Structural Context Pattern

Synopsis

A structural context is the setting in which modelers may assert and describe the paths of interchange between internal components. The structural context pattern provides semantic support for the goal of defining traversable connections within the scope of the context under study. The context element, participating internal components, and asserted connections fall in the *structural* domain; this pattern does not address *functional* or *behavioral* aspects. This pattern is concerned with asserting the *who*, *what*, and *where* aspe cts of exchanges while functional contexts tend to address *when* an d *why* connectivity may be exercised.

For example, the umbilical connection between a launch tower and a rocket exists only in a "pre-launch configuration" context. The purpose of the Structural Context Pattern is to show how a modeling element can establish a context for other elements and connections that exist in that context. Specifically, a context-providing element owns a set of references to those other elements and connections. A context can be a mission phase or system configuration, an assembly (where we want to design or capture its inner workings), a logical system (a distributed system like a thermal subsystem, for example), or even a cross-cutting view for analysis (end-to-end information flow, fault containment regions, etc.). Use the Structural Context Pattern to specify the connections that are in scope for a particular situation or state.

Structural Context Icon

Pattern Status Updates:

⚠ Some of the links to other patterns will appear to be broken - this will happen if we are linking to a pattern that is still in work!

Pattern Overview

Pattern Status	Tool Version
In Work	

Line Organization Owner	Submitter	Point of Contact
3101	IMCE Pattern Consolidation Working Group	Dan Dvorak

Related Patterns

- Interconnection Pattern: For creating the connections internal to a context.
- Structural Decomposition Pattern: For describing structural decomposition.
- Interface Definition Pattern: For specifying interfaces and junctions between interfaces
- 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.
- Requirements Pattern: For requirements specification, including specification of interface requirements
- Interconnection Pattern: For describing interconnections.
- Structural Context Pattern: For describing the contexts in which interconnection can occur.
- 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.
- Meta-pattern Notes : For constructing and analyzing deployments of your interfaces in specific contexts (scenarios, mission phases, testbeds, etc.)

Go directly to 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

The Context Pattern addresses identification and population of those settings and circumstances in which we wish to define interconnectivity. Moreover, the pattern accounts for the definition of multiple sets of settings and circumstances in which the system interconnections may differ from those of other settings and circumstances. That is, the Structural Context Pattern adds an additional set of concerns to those of Interconnection, namely, the partition of the system description into disjoint sub-descriptions that may correspond to the evolution of the system over time, to distinct aspects of the system, or some combination of the two.

In summary:

- Definition of disjoint system descriptions (parameterization by mission phase; by evolving system structural configuration, etc.) the union of which make up the system description ("What contexts does my system provide?")
- The mechanisms for asserting that components and connections participate in various system descriptions ("how do I build a context?")
- The consequences and rules that apply when you build a specific system description ("what have I asserted about each system description?")
- How to leverage inheritance, redefinition, and re-use to simplify the building of system descriptions ("how do I do it efficiently?")

We are **not** fully covering all of structural decomposition - additional information can be found in the Structural Decomposition Pattern. While the Structural Context pattern cannot be implemented without utilizing structural decomposition, the assumption is that most of the relevant structural decomposition has already been captured using the Structural Decomposition Pattern. Here the focus is on techniques for cleanly describing rich internal connectivity for a given context.

This pattern provides guidance and structure for defining a context and subsequently asserting that a set of connections is in scope for that context. Where the Interconnection Pattern describes how to make the internal connections of the context, this pattern covers the mechanism for ensuring that the connections are properly scoped to the context. It also provides guidance and discussion about the motivations and rationale for context selection, and ramifications to model structure and capabilities.

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) (where there is context-specific information)
- Interface Control Document (ICD) (where there is context-specific information)
- System block diagram with interconnections
- The following DODAF views (assuming context-specific information is present) (What is DoDAF?):
    - 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 (assuming context-specific information)
- 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

- What set of interfaces and junctions are present in this structural context? (Note - behavior models may be used to elaborate when, why, and how these connections may be "active"; however, the scope of this pattern is limited to asserting which connections are present physically or by design).
- Does the union of all views associated with a context visually present, at least once, every connection scoped to that context?

