewpoint—*REQUIRES AT LEAST* → Architecture Description Tuple.

Each of these Architecture Description Tuples required addresses at least one Concern using:

- Architecture Description Tuple—*ADDRESSES* → Concern.

[Well-formedness].
There is at least one Architecture Viewpoint—the subject of the viewpoint definition.
Each Architecture Viewpoint is attached to at least one Architecture Description Tuple using:

- Architecture Viewpoint—*REQUIRES AT LEAST* → Architecture Description Tuple.

Each Architecture Viewpoint is attached to zero or more Architecture Description Tuple using:

- Architecture Viewpoint—*ALLOWS* → Architecture Description Tuple.

Each Architecture Description Tuple is attached to zero or more Architecture Description Tuple using:
[
Architecture Description Tuple—*BEFORE* → Architecture Description Tuple.
OR
Architecture Description Tuple—*OR* → Architecture Description Tuple.
OR
Architecture Description Tuple—*AND* → Architecture Description Tuple.
OR
{Architecture Description Tuple—*STARTS WITH* → Architecture Description Tuple **AND**

Architecture Description Tuple—*THEN* → Architecture Description Tuple**}**
]

PRESENTATION METHODS.

Figure B2 provides an example of the Viewpoint Purpose and Governance form. It shows the concerns framed for the TRAK MVp-04 Assurance Viewpoint together with the corresponding MV-04 view.

Figure B3 provides an example of the Viewpoint Implementation form. It shows the tuples required to address a specified concern. The tuples form the well-formedness criteria for the TRAK MVp-04 viewpoint.

Graph.

```
 //
 //Describe Viewpoint Purpose & Governance - Specified Viewpoint
 //
MATCH viewpoint_purpose_governance=(view:Architecture_View)<-[:GOVERNS]-(viewpoint:Architecture_
Viewpoint {name:'MVp-04 Assurance'})-[:FRAMES]->(concern:Concern)
 RETURN viewpoint_purpose_governance
```

CONSISTENCY RULES.

Viewpoint Implementation matches Viewpoint Purpose:

- **IF**

  Architecture Description Tuple—ADDRESSES → Concern
- **THEN**

  (same) Concern ←*FRAMES*— Architecture Viewpoint —*GOVERNS*→ Architecture View —*REQUIRES AT LEAST*→ (same) Architecture Description Tuple.

for example, implementing this quality check using the CYPHER query language:

```
 //
 //
 // Quality - Identify tuples required to address viewpoint concerns missing from well-formedness criteria
 //
MATCH (vp:Architecture_Viewpoint:TRAK {name:'MVp-04 Assurance'})-[:FRAMES]->(conc:Concern)<-
[:ADDRESSES]-(adt:Architecture_Description_Tuple)
 WHERE NOT (vp)-[:'REQUIRES AT LEAST']->(adt)
 RETURN adt.name AS 'Tuples Missing from Well-Formedness Definition'
```

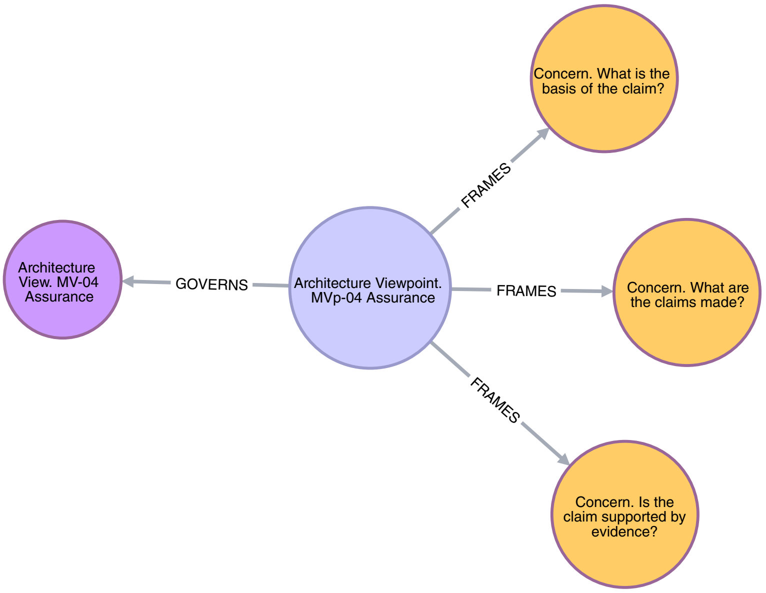

Concern. What is the basis of the claim?
FRAMES
Architecture View. MV-04 Assurance
GOVERNS
Architecture Viewpoint. MVp-04 Assurance
FRAMES
Concern. What are the claims made?
FRAMES
Concern. Is the claim supported by evidence?

FIGURE B2 Example—viewpoint purpose and governance (TRAK MVp-04 assurance viewpoint)

