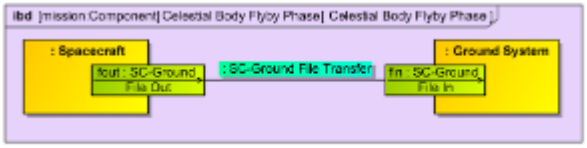
Interconnection Pattern

Synopsis

An interconnection is a connection between elements, such as a connection between a spacecraft and ground system, or a connection between a power switch and an electrical load. Use this pattern to describe connections between elements, in a context, through which energy or material or information flows. This pattern describes what flows through an interface and allows capture of those attributes in a particular context.



Pattern Overview

Status	Tool Version
In Work	SSCAE MagicDraw Packages versions 1702SP3-02 or later

Line Org. Owner	Submitter	Point of Contact
3101 - Engineering Development Office	IMCE Pattern Consolidation Working Group	Dan Dvorak <daniel.l.dvorak@jpl.nasa.gov>

Related Patterns

- Interface Definition Pattern: For specifying interfaces and junctions between interfaces
- Structural Context Pattern: For specifying context in which interconnections may be asserted.
- Interaction Pattern: For specifying interactions between functions
- Requirements Definition Pattern: For requirements specification, including specification of interface requirements
- Reconciliation/Abstraction Pattern: For constructing and reasoning across multiple levels of abstraction (logical/physical, conveyance of data across networks, etc.)
- Characterization Pattern: For describing values related to the analysis of an interface. This is useful when the specification is not yet determined and there are trades to be explored.

Skip directly to the SysML Examples...

Table of Contents:

Applicability

To help users assess the applicability of this pattern to their work (i.e., to the problem they want to solve or their area of interest), we describe the way in which this pattern addresses a few kinds of common concerns. In particular, we address:

- Content concerns: the kind of content users can capture in this pattern
- Artifact concerns: the kinds of artifacts (documents and views) that can come from this pattern
- Reasoning concerns: the kind of reasoning (analysis) that this pattern is meant to support
- Assumptions: what we expect to be true about the user's situation that is relevant to whether they can or should use the pattern.

Content Concerns

This pattern provides a mechanism for enumerating **as-exercised** connections between elements **within the scope of a specified context.** Where the Interface Definition Pattern facilitates the definition of what connections **may** exist between elements, this pattern allows the user to assert that, within the context of interest, previously defined interfaces are exercised. A complete representation of an interconnection includes:

- A context. The context is the domain or scope in which the connection between the elements is of interest.
- A set of participants. These are the elements which present the connectable interfaces. These elements "play a part" in the context.
- Connections. These are assertions that the Junction "plays a part" by connecting its associated interfaces in the context of interest.

At this generic level, we are concerned with the ability to cordon off a domain of interest and assert the junctions and interfaces that we want to have available for use in that context. While the Interface Definition Pattern allows the user to specify what is **allowed** by design, this pattern specifies the orchestration of those junctions within discrete contexts.

Artifact Concerns

This pattern supports the enumeration of content that is present in the following conceptual systems engineering artifacts[note]:

- Functional Block Diagram (FBD)
- Interface Requirements Document (IRD)
- Interface Control Document (ICD)
- System block diagram with interconnections
- DODAF SV-1 Systems Interface Description
- DODAF OV-2 Operational Resource Flow Description
- DODAF SV-2 Systems Resource Flow Description
- DODAF SV-3 Systems-Systems Matrix
- N-squared diagrams
- Deployment Diagrams (Testbeds, etc.)

Note: we refer here to the underlying content present in these artifacts, rather than any particular paper examples. This pattern is not "how to make an IRD;" instead, we assert that this pattern supports the capture of much of the content one would find if one examined many examples of IRDs and retained the core attributes and concerns found therein.

Generic Reasoning Questions

This section captures some may be answered by the relationships and content present in the pattern. This is not an exhaustive list; there are other examples for particular domains, etc. This is another useful way for users to determine whether they have the same reasoning needs as the pattern developer, and thus whether the pattern is suitable for their use.