Assumptions

- To make use of this pattern, we assume the modeler is aiming to represent a multi-dimensional system, whether in time (mission phases, scenarios, etc.), in location (deployment of components), or other concerns (security zones, component suppliers, design authority, etc.)
- We assume that the modeler has already done the predecessor work of selecting and building the decomposition trees, analysis views, etc. that are to be used in the system model (via the Structural Decomposition Pattern). This pattern does not provide guidance about doing that selection; rather; it supports the elaboration and assertion of detail about the interior of whatever decomposition level you are working in. Note that the exercise of decomposition and context definition can happen iteratively; we wish to emphasize here that doing the decomposition and construction of trees is not covered in this pattern.

Pattern Implementation

Here we describe the elements that make up the Structural Context 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 the problem of representing and maintaining context and how a general arrangement of elements solves it. The solution is presented first in modeling language independent terms (in the Ontology Description Section) and then as a SysML embedding (in the SysML Implementation Section).

Generic/Ontology Implementation

The word "context" originates from the Latin words *con* and *textere,* which mean "together" and "to weave" respectively. We use the word to mean the collection and connection of elements that are relevant to some concern, goal, location, assembly, etc. Practically, any time an element has internal structure, it implicitly provides a context in which that internal structure may be seen (and, to go back to our definition, the context is where the internal structure is "woven" together).

"Context" is a convenient word to refer to the arrangement of the internal structure of an element – and it turns out we don't need a special element in the ontology to represent it. Certain types of elements may provide a context or be in a context, so when we

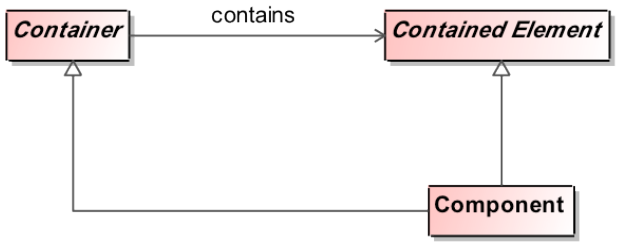
define the parts of the ontology that relate to contexts, we do it in terms of Containers (of other elements, which necessarily means the container is/has a context) and ContainedElements, which are kinds of elements that can "participate" in a context.

Summary

1. The idea behind a context is really useful. With context...
    a. We can assert the existence of internal structure and how it is interconnected
    b. The internal structure and interconnections are inherently bound by anything that binds the context - whether that is lifetime (when the context is "active") or other composition rules (if the context element participates in another structure, all internal parts and connectivity come with it).
2. The idea of a context is relevant for structure AND function, but in this pattern we are only addressing structure.
3. We **don't need** an object in the ontology called "context."
    a. The ability to provide context (i.e., contain and interconnect other components) is inherent to the ontological concept called **Container.**
    b. The ability to participate in a context (i.e., be contained) is inherent to the ontological concept called **ContainedElement.**
    c. The assertion that a **Container** has a participating **ContainedElement** is called **Contains**.

Ontological Elements

As we mentioned before, we don't actually have an ontological concept called "Context." Instead, we have base:Container, base:ContainedElement, and the base:contains relationship between them:

Image	Description
 Note: we have omitted the "base" and "mission" prefixes on this diagram.	Let's talk about the base:contains relationship. There are two ways we can think of "containment": 1. Namespace containment - the definition of an element is within the definition of another element, such as nested classes in software. This form of containment is entirely scoped to the definition level. Think the "circle-and-crosshairs" type of nesting of SysML. 2. Composition containment - this is best described with some qualitative heuristics about the container and the contained: a. The elements have a part/whole relationship b. They have the same life-cycle c. When one is moved, the other moves with it d. When one is destroyed, the other is also destroyed.

In this pattern, our focus is on the second kind of containment. We are not trying to define the package structure or class nesting; rather, we are talking about how elements participate in the definition of other elements.

You may be wondering why only mission:Component is shown on this diagram, when it seems that Missions, Junctions, etc. could be a type of *base:Container*. You're right! Those other elements are implementations of *base:Container* and *base:ContainedElement*. For the sake of a simple diagram, we chose not to try to draw every "descendant" of *base:Container* and *base:ContainedElement*. For a full set, check out the definitions of Container and ContainedElement in the Base Ontology and look for those super classes asserted in the definitions of concepts in the Mission Ontology.

Wait a minute, you're saying. I'm confused about this base:contains relationship! Can a *base:ContainedElement* be in multiple *base:Container* ? How do I handle my various kinds of structure and organization in my system!? Good news, everyone! We discuss that in the Structural Decomposition Pattern. In this pattern, we assume that you have solved that problem and are dealing with expressing the connections **within** the contexts you've created.