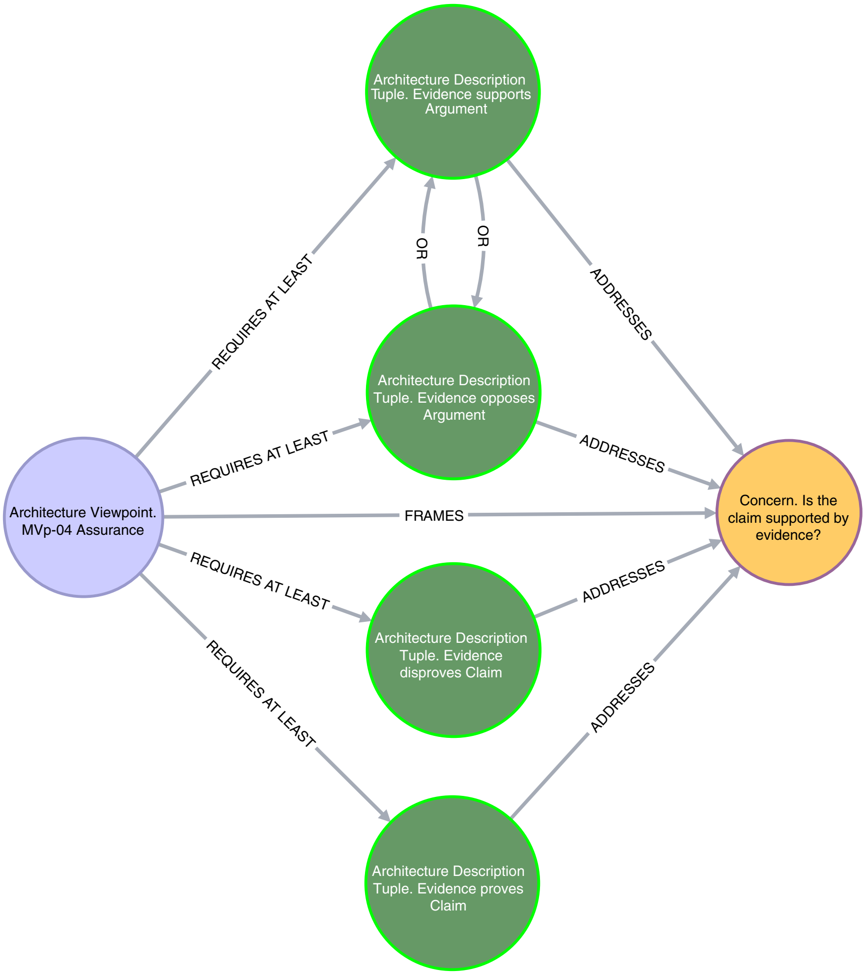

Architecture Description Tuple. Evidence supports Argument
Architecture Description Tuple. Evidence opposes Argument
Architecture Viewpoint. MVp-04 Assurance
Concern. Is the claim supported by evidence?
Architecture Description Tuple. Evidence disproves Claim
Architecture Description Tuple. Evidence proves Claim
REQUIRES AT LEAST
REQUIRES AT LEAST
REQUIRES AT LEAST
REQUIRES AT LEAST
OR
OR
ADDRESSES
ADDRESSES
ADDRESSES
ADDRESSES
FRAMES

FIGURE B3 Example—viewpoint implementation form showing well-formedness criteria for a specified concern (TRAK MVp-04 assurance viewpoint)

COMMENTS

- Any Architecture Description Tuple that is permitted is identified using "Architecture Viewpoint —*ALLOWS*→ Architecture Description Tuple."
- The required minimum content is identified using "Architecture Viewpoint —*REQUIRES AT LEAST*→ Architecture Description Tuple" and then "Architecture Description Tuple—*AND*/*OR*/*STARTS WITH*→ + —*THEN*→".
- AntiConcerns are defined using "Architecture Viewpoint —IS NOT SUITABLE FOR→ Concern"-AntiConcerns can be used to help architects choose appropriate viewpoint by directing them to another one.
- "Architecture Description Tuple —*BEFORE*→ Architecture Description Tuple" can be used to define a required architecture description sequence for consistency, for example:
  - assurance: establish Claim—Argument—Evidence chain *BEFORE* asserting Evidence—*PROVES/DISPROVES*→ Claim
  - interfaces: identify interface *BEFORE* characterizing interface

B.2 Example TRAK viewpoint—TRAK MVP-04 assurance viewpoint

The following is an example of the definition of a viewpoint taken from the TRAK architecture framework viewpoints specification.[4]

MVP-04 ASSURANCE.

VERSION NUMBER.
3
DATE.
December 8, 2017.
DESCRIPTION.

Describes a claim made about any other element with supporting (or opposing) arguments and evidence to establish how and whether a claim is proved or disproved (as a result of the assessed evidence).

Typical claims for solutions include that a system is safe, fit for purpose and meets its requirements.

CONCERNS ADDRESSED (Table B1).

Note: stakeholders/roles may be associated with the Enterprise, Concept, Solution, Project, Architecture Task for example, Builder of Enterprise, Auditor of Solution, Owner of Concept.

ANTI-CONCERNS.

-

DECLARED TUPLES.

Identification of Claim/Forming Argument

- Claim *about* Architecture Description Element
- Claim *has part* Claim
- Claim *supports* Claim
- Claim *opposes* Claim (a counter-claim)
- Argument *supports* Claim
- Argument *opposes* Claim
- Argument *opposes* Argument (a counter-argument)
- Argument *has part* Argument
- Architecture Description Element *traces to* Argument [where architecture of system of interest forms basis of Argument]
- Organisation *makes* Claim
- Role *makes* Claim

Verification of Claim/Argument.

As identification +

- Evidence *proves* Claim
- Evidence *disproves* Claim
- Evidence *supports* Argument
- Evidence *opposes* Argument
- Evidence *has part* Evidence

OPTIONAL TUPLES.

Context—Roles

- Job *plays* Role
- Organisation *plays* Role
- where a typical assurance-related role might be “Assessor,” “Auditor,” “Design Authority,” “Regulator” in conjunction with
- Role *extends to* Resource (the object of the claim)

TABLE B1 MVp-04 stakeholder concerns

Stakeholder	Concern of Stakeholder
Acquirer, Auditor, Builder, Developer, Maintainer, Operator, Owner, User	What are the claims made? What is the basis of the claim? Is the claim supported by evidence?

