34th Annual INCOSE international symposium
hybrid event
Dublin, Ireland
July 2 - 6, 2024

Modeling Principles to Moderate the Growth of Technical Debt in Descriptive Models

Ryan Noguchi, The Aerospace Corporation

Agenda

- Introduction—The Discipline of Model Architecting
- Technical Debt in Descriptive Models
- 5 Model Federation Principles
- 3 Model Layers Principles
- 6 Modeling Domain Principles
- 4 Modeling Semantics Principles

Introduction—The Discipline of Model Architecting

- Architecting principles and heuristics
  - Guide architects and facilitate rapid and effective architectural decision-making
  - Similar principles are needed to facilitate the practice of model architecting
- This presentation describes 18 modeling principles
  - Commonly observed in modeling practice
  - Strongly influence model technical debt
  - Consider them when making decisions about model architecture and implementation
  - Many of these principles are primarily oriented toward object-oriented (OO) models
    - Particularly those using SysML or UAFML
- None of these principles are absolute
  - Each represents a concept that is generally beneficial, not necessarily universally ideal

The Technical Debt Concept

- Technical debt is a widely-used concept in the software domain:
  - A metaphor for development or sustainment costs deferred to the future (Cunningham, 1992)

- Like financial debt, technical debt represents the deferred costs of repairing, reworking, or replacing a product that wasn't built perfectly from the beginning
  - Created "when developers violate good architectural or coding practices, creating structural flaws in the code" (Curtis et al.)

- Much research and evolution of practice in the software domain is focused on managing technical debt
  - Limiting its growth
  - Paying off debt over time
  - Avoiding growth of technical debt resulting from ignorance

Cunningham, W. (1992). The Wycash portfolio management system, in Addendum to the Proceedings of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), ACM Press.
Curtis, B., Sappidi, J., & Szynkarski, A. (2012a). Estimating the Size, Cost, and Types of Technical Debt. 3rd International Workshop on Managing Technical Debt.

5 Model Federation Principles

- A model federation is a distributed set of models connected by a controlled set of model usage relationships
  - To enable their content to be shared whilst retaining their autonomy and ability to evolve independently
- Federation adds complexity and requires active architecting attention
  - Tradeoffs among different federation architectures and approaches

Model Federation Principle: Consider Federation Early (CFE)

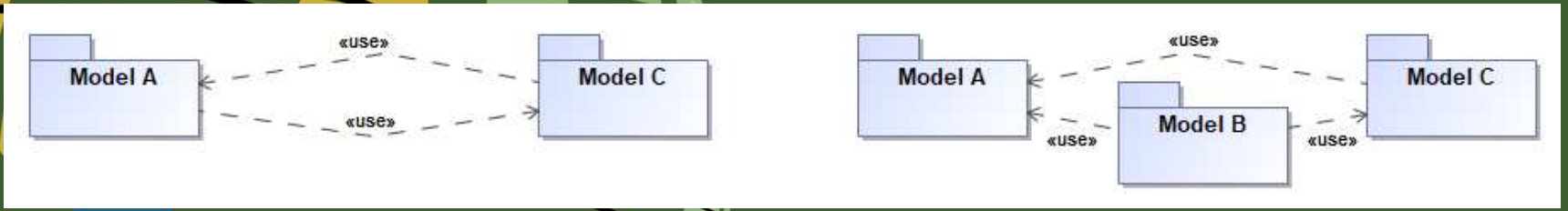
- Most modeling efforts begin with a single, monolithic model
  - Scaling and federation are often afterthoughts
- As the model grows in size and use, the value of federation grows
  - Changing model federation architecture becomes increasingly difficult as the model grows
  - Often results in a large “balloon payment”
- Taxes associated with delayed federation are deceptively low
  - Waiting until “the last responsible moment” to make model architectural decisions often results in being too late to avoid significant rework costs
- Modeling projects generally should address federation proactively
  - Not all model projects should use federation
  - Federation should be done to address specific functional objectives for the models

Model Federation Principle: Partition for Cohesion (PfC)