SysML Implementation

The concepts we described in the last section are mapped to concrete implementation in SysML so that they can be used to encode system information 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.

Ontology	Classification	SysML	Metaclass	Stereotype

base:Container	Concept	*base:Container* does not map precisely to one SysML metaclass. *base:Container* most closely matches any SysML element that can own properties; however, we restrict *base:Container's* presence in SysML to exist through it's "descendants": the IMCE stereotypes listed at right.	Structured Classifier	«mission:Component», «mission:Junction», «mission:Item», «mission:Flow»
base:ContainedElement	Concept	Like *base:Container, base:ContainedElement* does not map precisely to one SysML metaclass. *base:ContainedElement* most closely matches any SysML element that can be the type of a property; however, we restrict *base:ContainedElement's* presence in SysML to exist only through it's "descendants": the IMCE stereotypes listed at right.	Type	«mission:Component», «mission:Junction», «mission:Item», «mission:Flow»
base:contains	Relationship	A base:contains B if there exists a property typed by B in the namespace(what?) of A. The association/aggregation relationship (black/white diamond association model element) is helpful but not sufficient, as the former rule can be satisfied without an association relationship.	Composite Aggregation Association (optional)	n/a

Concept Mapping: *base:Container* and *base:ContainedElement* are abstract classes (signified with italics on diagrams), which means that they are not implemented directly, but instead contribute meaning to their "descendants." The ability to provide a context for other elements (*base:Container*) is already inherent in the category of SysML elements that can own TypedElements. Similarly, *base:ContainedElement* already exists in SysML as Types (i.e., things that can become the type of a property).

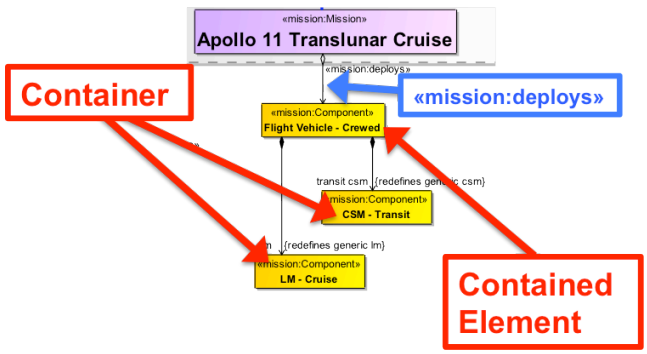
Relationship Mapping: The base:contains relationship is not made explicit in SysML, as it is inferred by looking at the members of a *base:Contaner*. If the *base:Container* owns members (specifically, structural kinds of properties, such as Part Properties or Reference Properties) whose type is a *base:ContainedElement* (rather than a String or a Value Type), then the *base:Container* base:contains the *base:ContainedElement* .

The Structural Decomposition Pattern addresses how hierarchies of contexts should be derived, projected into analysis space, and mapped to each other. The rules we describe in this pattern are universal to contexts that describe groupings of elements according to analysis concerns AND to more classical structural decomposition.

SysML Examples

Let's look at how our ontological concepts appear in SysML using a small set of elements (this is followed by a far more complex example):

Image	Description

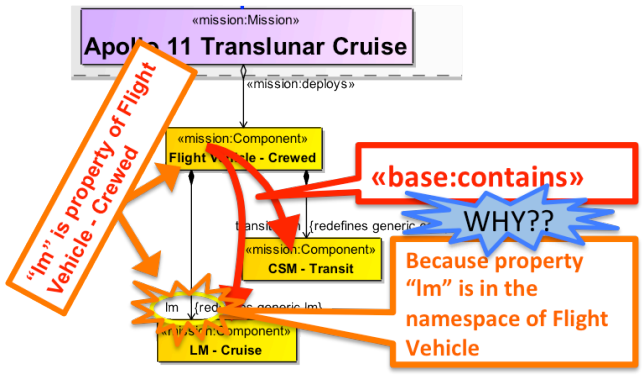


Let's look at a small part of our system during the purple Translunar Cruise phase.[1] In this phase we have the crewed flight vehicle, which consists of the combined Command / Service Module (CSM) and the Lunar Module (LM). (Don't worry, there are lots of other parts but we're not showing them yet). We've highlighted the *base:Containers* and *base:ContainedElements* found on in this small structure.