- Does an Interface Definition (two interfaces and a junction) exist for every asserted connection?
- Are any Interface Definitions not exercised in any context?
- Are all connected interfaces of compatible types? (end-to-end as well as piecewise compatibility)
- Given two interfaces, in what contexts is their connection exercised?
- For a given context, what items are exchanged between systems (N^2 chart).
- Given a component, what is connected to this component through any of its interfaces?
- Given a component, what is connected through a specific interface?
- What components produce messages or flows with some specific (property, value, data type, stereotype, etc.)?

Data Domain:

- For a given context, what are the access points to each participant?
- What is the set of protocols that each participant must implement?

Assumptions

There are currently no assumptions made about the user's situation relevant to this pattern.

Pattern Implementation

Here we describe the elements that make up the Interconnection Pattern, their relationships, responsibilities, and collaborations. The solution does not describe a particular concrete design or implementation. Instead, the pattern provides an abstract description of a design problem and how a general arrangement of elements solves it. The solution is presented first in in modeling language independent terms (in the Ontology Description Section) and then as a SysML embedding (in the SysML Implementation Section).

Generic/Ontology Implementation

We start with a set of components and a context in which they connect and exchange items or information. We want to assert that all interconnections we create between the components are true in that context. Ideally, we start with a set of predefined Components, Interfaces, and Junctions as described in the Interface Definition Pattern - this is our library of possible connections. Within the context, we can then assert the intended connection.
Click here to expand/collapse the rest of this section...

SysML Implementation

The concepts we described in the last section are mapped to concrete implementation in SysML so that they can be used to actually define interfaces in a model. In this section we describe first the embedding of the ontology into SysML (so that the user can understand how the concepts are made concrete) and then provide examples.

Embedding

The following table describes how the elements in the ontology appear in SysML. The easiest form of mapping is when one ontological concept is represented by one SysML element: for example, a JunctionParticipant as described in the ontology is mapped to a Connector in SysML. However, one-to-one mapping is not always possible, or even the best mapping.

Ontology	Classification	SysML	Metaclass	Stereotype
Context	Concept	Element (any element that can contain properties) Block, Component	Classifier?	n/a
ComponentParticipant	Concept	PartProperty	Property	<<Part Property>> (from SysML)
InterfaceParticipant	Concept	Port typed by Interface	Port	<<Proxy Port>>
JunctionParticipant	Concept	Connector typed by Junction	Connector	n/a
Context-specific Presents	Relationship	Existence of ComponentParticipant typed by Component owning Port typed by Interface	n/a	n/a
Context-specific Joins	Relationship	Existence of connector which 1) connects two ports typed by two interfaces which are 2) joined by the Junction which types this connector	n/a	n/a

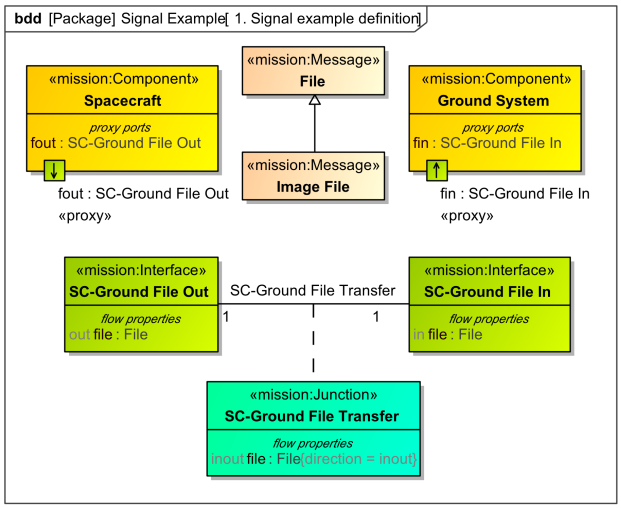
Concept Mapping: The context and all participants map to SysML Metaclasses directly (although ComponentParticipants are currently restricted to PartProperties, although they could be more generic properties). No special stereotypes are necessary for the embedding.

Relationship Mapping: There are no specific stereotypes or elements to which the Joins and Presents map. The embedding is more of a restriction stating that a ConnectorParticipant can only join the interfaces joined by the Junction by which it is typed and that a ComponentParticipant can only present InterfaceParticipants which are presented by the Component by which it is typed.