- A key role of architects is partitioning a large component into discrete components
  - And allocating responsibilities and defining interfaces between components
- Model partitioning should be driven by considerations of both model governance and model usage
  - These often represent competing sources of tension for model partitioning
- Align model partitions with organizational responsibilities for model content
  - Improves efficiency of model governance
- Model usage is facilitated when fewer model boundaries are crossed in queries
  - Each boundary crossing can result in semantic mismatch, complicate query construction, or impede tool performance
- Impact of misalignment—too much coupling, not enough cohesion—can manifest in substantial additional effort needed to build, sustain, and use those models

Model Federation Principle: Federate to Interrogate (FtI)

- Directional usage relationships between federated models should be designed to avoid model usage cycles or interdependencies
  - These can result in performance problems
- The diagram on the left depicts a simple dependency cycle
  - Federations should be architected to avoid cycles
  - Federations should align model usages with the most important query navigation paths
- Queries navigating reverse paths can be accommodated by introducing an additional "bridging" model (Model B) as shown in the diagram on the right



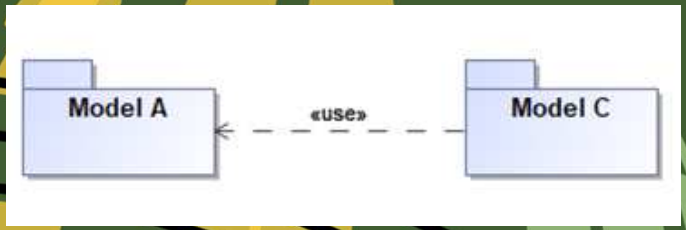
1) a model usage cycle

2) Breaking the cycle via a "bridging" model

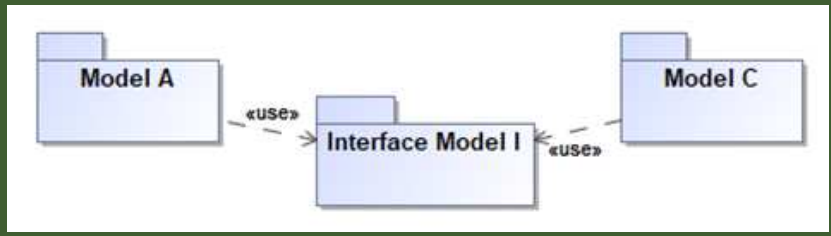
Martin, R. & Martin, M. (2006). Agile Principles, Patterns, and Practices in C#. Pearson.

Model Federation Principle: Dependency Inversion Principle (DIP)

- Analogous to the OO software principle of the same name
  - "More abstract software modules should depend on less abstract modules" (Martin & Martin, 2006)
  - More abstract modules are typically more stable than more concrete modules
  - Easier to manage dependencies when those dependencies are more stable
  - As a result, violating DIP is riskier than following DIP



Dependency is from the more concrete model C to the more abstract model A, consistent with DIP



An interface model can insulate two models from volatility in the other; model I is more abstract (and stable) than models A and C

Martin, R. & Martin, M. (2006). Agile Principles, Patterns, and Practices in C#. Pearson.

Model Federation Principle: Define Clear Interfaces (DCI)

- Connections between federated models should establish consistent mechanisms for:
  - Semantically connecting concepts represented by model elements
  - Querying those models.
- Connect specific model elements with specific relationships carrying specific semantics
- It is particularly vital to define model interfaces when connecting modeling “dialects”
  - Every SysML model represents a potentially different domain language with its own concepts and its own context
- Failing to implement a standard often results in inconsistent modeling patterns being used and makes it difficult for users to get correct answers to their queries