[1]: We are representing mission phases as specializations of a generic "Apollo 11 Mission." You will see the relationship between this phase and the larger mission later in this pattern.

This kind of structure is probably not new to you, but we show it to illustrate two of our concepts of context: *base:Containers* and *base:ContainedElements*. All the «mission:Mission»s or «mission:Component»s that are "made up of" other components (i.e., have part or reference properties) inherently provide context for relating those components. A good test for whether something provides a structural context is whether you can or would want to make an IBD of its internal structure.

A *base:ContainedElement* is generally something that you think you might want to put on an IBD: «mission:Component»s, for the most part (although Junctions, Items, and Flows are also *base:ContainedElements*).



So where does the base:contains relationship come in?

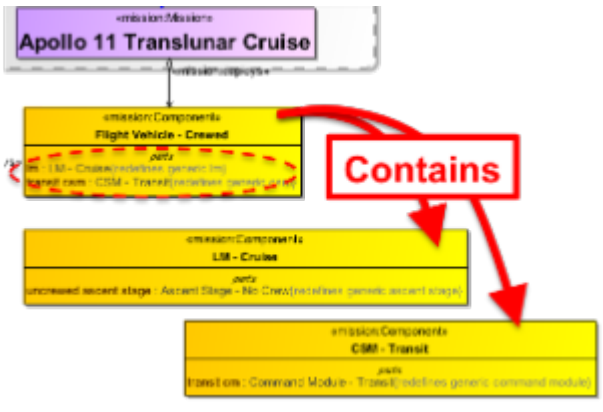
We have a base:contains relationship between two elements (specifically between a *base:Container* and a *base:ContainedElement*) if the *base:Container* owns (or nests, like an inner class in software or the circle-and-crosshairs in SysML) a **property** that is **typed** by the *base:ContainedElement*.

Modelers usually assert this "A-owns-property-typed-by-B" pattern[1] by drawing an aggregation relationship (black and white diamond) between two elements. Using aggregation relationships like you see at left automatically populates this pattern in the model: in the example, drawing the association between "Flight Vehicle - Crewed" and "LM - Cruise" caused the automatic creation of a property called "lm" (typed by "LM - Cruise") in the namespace of "Flight Vehicle - Crewed".

[1]: See the embedding for base:contains in the SysML embedding table for more details.

The actual association relationship object is not mandatory in SysML - you can manually create a property in a block and type it with another block. When you do that, you assert that a base:contains relationship exists even though there will be no corresponding association relationship in the model. While it is uncommon to create part or reference properties relationships without using composite aggregation relationships (black diamond association), it is semantically valid.

In the image at right, we have removed the association relationships from the model and re-created the properties manually. The assertion that the Flight Vehicle base:contains the LM and CSM is still present due to the existence of the typed properties in the Flight Vehicle.



Does this mean that there is a base:contains relationship between a component and value properties? No - click here for an explanation. Why did we use the directed composition? Click here for a discussion of navigability.

SysML Example: Investigating Contexts through the structural configuration changes of the Apollo 11 Mission

Over it's mission timeline, Apollo 11 demonstrated drastic structural configuration changes - not only as modules detached and reattached,

but as the three crew members physically moved about within the assembly structure of the mission. We examine some of those differences here to illustrate context modeling.

Expand to continue reading...

SysML Example: Review of Inheritance and Re-use in a Context

Use inheritance to define the features and structures that are common to entire categories of element at a higher level - one time.

Expand to continue reading...

SysML Example: Review of Redefinition in a Context

We use redefinition to "override" generic attributes with context-specific ones.

Expand to continue reading...

Fun with Reasoning Questions

Expand to continue reading...

Rules/Axioms/Invariants

Potential Rules:

- Existence of part property typed by ContainedElement implies contains relationships between classes
  - only part properties?
- Part properties and connectors must be nested / in the namespace of element acting as the Container. For connectors, this means the "lowest common denominator" container in the decomposition hierarchy in this scope. IBDs should also be within the namespace of the block they describe, but that is not mandatory. MagicDraw enforces the property and connector nesting rule.

Model Implementation Concerns

Elements Appearing in Multiple Contexts

We will address, in the Structural Decomposition Pattern, the issue of maintaining model consistency when elements are contained by more than one other element (such as when elements actually change containment over the life-cycle of the mission, or when an element exists in both a structural decomposition AND an an analysis context.