SysML Examples

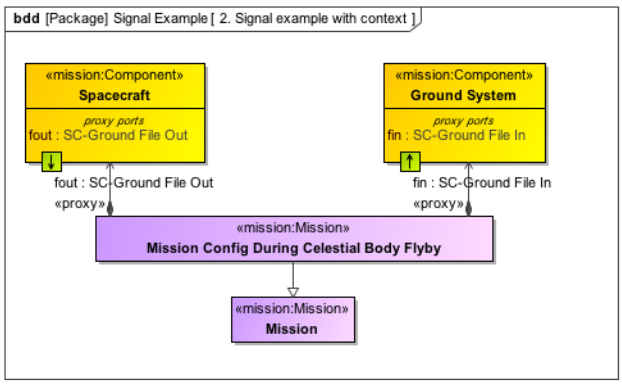
The following are simple pedagogical examples which make use of the concepts and their embedding into SysML:
Simple example: Spacecraft & Ground System exchange a file

Image	Explanation



Here we see the definition level elements in our system: A spacecraft, a ground system, and the interfaces they present to exchange Files. The interfaces are joined by the SC-Ground File Transfer Junction. We also see that we can be more specific than just "File" and identify the existence of "Image Files" as well.

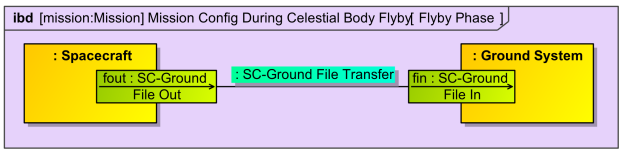
Note: for an explanation of the directions of the flow properties in the Interfaces and the Junction, see the discussion of flow direction in the Interface Definition Pattern.



We now assemble our Spacecraft and Ground System into a Context. Notice that nothing special is required to identify the context as such; by owning and mediating information exchanged between participating components, it becomes a context.

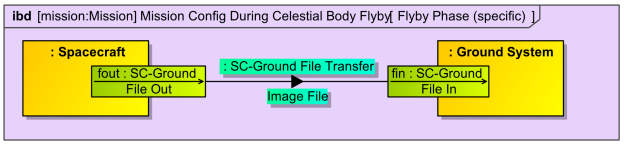
Note also that we have a generic Mission of which Mission Config During Celestial Body Flyby is a specialization. We define attributes that are universally true for the Mission at the Mission level, and define structure and attributes specific to the Celestial Body Flyby part of the mission in that context.

Note: if the Spacecraft and Ground System can be defined once across the entire Mission, it would be simpler to draw the composition relationships from the Mission to the SC and GS rather than to the Celestial Body Flyby element as we have done here. We discuss the pattern for decomposition (including inheritance and redefinition) in another pattern.



Here we see the inside of the Mission Config During Celestial Body Flyby element. We see the Spacecraft and Ground System playing roles, and we have drawn a Connector between the ports these participants present. We have typed the Connector with the SC-Ground File Transfer Junction to complete the pattern.

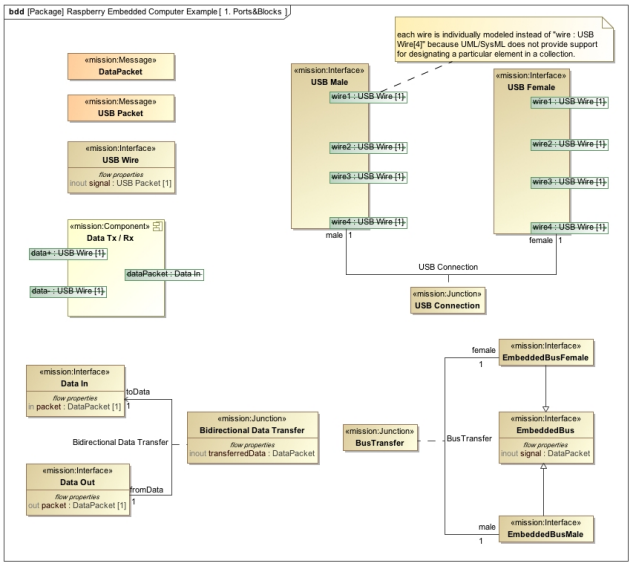
Of particular importance here is that the Mission Config During Celestial Body Flyby context now OWNS the Connector between SC and GS. This is extremely useful because it is a formal expression of utilizing that interface within that context. The context may be queried for all owned Connectors, and that set represents every connector that may be used in that context (i.e., if you leave out a connector, you may not traverse that junction in that context).