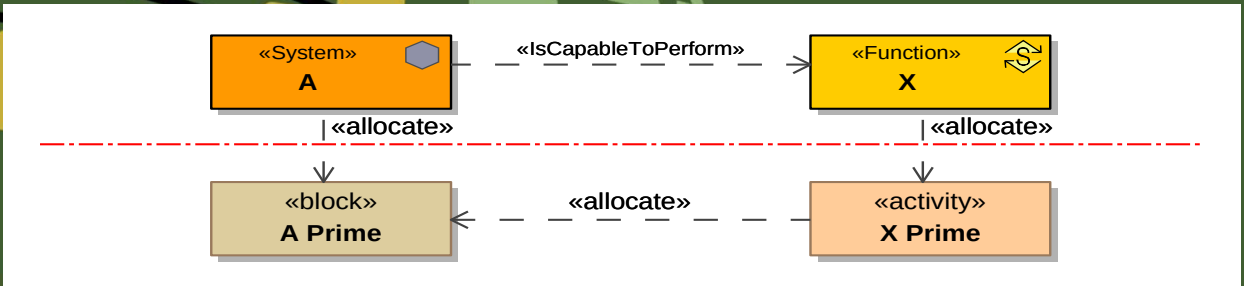


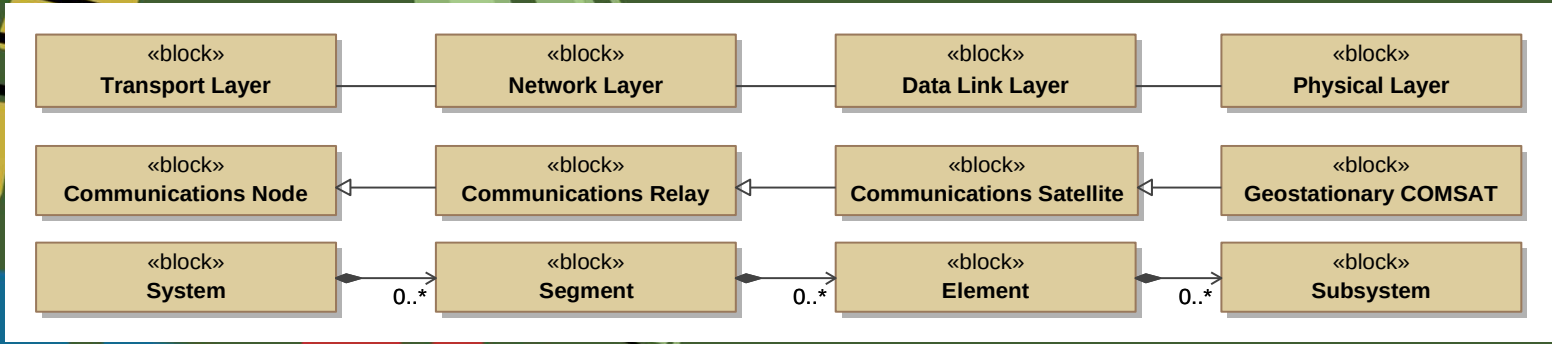
Figure adapted from:
Martin, J.N. & Brookshier, D. (2023). Linking UAF and SysML Models: Achieving Alignment between Enterprise and System Architectures. 33rd INCOSE International Symposium.

3 Model Layers Principles

- Descriptive models are constructed with multiple layers
  - **Layers of abstraction** facilitate separations of concerns
  - **Taxonomic layers** represent key domain concepts at different levels of granularity
- The model architect must make critical decisions about the definition of these layers
  - Selecting the right **number** of layers
  - Identifying the **intent** of each layer
  - Defining the **interfaces** between layers

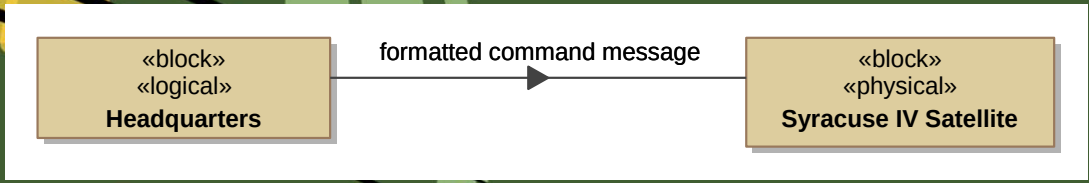
Model Layers Principle: Just Enough Layers (JEL)

- Too few abstraction layers increases risk that additional intermediate layers must be added later
- Too many abstraction layers drives additional complexity and taxes
- Too few taxonomic layers can create difficult rework for each usage or specialization of each model element
- Too many taxonomic layers can lead to an explosion of additional model elements



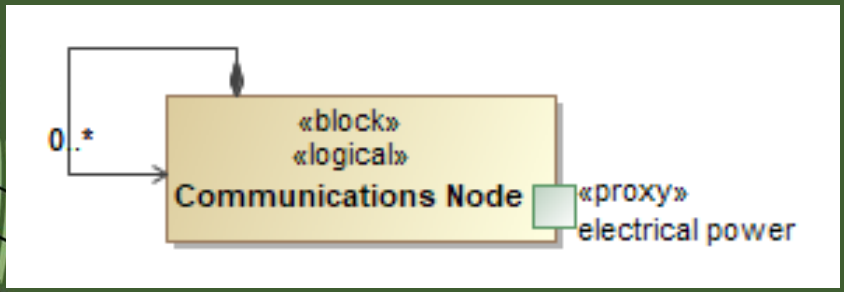
Model Layers Principle: Don’t Cross Streams (DCS)