Note: stakeholders/roles may be associated with the Enterprise, Concept, Solution, Project, Architecture Task for example, Builder of Enterprise, Auditor of Solution, Owner of Concept.

Universal

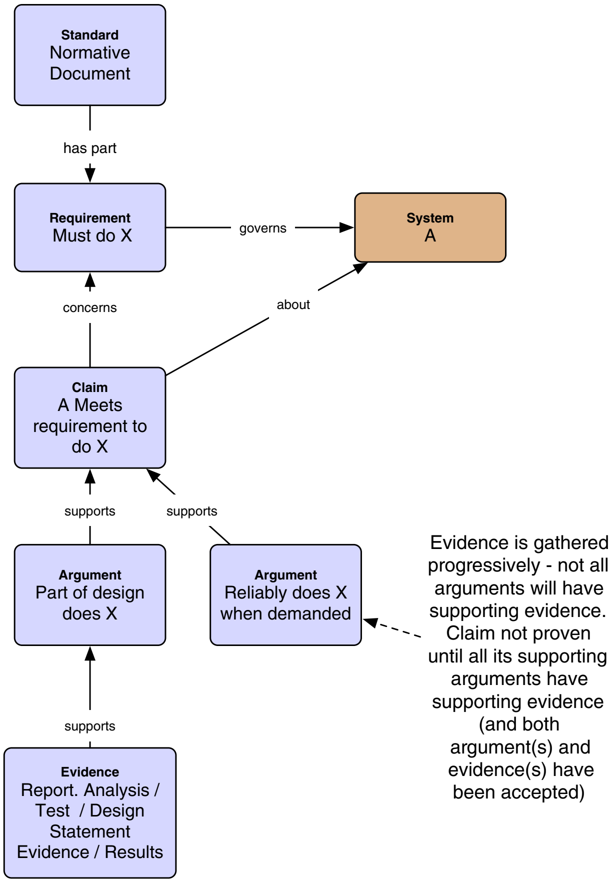
- Claim *about*, Concern *about*, *traces to* Argument, *traces to* Document, Requirement *governs*, *satisfies* Requirement, Standard *governs*, *satisfies* Standard, Contract *governs*, *satisfies* Contract, *traces to* Contract, *traces to* Requirement, *traces to* Standard may be added to any Architecture Description Element

If any of these optional metamodel elements are added then the appropriate TRAK Master Architecture View must be provided.

WELL-FORMEDNESS.
A MV-04 shall contain:

- at least one Claim (the subject of the view)
- every Claim is connected to the object of the claim (not itself) (Claim *about*...) or another Claim (Claim *has part* Claim)
- at least one Argument is connected to at least one Claim (Argument *supports*/*opposes* Claim)
- every Argument is connected to a Claim (Claim *supports*/*opposes* Argument) or another Argument (Argument *has part* Argument)
- at least one Evidence is connected to at least one Argument (Evidence *supports*/*opposes* Argument)
- and only then that same Evidence may be connected to a Claim (Evidence *proves*/*disproves* Claim)
- every Evidence is directly connected to an Argument (Evidence *supports*/*opposes* Argument) or indirectly connected to an Argument as part of another Evidence (Evidence *has part* Evidence)

PRESENTATION METHODS.
Figure B4 provides an example describing the design verification of a requirement—the claim is that the design meets the requirement.



Evidence is gathered progressively - not all arguments will have supporting evidence. Claim not proven until all its supporting arguments have supporting evidence (and both argument(s) and evidence(s) have been accepted)

FIGURE B4 MVp-04 Example 1—design verification

The example in Figure B5 shows part of a description from a System Design Authority of a claim about their overall system. This results in a requirement being placed onto the supplier of System C who therefore has to provide a description of their System C meets two requirements placed on the supplied via a System Requirement Document.

Figure B6 provides an example of a description of the supplier's response to the System C System Design Requirement Document placed upon them by the System Design Authority for the containing system .

VIEWS NEEDED IN ORDER TO CONSTRUCT.

Since the object of the Claim (Claim about [any TRAK metamodel element]) has to be shown the Master Architecture View for that object will be needed before the MV-04 can be drawn.

The Master Architecture View for any TRAK metamodel element is defined in Table 3.

CONSISTENCY RULES

- Order of Closing the evidential loop: The assertion that some evidence proves or disproves a claim can only be made after a supporting or opposing argument has been made in respect of this evidence (for the claim):
  - {Evidence *supports/opposes* Argument *supports/opposes* Argument Claim} **BEFORE** {(same) Evidence *proves/disproves* (same) Claim}
- the argument and evidence assertions must be consistent with the assertion that the claim is proved or disproved. For "*proves*" this requires a wholly supporting chain from evidence through the argument(s) to the supported claim(s).