Please see the open questions section for a more detailed description of options for asserting disjointness in time.

Model Structure Implications

- Something is only scoped to a context if it is within the namespace of the context. That means connectors and part properties must be nested in the context block (container). MagicDraw enforces this rule for structural properties and connectors, but not diagrams. Recommended practice is to also keep IBDs owned by their context (although it's possibly to navigate back to the context block because IBDs have a "Context" attribute).

Defining and Inheriting Context "Invariants"

- If some components in a system do not change throughout all phases or deployments, it is permissible to compose them into a generic context and then specialize it to show variations.
- When to use redefinition.

Modeling buses, broadcast, multicast, etc.

In the Interconnection Pattern we stated that one should avoid attaching multiple connectors to one port. Our rationale is that doing so creates ambiguity - is information always sent on all connectors? If not, which connector is picked? How do you describe that? What is the behavior in receiving multiple inputs simultaneously? To help clarify our guidance for dealing with this concern, we will look at three approaches and discuss their pros and cons.

Expand to continue reading...

Validation/Well-Formedness Reasoning

This section describes the set of assertions that relate to correct implementation (well-formedness) of the pattern.

- The union of all views describing a context should show every property (part, connector) owned by the context at least once.
- A view that displays all properties (parts, connectors, etc.) of a context must be "true" (if ugly).

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

Question	Discussion
What is methodology or rules for when to make something a context?	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.
How do you assert semantically that contexts are disjoint in time?	⚠This should probably be moved to the Structural Decomposition Pattern. How do we express in a query-able way that two contexts cannot exist simultaneously? How to infer that a part is the same in two contexts that are disjoint in time? There are a couple of options: 1. Convention: if we reserve black-diamond composition to mean whole/part with existence/destruction semantics, and we inspect (through code or by hand) trees to determine whether they are members of contexts which ultimately specialize a super-context (think of the mission phases specializing a generic Apollo 11 mission context), then we can adopt a convention that those elements are singleton classes and the specializations of the super-context are disjoint in time. 2. Additional imposed semantics: a "Singleton" stereotype combined with either "disjoint" relationships between contexts or traversal of some "mission timeline" concept placing contexts on a timeline could more rigorously assert disjointness in time and continuity of elements throughout contexts.
I am not ready to model in such detail, how much of the pattern is actually required?	We do not provide (at this writing) a mechanism for "sketching" patterns. In order to use reasoning and other tooling provided by IMCE to support this pattern, it must be fully implemented. However, there is nothing preventing modelers from "sketching" patterns - perhaps with light weight dependencies, information flows, or other model elements that are consistently used for "sketching."

Why aren't Interfaces in the Ontology as ContainedElements and Containers?	Interfaces do not "contain" other elements in the ontological sense of contains. Interfaces "present" other interfaces; are "traversed" by items, and cannot "contain" internal structure (recall that value properties are not internal structure). Interfaces are therefore not a kind of Container. They are also not contained by any of the Containers (their relationship with other interfaces is "presents"). You wouldn't expect to see an interface on an IBD directly, for example. Thus, interfaces are not a kind of ContainedElement either.
Does "contains" go between ANY Container and ANY ContainedElement?	We need more use cases to fully answer this - however, at the moment Steve says that contains should be "homogenous" in the sense that Items should contain other Items, Flows contain other Flows, Components contain other Components, etc. The exception is that Missions can contain Components, Missions, and Junctions and Components and Junctions can contain other Components and Junctions.
From Bjorn's comment... **should contexts be defined by composition of other elements, or should the context "be state variables that constrain when a given relationship is active (or we just don't care one way or the other). There are hints that Functional or Behavioral constraints will find some merit in being done the same way?"**	We do not preclude people from modeling using that approach; however, we find the expressivity provided through use of composition to define contexts to be extremely valuable for many problems encountered on projects (particularly in cases where interconnection, routing, and paths are under inspection). We see these alternatives providing different benefits under different circumstances and this pattern is concerned with the composition approach.

Advanced Questions

A container "contains" contained elements. Does that mean value properties are contained elements?
Do I have to use directed composition?
How do you encode role names in the ontology if part properties are not actual objects in the ontology?
How do we tell inherited properties apart? A.K.A. What the heck is a property path?
Modeling "degenerate" cases (A.K.A. "My dog only has THREE legs!")

References and Pattern Resources

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

Community Page

The Community Page: Structural Context 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