- Avoid creating connections between abstraction layers
  - Communication across abstraction layers dilutes separation of concerns
  - Often results when mixing structure and abstraction in a single hierarchy
  - Also seen when combining contextual tenses within the same model
  - Easily renders the model inconsistent, incoherent, and very difficult to correct
- Modeling methodology should clearly define the use of these layers and identify specific interface points between those layers
  - Enforce consistency and avoid breaking encapsulation



Model Layers Principle: Scale Free Decomposition (SFD)

- Modelers often overuse bespoke levels of decomposition
  - A specific taxonomy of distinct element types at each level
- Often, the natural representation is a tree structure where composite and atomic elements can be treated equivalently
  - e.g., a scale-free (recursive) mode of decomposition
  - Minimizes the addition of extraneous decomposition layers
  - Also facilitates satisfaction of the other two Model Layers Principles
    - Just Enough Layers
    - Don't Cross Streams



6 Modeling Domain Principles

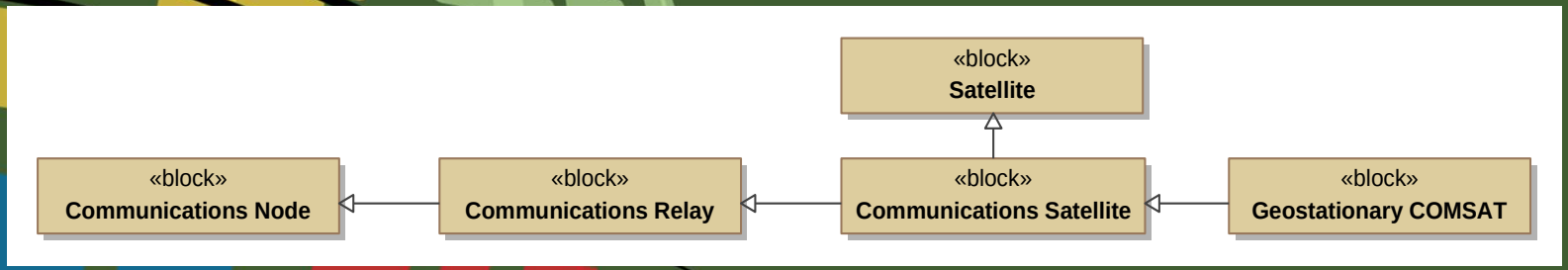
- Descriptive models for MBSE must accurately represent the domain being modeled
  - At least to the extent that it answers the stakeholders' questions
- Misalignment between the way concepts are modeled and stakeholders' expectations of those concepts is a significant source of technical debt

Modeling Domain Principles: Other People's Profiles (OPP)

- Avoid reinventing the square wheel
- While it can be very useful to create new metamodels and profiles to better represent vital concepts within the model's domain, this also has a downside
  - Often, homegrown approaches are narrowly focused and poorly documented, reducing their reusability and understandability.
- Unique profiles must be continually maintained through the life of the project
  - Often this maintenance cost can be significantly reduced by extending standards
- By converging on a preferred set of these standards and contributing to their evolution, the MBSE community can better leverage reuse, improve model interoperability, and more efficiently use their resources

Modeling Domain Principles: Single Responsibility Principle (SRP)

- Definitions should be minimal in scope
  - Represent a single coherent concept, not a combination of multiple distinct concepts
- A composite concept is usually best represented using multiple inheritance
- Failure to adhere to this principle often results in the need to eventually break up the definition into its constituent components
  - This results in substantial propagation of rework to the specializations and usages of that definition as well as users of those usages
- Modelers very experienced with OO programming can be subconsciously biased against multiple inheritance due to their software experience