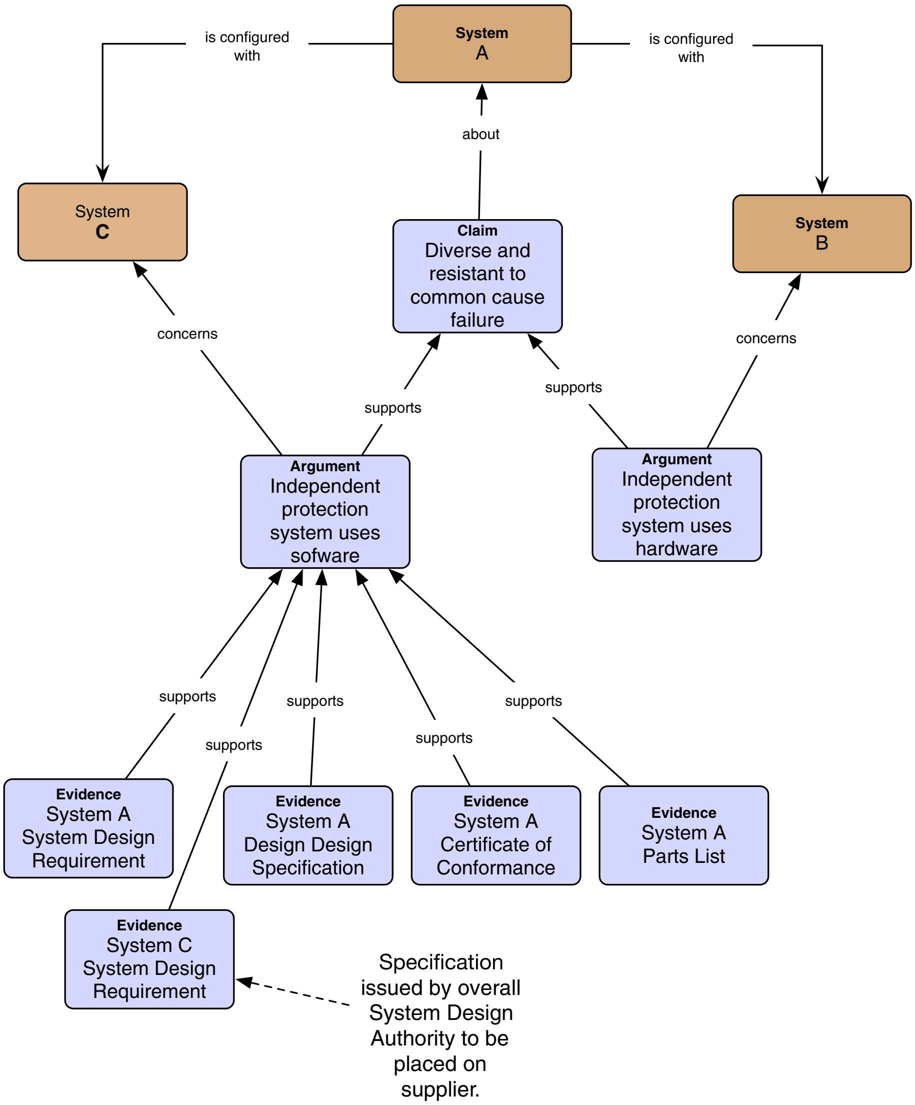

is configured with
System A
is configured with
about
System C
Claim Diverse and resistant to common cause failure
System B
concerns
supports
supports
concerns
Argument Independent protection system uses sofware
Argument Independent protection system uses hardware
supports
supports
supports
supports
supports
Evidence System A System Design Requirement
Evidence System A Design Design Specification
Evidence System A Certificate of Conformance
Evidence System A Parts List
Evidence System C System Design Requirement
Specification issued by overall System Design Authority to be placed on supplier.