Optionally, one may use the SysML construct of Conveyed Information to informally show more specific flows and add some convenient visual cues (i.e., the arrow). Here,we show the same exchange of "file," but in this case it happens to be Image Files (which is a specialization of File). Use of Conveyed Information in this form is a shorthand for defining a new Junction. The Conveyed Information should be interpreted as implying a specialized Junction of SC-Ground File Transfer where Image File is the traversing element instead of File.

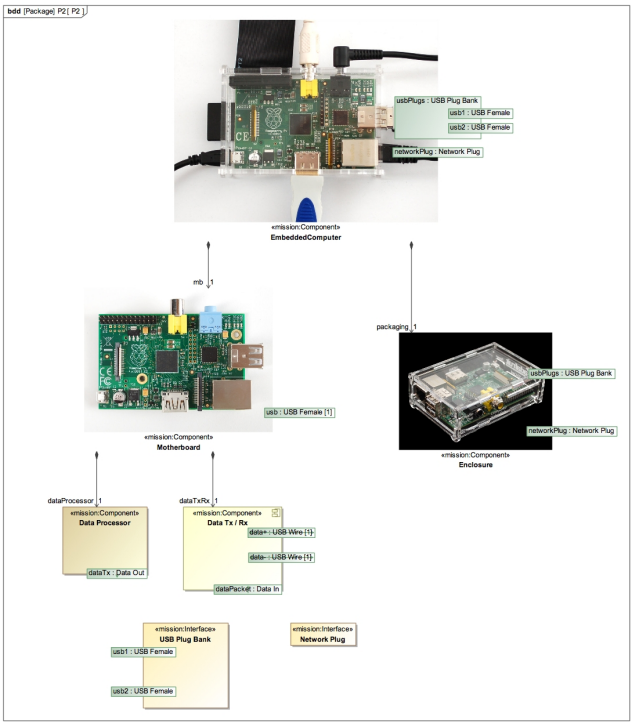
SysML Example: Raspberry Pi

Image	Explanation

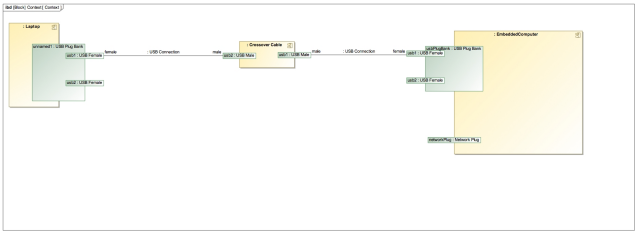


We first identify the definitions of the things we wish to express connectivity between (Components, Junctions, Interfaces, Messages). We have joined interfaces transferring DataPackets, and a USB interface that calls out USB wires individually.

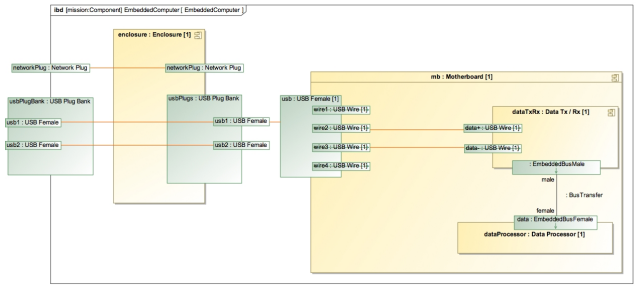
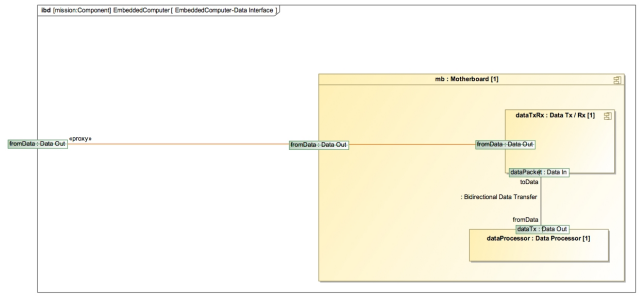
Note: for an explanation of the directions of the flow properties in the Interfaces and the Junction, see the discussion of flow direction in the Interface Definition Pattern.