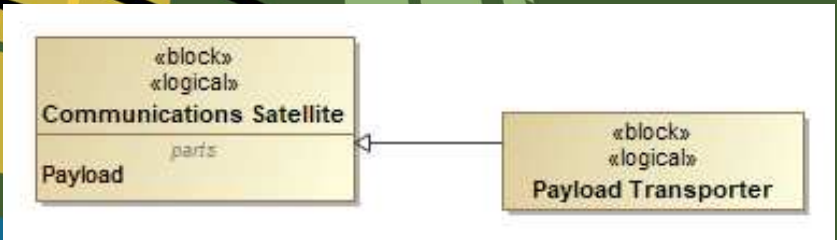


Modeling Domain Principles: Open/Closed Principle (OCP)

- Model element type definitions should be designed to be specialized without needing to be modified
  - Typically, modification is needed because the definition is overly constrained
  - While it isn't possible to anticipate all future contexts, enabling future flexibility is not difficult when done early
- In this figure, the Communications Satellite block has a Payload part with the default multiplicity of 1
  - This multiplicity limits its use in the frequently observed cases in which the satellite bears multiple payloads
  - If the multiplicity is corrected, rework is driven to many of its specializations
  - That rework typically requires manual inspection of each specialization to determine if rework is warranted

Modeling Domain Principles: Liskov Substitution Principle (LSP)

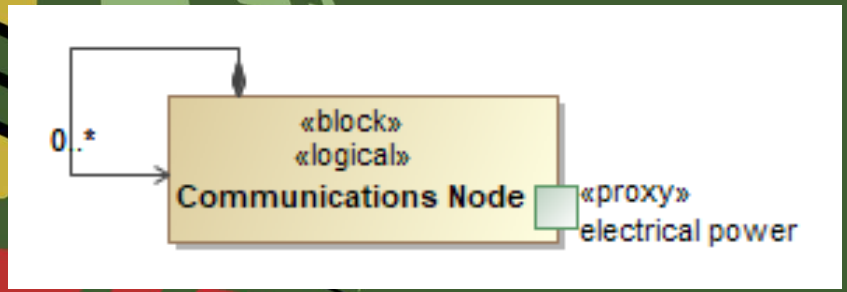
- Any specialization of a base classifier should be a valid substitute for that base classifier (Liskov, 1988; Martin, 1996)
- In descriptive models, generalization should be reserved for those contexts for which that substitution is valid
  - Not just used as a convenient mechanism for reuse
- In OO software, inheritance is often avoidable as most use cases for inheritance are better implemented by composition
  - However, in OO modeling, inheritance carries semantics of substitutability and should be reserved for that purpose



Liskov, B. (1988). Data abstraction and hierarchy. SIGPLAN Notices 23.
Martin, R. (1996). The Liskov Substitution Principle. C++ Report, Vol 9 (2).

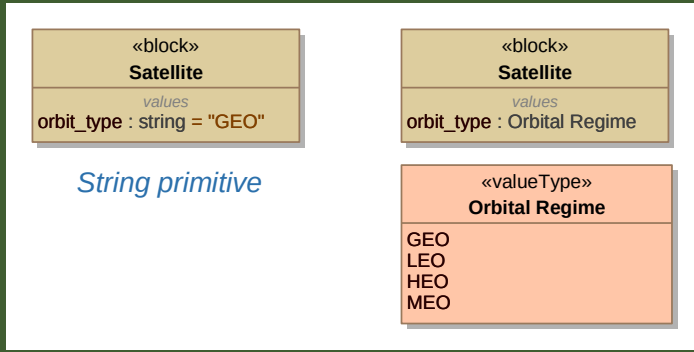
Modeling Domain Principles: Interface Segregation Principle (ISP)

- Type definitions should avoid defining features that are not required (or even meaningful) in all of its specializations
- In the diagram below, the Communications Node has an electrical power port whose multiplicity value makes it non-optional
  - As a result, all Communications Nodes must have one, even if not needed
  - It is often difficult to deactivate those features for those specializations that don't use them

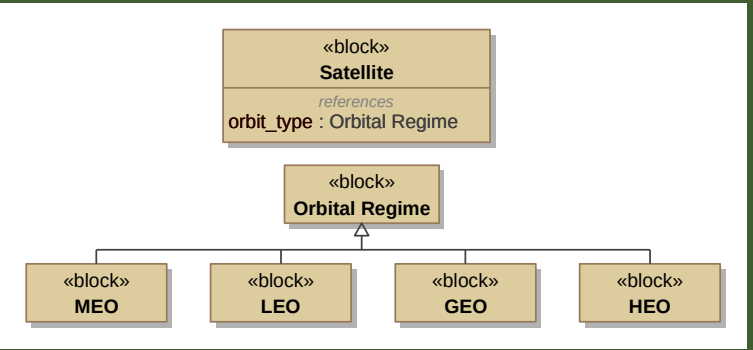


Modeling Domain Principles: Reify for Reuse (RfR)