FIGURE B5 MVp-04 Example 2—system design authority claim

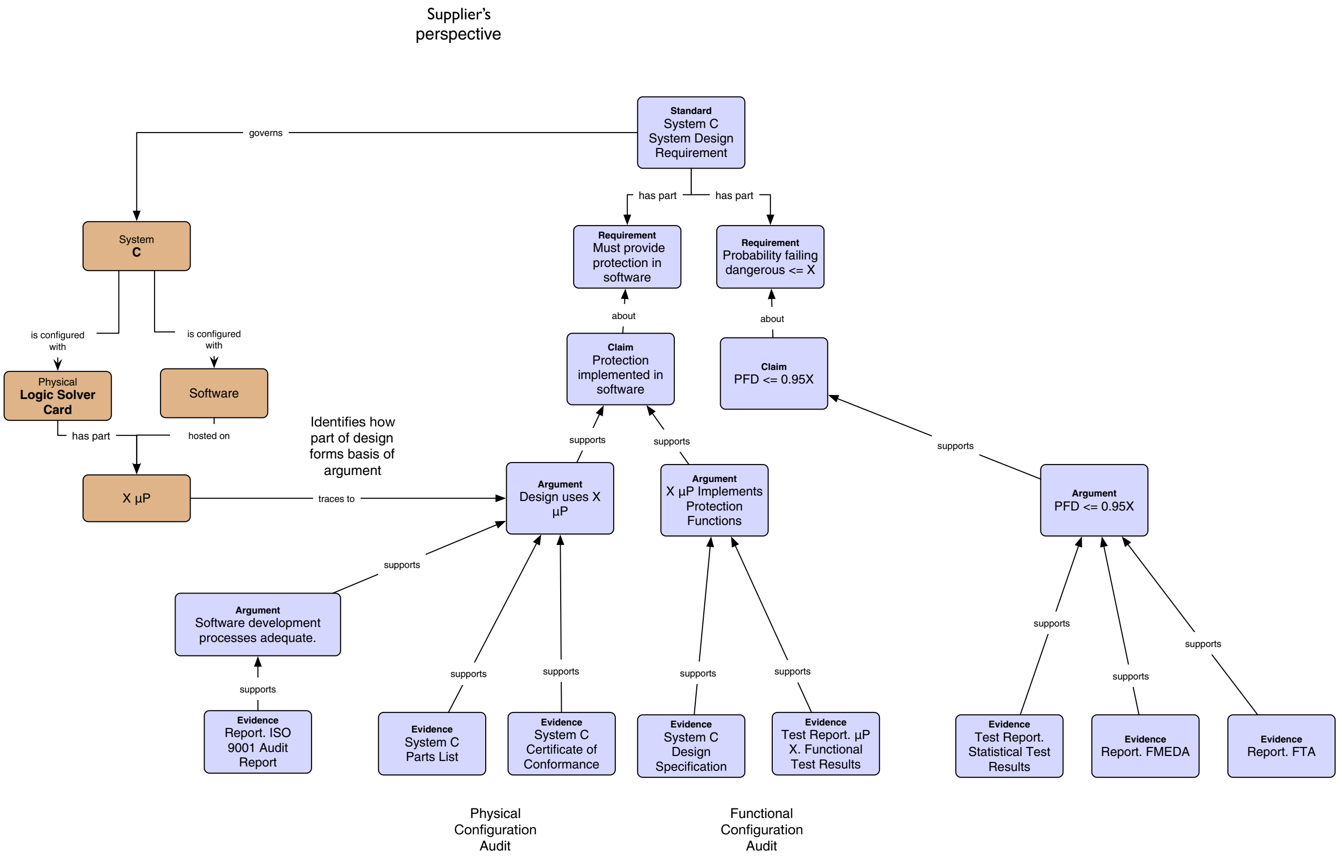


FIGURE B6 MVp-04 Example 3—supplier claim

For "*disproves*" this requires at least one "*opposes*" relationship (either or both of the evidence does not support the argument or the argument does not support the claim):

IF {Evidence supports Claim} **THEN** {(same) Evidence supports Argument supports (same) Claim}.

OR

IF {Evidence opposes Claim} **THEN EITHER** {(same) Evidence opposes Argument supports (same) Claim} **OR** {(same) Evidence supports Argument opposes (same) Claim}.

COMMENTS.

The MV-04 is the master architecture view for Claim, Argument and Evidence.

The supporting (opposing) parts of an Argument or Evidence by inference also support the Claim or Argument respectively to which the top-most "whole" Argument or Evidence is connected. The part Arguments or part Evidences may also support other Argument or Evidence elements.

A counter-claim is established using "Claim *opposes* Claim."

A counter-argument is established using "Argument *opposes* Argument" (and is usually followed by (same) Argument *opposes* Claim).

When the "acceptance date" attribute of a Claim, Argument or Evidence element is non-null and valid (not in the future) that element is deemed to have been accepted by the assessor of the claim.

Claims can be made against any element in any TRAK perspective. Claims can be made against a concept, the enterprise and its capabilities and goals, against a system and its ability to realize these capabilities. Claims can also be made against a project, its structure or activities (and thence against the introduction or removal from service of a system). Claims can also be made against standards or against a contract and its requirements.

Applied to the solution perspective this viewpoint supports the creation of a structured safety argument ("safety case"). It can also be used for design verification against the requirements for the design where there is a set of claims that the design meets these requirements. In this sense it could be used to describe how the organization's processes meet a set of external normative requirements ("Standards" in TRAK).

APPENDIX C. —CYPHER QUERIES

C.1 Cypher introduction

The sample queries within this article use CYPHER—the query language that supports the Neo4J graph database. The following is from the Neo4J Developer Manual.[43]

The basic structure of a query is:

- MATCH: The graph pattern to match. This is the most common way to get data from the graph.
- WHERE: Not a clause in its own right, but rather part of MATCH, OPTIONAL MATCH and WITH. Adds constraints to a pattern, or filters the intermediate result passing through WITH.
- RETURN: What to return.

The MATCH and WHERE clauses specify paths that is, one or more 1-tuples, each consisting of a node—connector—node and a direction:-.

```
(a_node:Label {a_property:value})  e.g. (vp:Architecture_Viewpoint:TRAK {name:‘MVp-04 Assurance’})
-[a_relationship:Label]->
```

Labels are used to classify elements in much the same way as a Class or Stereotype might be in other software notations.

A query is an instruction to follow a path in one or more specified directions.

```
MATCH version_includes_change=(mm_spec:Standard {‘DCMI title’: TRAK. Enterprise Architecture Framework. Metamodel})<-[:`IS VERSION OF`]-(ver:Version {`release date`:‘July 2, 2016’})-[:INCLUDES]->(chg:Change)
```

Note that this shows that it is possible to follow paths in opposite directions within the same statement—in this case from Version.

CYPHER also supports a repeated relationship pattern, for example to follow an inheritance tree:

```
MATCH (ntade:Node:TRAK)-[:`INHERITS PROPERTIES FROM`*1..]->(ade:Node {name:"Architecture Description Element"})
```

This starts with a TRAK Node assigned to “ntade” and looks for the relationship INHERITS PROPERTIES FROM one or more times to the specified element(s) at the end of this traversal. This allows a complete tree to be returned without prior knowledge of the depth. To return the inheritance tree for the TRAK System metamodel element:

```
//Return the inheritance tree for the TRAK System metamodel element
//
MATCH inheritance_tree=(ntade:Node:TRAK {name:‘System’})-[:`INHERITS PROPERTIES FROM`*1..]->
(parent_node:Node)
//return the graph
RETURN inheritance_tree
```

which produces Figure C1.