Here we show the system definition: the embedded computer contains a motherboard and some packaging. We want to show how the USB connection is made to the motherboard through the packaging. We also have a laptop with two USB ports and a crossover cable.

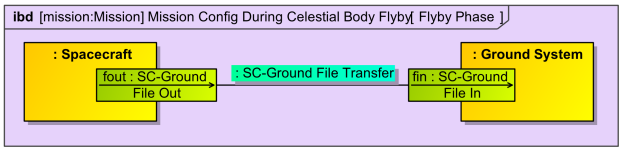
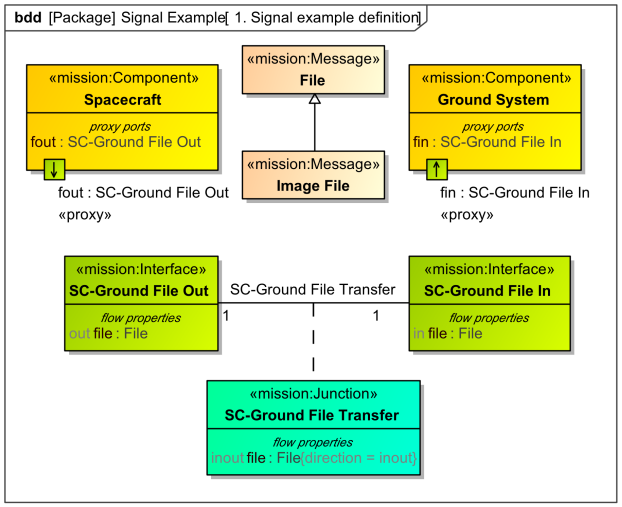
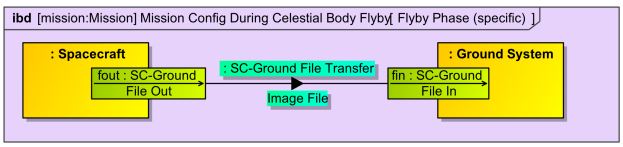


This diagram shows the usage of the crossover cable to connect the laptop to the embedded computer. The connectors are typed by the USB Connection Junction.

	Here we show how the USB interface on the EmbeddedComputer component is actually connected to the Data Tx/Rx component in the Motherboard. The orange connectors shown here are Binding Connectors, which indicate identity (the external USB Female interface IS the USB Female interface on the Packaging which IS the USB Female interface on the Motherboard, and the wires there ARE the wires on the Data Tx/Rx). Another way to think of it is that the USB Female interface on the Motherboard is actually exposed through the Packaging to the outside. We also see the connection between the Data Tx/Rx and the Data Processor via a Bus Transfer.
	The previous diagram showed a very low level exchange of packets - what if we want to show a higher level, logical data exchange? Here we see the connectivity of Data in the system - the Data Tx/Rx data port is exposed to the outside, and internally can exchange data with the data processor. How to map between these levels of abstraction will be addressed in a future pattern.

Fun with Reasoning Questions

Reasoning Questions Explained: Spacecraft & Ground System exchange a file

Image	Explanation
	**Q: Does an Interface Definition exist for every asserted connection?** A: Yes. We can tell because there is only one connection in this example and it is typed by a junction, which means that there must be an appropriate interface definition... (if it were not typed a junction, i.e. untyped or typed by something else, that would be a validation error...?) If it is not typed by a junction, we can't be sure that the interfaces are compatible. **Q: Are all connected interfaces of compatible types?** A: Yes. We assume that we have already validated the all interface definitions in this model, so since we are able to type the connector with the junction means the interfaces match. See the Interface Definition Pattern (TODO - add anchor to validation stuff) for description of checking interface correctness. **Q: For a given context, what items are exchanged between systems?** A: In this example, the only exchange is between the spacecraft and the ground system. More generally, we can find all of the owned connectors for the context block, look at their ends (nested connector end or part with port) - TODO - nested connector end stereotype not always being set, when done manually, property path not set...
	