- Modelers often rely too heavily on primitive types
- Enumerations are often a better choice
- If a concept is frequently reused in different contexts it is often better to **reify** it
  - i.e., create a class to represent it
- This provides greater flexibility to model users and facilitates model maintenance and evolution
  - The concept appears only once in the model rather than countless times buried within other model elements
  - Using a model element rather than an enumeration offers opportunities to take advantage of generalization



Enumeration



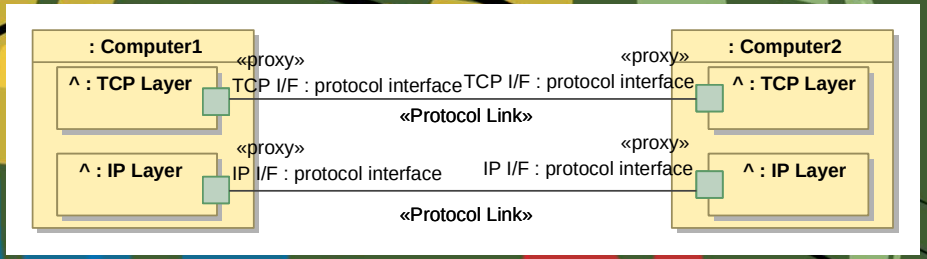
Reification

4 Modeling Semantics Principles

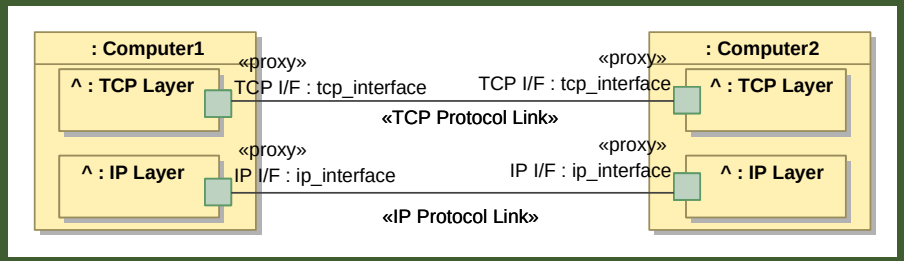
- Some of the most common but least visible problems in models are semantic in nature
  - Inconsistent use of modeling constructs, resulting in ambiguity
  - Mismatches between the modeler's and model users' expectations of the semantics of the concepts being modeled, resulting in confusion or misinterpretation

Modeling Semantics Principle: Avoid Undertyping

- Modelers frequently underspecify model elements
  - Often the intent is to avoid being constrained by deciding on a narrowly defined type or stereotype
- In the figure on the left, the single stereotype «Protocol Link» types all connections
  - Regardless of whether they are appropriately connected at that level of abstraction
- Often, a suitable corrective action is to create a set of specializations of the type or stereotype to use in these different roles, as shown on the right
  - Here, «TCP Protocol Link» and «IP Protocol Link» are specializations of «Protocol Link»
  - This allows the different types or stereotypes to be treated either as equivalent or different