A table output (Table C1) can instead be produced by referring to the specific property in the RETURN clause.

```
//Return the inheritance tree elements for the TRAK System metamodel element
//
MATCH inheritance_tree=(ntade:Node:TRAK {name:‘System’})-[:`INHERITS PROPERTIES FROM`*0..]->
(parent_node:Node)
// Note – using *0.. returns the child System element as well
//return a table
RETURN parent_node.name AS `Node Name`,parent_node.definition AS Definition ,parent_node.comment AS Comment
ORDER BY parent_node.name
```

Note the use of back-ticks where there is a space in a name—`Node Name`

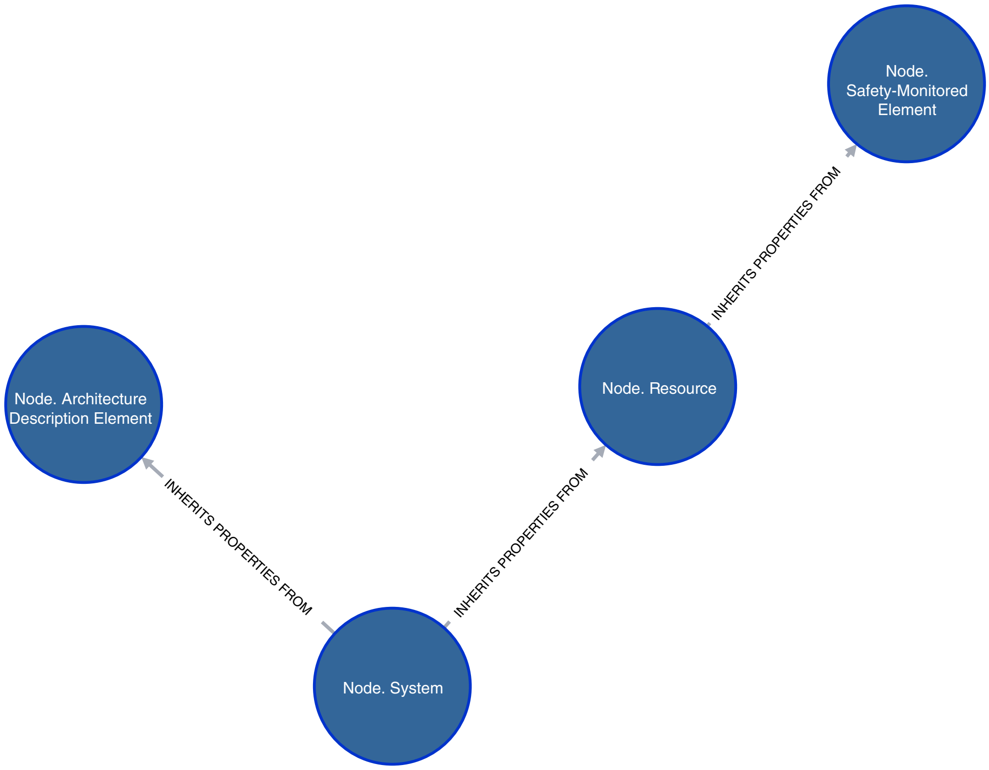


FIGURE C1 Graph of inheritance tree for TRAK system metamodel element

C.2 Queries describing model element versions, lifecycle and change (Section 6.1)

```
//
//List Changes Made to specific version of TRAK Metamodel spec.
//
MATCH version_includes_change=(mm_spec:Standard {`DCMI title`: 'TRAK. Enterprise Architecture Framework. Metamodel'})<-[:`IS VERSION OF`]-(ver:Version {'release date':'July 2, 2016'})-[:INCLUDES]->(chg:Change)
OPTIONAL MATCH chg_request=(chg)-[:`RESPONDS TO`]->(:Change_Request)
OPTIONAL MATCH  deletions=(chg)-[:DELETES]->()-[:FROM]->()
OPTIONAL MATCH additions=(chg)-[:ADDS]->()-[:TO]->()
OPTIONAL MATCH modifications=(chg)-[:MODIFIES]->()
RETURN version_includes_change,chg_request,deletions,additions,modifications
```

which returns the result as Figure 12.

```
//
//Change history/life - TRAK Evidence element
//
MATCH element_birth =(artefact {name:"TRAK. Enterprise Architecture Framework. Metamodel"})<-[:`IS VERSION
OF`]-(v0:Version)-[:INCLUDES]->(c0:Change)-[:ADDS]->(ade2:Node {name:"Evidence"})
OPTIONAL MATCH change=(vi:Version)-[:INCLUDES]->(c1:Change)-[:ADDS|MODIFIES|DELETES]->(ade2)
OPTIONAL MATCH adds_to = (c1)-[:ADDS]->()-[:TO]->(ade2)
OPTIONAL MATCH deletes_from = (c1)-[:DELETES]->()-[:FROM]->(ade2)
OPTIONAL MATCH artefact_versions = (vi)-[:FOLLOWS*1..]->(v0)
RETURN artefact_versions,element_birth ,change,adds_to,deletes_from
```

which returns Figure 13.

The OPTIONAL MATCH statements allow tuples to be added, if found, to the primary MATCH (element_birth) statement. The metamodel element will always be returned but modifications, additions and deletions may not have occurred so the query should never return a null result.