	**Q: Are any Interface Definitions not exercised in any context?** A: No. In this example, there is only one interface definition pair and only one connector, and the connector is typed by the junction. More generally, you can find the set of junctions by finding all connectors and getting their types, and compare that to the set of junctions that exists in the model. If there are extra junctions or connectors, there is a mismatch.

Rules/Axioms/Invariants

The following table expresses the modeling rules:

ID	Modeling Rule
1	All content in an context must be consistent / exist / asserted in the model (hiding a model element or connection from a diagram must be for aesthetics, not for semantics / design / assertion). You must be able to display every connection in that context and have it be true (but maybe ugly). Union of IBDs in a particular context should show truth for that context.
2	All connectors in a context must be valid/true/exist for the "duration" of the context.

Syntactic Rules

JPL SysML Restriction Rules

ID	Restriction Rule	Owning Package
JPL-SysML-07	A binding connector must not have a type	
Onto-Connector-01	A binding connector can be used in several ways. These are enumerated in Onto-Connector-01.1 & Onto-Connector-01.2.	JPL Onto SysML Restrictions about Connectors
Onto-Connector-01.1	In an IBD, a binding connector can be used to connect a nested proxy port of a proxy port on a part to an another proxy port within the same part context.	
Onto-Connector-01.2	Several cases must be distinguished for binding connectors involving non-port properties, these are enumerated in Onto-Connector-01.2.1, 01.2.2, 01.2.3, 01.2.4, & 01.2.5.	
Onto-Connector-01.2.1	A proxy port may have a SysML flow property typed by a Mission:MaterialItem (i.e., a kind of SysML Block). Such flow property must be bound by a binding connector to a SysML part property typed by a compatible Mission:MaterialItem. The part property bound could be inside or outside the block owning such a proxy port.	
Onto-Connector-01.2.2	A proxy port may have a SysML flow property typed by a Signal. The binding of such a flow property depends on the isBehavior meta-property of the proxy port.	
Onto-Connector-01.2.3	A proxy port may have a SysML flow property typed by a SysML ValueType. Such flow property must be bound to a SysML value property typed by a compatible SysML ValueType.	
Onto-Connector-01.2.4	A proxy port may have a SysML reference property. Such a reference property must be involved in an end-to-end chain of binding connectors, one end of which must be a part property, the other end of which must be a reference property and all intermediate bindings must involve either a proxy port with a single reference properties or a reference property in the IBD context.	

Onto-Connector-01.2.5	A proxy port may have a SysML value property typed by a SysML ValueType. Such value property must be bound to a SysML value property typed by a compatible SysML ValueType inside or outside the part or block owning the proxy port.	
Onto-Connector-02	An assembly connector must be typed by a SysML AssociationBlock	JPL Onto SysML Restrictions about Connectors
Onto-Connector-03	An assembly connector relates proxy ports typed by Mission:Interfaces joined via a directed Mission:Junction typing the assembly connector. If the proxy ports joined by an assembly connector have nested ports or nested properties, then interconnections for the nested ports and properties must be defined in an IBD typing the assembly connector instead of nested connectors in the IBD where the assembly connector is.	JPL Onto SysML Restrictions about Connectors
Onto-Connector-04	SysML Connectors are partitioned in two disjoint classes: binding and assembly connectors	JPL Onto SysML Restrictions about Connectors
Onto-Connector-05	An assembly connector can only be the realizing connector for 1 item or information flow	JPL Onto SysML Restrictions about Connectors
Typing-01	A Connector can only be typed by a kind of Association	Allowed Property Type Rules
Typing-05	A ParticipantProperty can only be typed by a Block	Allowed Property Type Rules
Typing-06	A PartProperty can only be typed by a Block	Allowed Property Type Rules
Workaround-01	InterfaceBlock can have only 1 Flow or Reference property	JPL SysML1.3 Workarounds

Model Implementation Concerns

Connectors between Nested Ports:

- Connections between nested ports can be shown in one of two ways: 1) with connectors between the nested ports owned by the original context, or 2) with connectors owned by a Junction. There is an important trade in picking one of these two approaches. If connectors are shown between nested ports on the original context, then these connectors cannot be reused in other contexts. On the other hand, when they are owned by a Junction, the relationship between nested ports can be reused across multiple contexts but the nested connectors cannot be shown in the IBDs owned by other contexts.