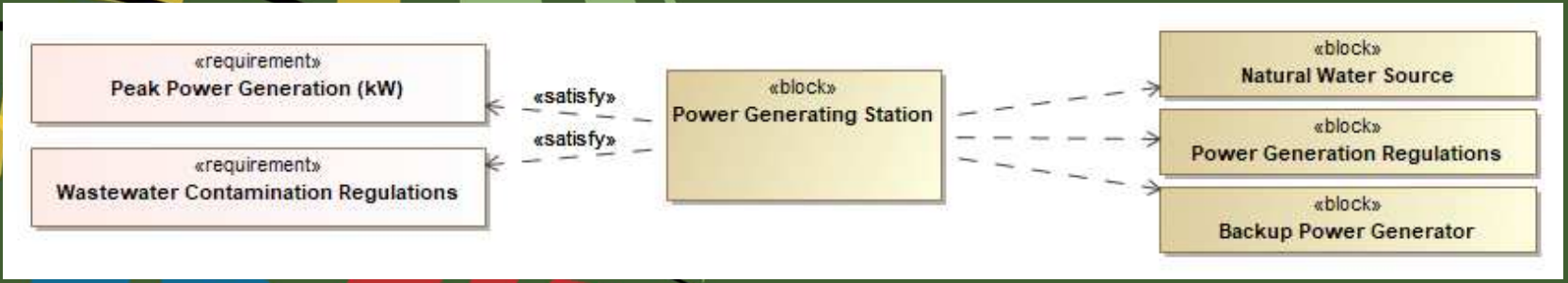
Example of undertyping



Example without undertyping

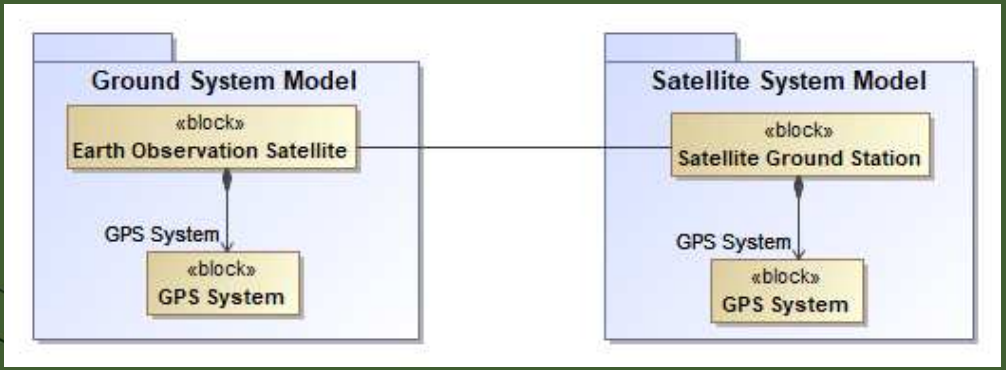
Modeling Semantics Principle: Avoid Overloading

- Modelers often overload concepts with multiple meanings
- In this example of semantic overloading the «dependency» and «satisfy» relationships are each used in the same model to express multiple distinct meanings
- Overloading results in rework that can be very difficult to find since each instance must be assessed separately to determine which of the overloaded meanings is the correct one
- This model view shown is not necessarily *wrong*, but overloading adds some risk of technical debt that more refinement will be needed in the future involving major rework



Modeling Semantics Principle: Avoid Composition Misuse

- Modelers often overuse composition when describing entities that are not intended to be duplicated
  - Composition establishes the existence of an individual usage of a block definition in the separate context
    - That individual usage is distinct and separate from every other usage
  - While this is often appropriate, in other cases it is dangerous and misleading
    - In the diagram below, the two model elements representing the GPS system are intended to represent the same system, not two separate copies of that system
- This problem often does not manifest when models are used in isolation but emerges when models are federated



Modeling Semantics Principle: Intrinsic is Permanent

- Modelers often use permanent modeling constructs to represent characteristics that are transitory or context-dependent
  - This results in ambiguity or misinterpretation when the context changes
- E.g., using stereotypes for a “system of interest” or a “stakeholder”
  - Both concepts are context-dependent, not intrinsic properties of the modeled entity
  - This can significantly hinder the model’s reuse and interpretation within a federation
- Instead, contextually dependent characteristics should be modeled using modeling constructs that properly convey the context in which the assertion is being made
  - e.g., being a stakeholder is usually better modeled as a role or a relationship
  - The nature of that relationship may differ significantly between stakeholders

Summary

- This presentation described 18 key modeling principles
  - These principles reflect over a decade of observation and experience of many descriptive modeling efforts and their need for rework to accommodate their changing context.
- These principles are not fundamentally novel or unprecedented
  - Most are related to principles and heuristics well known in system architecting and software domains
- Each of these principles represents a suggested modeling choice
  - Intent is to reduce the assumption of excessive technical debt in these models
- Following these principles does not guarantee success
  - However, neglecting these principles elevates risk

34th Annual INCOSE international symposium

hybrid event

Dublin, Ireland
July 2 - 6, 2024

www.incose.org/symp2024
#INCOSEIS