TABLE C1 Tabular output of inheritance tree for TRAK system metamodel element

Node name	Definition	Comment
Architecture description element	An individual architecture description object that is used to describe or represent an item of real-world architecture. An architecture description element can appear in an architecture description.	The parent of all TRAK element types used in ADs. The intent of the attributes is to capture identification and ownership to allow bi-directional exchange of ADs between two organizations.
Resource	Part of the solution—a generic thing that refers to human-related and machine-related entities.	Abstract. Parent class of System, Physical, Software, Human Resource. Not used in any TRAK view.
Safety-monitored element	Something that has possible safety-related attributes	Enables Safety Integrity Level (SIL) to be captured if appropriate. Abstract. Applied to Resource and to Function. Not used in any TRAK view.
System	A composite structure exhibiting emergent behavior.	Can consist of purely human elements. A system is stable and maintains itself this way—the constituent parts must therefore collectively provide the mechanism to keep the system stable. If an essential part of a system is removed it destroys that system (it might become a different system because the system boundary is then different). As ever “the whole is more than the sum of the parts”

C.3 Queries describing metamodel elements and properties (Section 6.2)

```
//
//List TRAK System Metamodel Element properties and Values
//
MATCH adt=(nt:Node {name:‘System’})
OPTIONAL MATCH adft2=(nt)-[r]->(p:Property)
OPTIONAL MATCH nt_propvalue=(p)<-[]-(:Value)
OPTIONAL MATCH ntinhp=(ntp)<-[:`INHERITS PROPERTIES FROM`*1..]-(nt)
OPTIONAL MATCH nt_inhr=(nt)-[:`PARTICIPATES IN SAME RELATIONSHIPS AS`*1..]->(n2)
OPTIONAL MATCH adft3=(ntp)-[r2]->(p2:Property)
OPTIONAL MATCH ntp_propvalue=(p2)<-[:`IS DEFAULT VALUE FOR`|`IS ALLOWED VALUE FOR`]-(:Value)
RETURN adt,ntinhp,nt_inhr,adft3,adft2,nt_propvalue,ntp_propvalue
```

which returns the equivalent of Table 3 for the System element instead of the graphical form of Figure 14.

```
// TRAK Metamodel Table 4.2 Block elements (latest version) - beginning with ’R’
//
MATCH (nt:Node:TRAK)<-[:INCLUDES]-(v:Version)-[:`IS VERSION OF`]->(:Standard {`DCMI identifier`: TRAK00002})
WHERE nt.name=~‘Role’
OPTIONAL MATCH (nt)-[ipf:`INHERITS PROPERTIES FROM`*1..]->(ntcp)
WHERE NOT (v)<-[:FOLLOWS]-(:Version)
WITH nt,++ ntcp.name AS pName
ORDER BY ntcp.name
RETURN nt.`date created` AS `Date Created`, nt.`date modified` AS `Date Modified`, nt.name AS `Metamodel Element (Block)`, nt.`architecture perspective` AS Perspective,nt.definition AS `Definition`,collect(pName) AS `Inherits Properties From`,nt.`tests for` AS `Tests For`,nt.`tests against` AS `Tests Against`,nt.comment AS `Comments`
ORDER BY nt.name ASC
```

which returns part of table 4.2 in the TRAK Metamodel specification (Table 3).

```
//Metrics - Characterise Description of TRAK Metamodel
//
MATCH (adt:Architecture_Description_Tuple:TRAK)
RETURN “No. Metamodel Architecture Description Tuples” AS Type, count(adt) AS Count
UNION
MATCH (ntade:Node:TRAK)-[:`INHERITS PROPERTIES FROM`*1..]->(ade:Node {name:“Architecture Description
Element”})
RETURN “No. Metamodel Node Elements (Architecture Description Element)” AS Type, count(ntade) AS Count
UNION
MATCH (ntade:Node:TRAK)
WHERE NOT (ntade)-[:`INHERITS PROPERTIES FROM`*1..]->(:Node {name:“Architecture Description Element”})
RETURN “No. Metamodel Node Elements (NOT Architecture Description Element)” AS Type, count(ntade) AS Count
UNION
MATCH (nt:Node:TRAK)
RETURN “Metamodel Node elements (Total)” AS Type, count(nt) AS Count
UNION
MATCH (ct:Connector:TRAK)
RETURN “No. Metamodel Connector Elements (Total)” AS Type,count(ct) AS Count
UNION
MATCH (ctade:Connector:Architecture_Description_Element:TRAK)
RETURN “No. Metamodel Connector Elements (Architecture Description Element)” AS Type, count(ctade) AS Count
UNION
 MATCH (p:Property)
RETURN `Properties (All)` AS Type,count(p) AS Count
UNION
MATCH (:Architecture_Description_Element)-[:HAS]->(p_ade:Property)
RETURN `Properties (of Architecture Description Elements)` AS Type, count(DISTINCT(p_ade)) AS Count
UNION
 MATCH (v:Value)
RETURN `Values` AS Type,count(v) AS Count
```

which returns Table 4.

C.4 Queries describing metamodel tuple definition (Section 6.3)

```
//Describes a 2-ary tuple TRAK Metamodel Tuple
//
//First tuple - definition of Resource Interaction from Software
//
MATCH first_tuple_uses_conn=(adt1:Architecture_Description_Tuple {name:Resource Interaction from Software})-
[:`USES CONNECTOR`]->(connector:Connector)
WITH first_tuple_uses_conn,adt1
MATCH first_tuple_path=()<-[:`FINISHES AT NODE`]-(adt1)-[:`STARTS AT NODE`]->()
//
//Second tuple - definition of Resource Interaction to System
//
WITH first_tuple_uses_conn,first_tuple_path,adt1
MATCH second_tuple_uses_conn=(adt2:Architecture_Description_Tuple {name:Resource Interaction to System})-
[:`USES CONNECTOR`]->(connector:Connector)
WITH first_tuple_uses_conn,first_tuple_path,second_tuple_uses_conn,adt2,adt1
MATCH second_tuple_path=()<-[:‘FINISHES AT NODE’]-(adt2)-[:‘STARTS AT NODE’]->()
//
```

```
//2-tuple definition - Resource Interaction from Software to System
//
WITH first_tuple_uses_conn,first_tuple_path,second_tuple_uses_conn,second_tuple_path,adt1,adt2
MATCH two_ary_tuple=(two_tuple:Architecture_Description_Tuple {name:Resource Interaction from Software to
System})-[:`STARTS WITH`]->(adt1)-[:THEN]->(adt2)
RETURN first_tuple_uses_conn,first_tuple_path,second_tuple_uses_conn,second_tuple_path,two_ary_tuple
```