Conveyed information:

- The SysML concept of Conveyed information represents a short-hand for defining specialized Junctions for a particular connector. These should be transformed into Junctions before preforming analysis on the model. Conveyed information is also useful as a visual reference, but be careful to keep this conveyed information consistent with the directions specified in the interface definition.

Multiple Connectors attached to a Port:

- We recommend that you **avoid** attaching multiple connectors to one port.
- Because: there does not exist a clear way to specify the routing and/or duplication of signals and tokens when multiple connectors leave a port. While UML/SysML Activity Modeling provides constructs like Fork, Join, Decision, and Merge nodes, nothing analogous exists for the Class/Block/Part side of UML/SysML. To avoid ambiguity, we suggest that modelers avoid attaching multiple connectors to a port. Workarounds include:
    - Creation of routing components containing behaviors for determining information destination at runtime
    - Creation of bus-type components containing binding connectors between input and output ports for expressing balancing of continuous flows
    - Analysis of layers of abstraction to determine if multiple connectors is really necessary (if you are expressing interfaces, does it make sense to have a one-to-many interface at a logical level? At a physical level, is it actually possible to have a one-to-many connection?)
    - Inheritance: can you express the interface once between generic components instead of many specific ones?

- If modelers cannot avoid using multiple connectors to one port, we recommend that they either choose a consistent meaning for multiple connectors, or attach a specification of logic for information is to be produced on / collected from multiple connectors (such as a characterization or an attribute on the interface definition).

Completeness:

- Fill in: see comment in Jira. TODO

Validation/Well-Formedness Reasoning

- Every part with a connected port is typed «mission:Mission» or «mission:Component».
- Every connected port is typed «mission:Interface».
- Every connector is typed «mission:Junction».

Supporting Scripts/Tooling

We do not currently provide any supporting scripts or tooling.

Tooling Tricks

Trick	How to do it	Why?
Quickly type Connectors with an appropriate Junction	Right-click on the connector. You will see "Association" in the menu, near the bottom. When clicked, a list of available Junctions will appear for selection. Note: a Junction is only available if a) the connected ports are typed by Interfaces and b) a Junction exists between those Interfaces.	Faster, easier modeling!

Open Questions

Questions:

1. **Does the ontology need to have a concept of a context?** Resolved: no.
2. **What happens if the multiplicity of the association block is greater than 1?** Discussion: Should it be allowed? (Currently: no, it is not allowed.) What would that mean for connectors? (discussion 7/9/13)
3. **What is methodology or rules for when to make something a context?** Discussion: who is deciding where the boundaries are? What do we mean by context change? Structural configuration change? Depends on the stakeholder / engineer who needs to understand the distinctions. Conversation with Nicolas R. about consistency of information between IBDs in the same context. In this pattern we assert that if IBDs share a context, they must be consistent views of a consistent system. It appears that SysML allows an IBD to ALSO be a context, meaning that two IBDs could have their own elements. This seems very confusing and hard to manage (and would require BST expansion to make it work), so we are proposing in this pattern that people create a new context if they wish to create an IBD that would introduce inconsistency.
4. **Asserting a lack of connection:** How do we assert that something is **not** connected? A multiplicity 0 connector would be nice but is not currently possible. Concern is that absence of connector may not be intentional. Possibility of "null connector."
5. **How to show structural configuration changes?** For example, if your antenna is covered during ascent, or your pipe becomes disconnected, and is capped, etc.

References **and Pattern Resources**

Currently no Working Group approved references. See the Community Page: Interconnection Pattern references area for unofficial references.

Modeling Guide (somewhat outdated): SysML Modeling Guide

Further Examples

Currently no Working Group approved examples. See the Community Page: Interconnection Pattern examples area for further examples.

Community Page

The Community Page: Interconnection Pattern has been set up to collect Frequently Asked Questions, Discipline Specific (and extended)

examples and reasoning, and References. Everyone should have write access and are free to discuss and contribute.

Copyright