which returns Figure 15.

```
//Quality - Connector element not in any Architecture Description Tuple
MATCH (ade:TRAK)
WHERE (ade:Connector) AND (NOT ()-[:‘STARTS AT NODE’]->(ade) AND NOT ()-[:`FINISHES AT NODE`]->(ade)
AND NOT ()-[:`USES CONNECTOR`]->(ade))
RETURN ade.name AS `Metamodel Connector Element Not Used in Metamodel Tuple Definition`
ORDER BY ade.name ASC
```

which returns Table 5.

```
//Quality - Malformed Architecture Description Tuple
MATCH (adt_no_STARTS_AT_NODE:Architecture_Description_Tuple:TRAK)
WHERE NOT (adt_no_STARTS_AT_NODE)-[:`STARTS AT NODE`]->()
RETURN adt_no_STARTS_AT_NODE.name AS Tuple, ‘No STARTS AT NODE relationship’ AS Warning
```

which returns Table 6.

C.5 Queries describing and checking architecture viewpoint definitions (Section 6.4)

```
//
//Describe TRAK MVp-04 Assurance Viewpoint Purpose & Governance
//
MATCH vp_purpose_governance=(av:Architecture_View:TRAK)<-[:GOVERNS]-(vp:Architecture_Viewpoint
{name:‘MVp-04 Assurance’})-[:`FRAMES`]->(adt1:TRAK:Concern)<-[:HAS]-(st_holder:Stakeholder)
RETURN vp_purpose_governance
```

which returns Figure 16.

```
//
// First 12 tuples addressing MVp-04 Assurance Viewpoint Concerns
//
MATCH (vp:Architecture_Viewpoint:TRAK {name:‘MVp-04 Assurance’})-[:FRAMES]->(conc:Concern)<-
[:ADDRESSES]-(adt:Architecture_Description_Tuple)
RETURN vp.name AS Viewpoint, conc.name AS `Viewpoint Concern`,adt.name AS `Addressed Using`
ORDER BY vp.name ASC,conc.name ASC,adt.name ASC
LIMIT 12
```

which returns Table 7.

```
//
// Quality - Identify tuples required to address viewpoint concerns missing from well-formedness criteria
//
MATCH (vp:Architecture_Viewpoint:TRAK {name:‘MVp-04 Assurance’})-[:FRAMES]->(conc:Concern)<-
[:ADDRESSES]-(adt:Architecture_Description_Tuple)
WHERE NOT (vp)-[:`REQUIRES AT LEAST`]->(adt)
RETURN adt.name AS `Tuples Missing from Well-Formedness Definition`
```

which returns Table 8.

C.6 Queries describing the implementation of a metamodel using an ADL (Section 6.5)

```
//
//Return 10 (random) elements from the UML profile for TRAK implementation and the TRAK elements represented
//
MATCH (TRAK_element)<-[:`TO REPRESENT`]-(adl_element)-[:`IS A`*..]->(adl_parent)<-[:`HAS PART`]-
(adl_metamodel:Metamodel {name:’UML’})
OPTIONAL MATCH (imp:Implementation {name:‘UML profile for TRAK’})-[:USES]->(adl_element)
WITH imp.name AS Implementation,adl_element.name AS `ADL Element` ,adl_parent.name AS `ADL
Parent`,labels(adl_parent) AS `ADL Element Type`,TRAK_element.name AS `TRAK Element`,labels(TRAK_element)
AS `TRAK Element Type`,rand() AS number
ORDER BY number
LIMIT 10
RETURN Implementation, `ADL Element`,`ADL Parent`,`ADL Element Type`,`TRAK Element`,`TRAK Element Type`
```

which returns Table 9.

```
//TRAK metamodel elements missing from implementation using ’UML Profile for TRAK’
//
//Establish latest version of Metamodel - based on Model Configuration & Change Viewpoint
//
MATCH (v:Version)-[:`IS VERSION OF`]->(trak_mm_spec:Standard {name:‘TRAK. Enterprise Architecture Frame-
work. Metamodel’})
WHERE NOT (v)<-[:FOLLOWS]-(:Version)
WITH v AS latest_version
//
//Find metamodel elements not implemented - using ADL Implementation Viewpoint
//
MATCH (latest_version)-[:INCLUDES]->(trak_el:TRAK)-[:`INHERITS PROPERTIES FROM`*0..]->(ade:TRAK
{name:‘Architecture Description Element’})
WHERE NOT (:Implementation {name:’UML profile for TRAK’} )-[:USES]->(:UML )-[:`IS A`*0..]->(:UML)-
[:`TO REPRESENT`]->(trak_el)
RETURN latest_version.`release date` AS `Metamodel Version`, trak_el.name AS `TRAK Architecture Description
Element`,labels( trak_el) AS `Type(s)`,“Missing from \‘UML profile for TRAK\’” AS Implementation
ORDER BY trak_el.name
```

which returns Table 10.

Note that this tests for the absence of a label “UML” which is applied to the elements representing UML metamodel elements within the model.