

Modeling and Measuring Software Component Architecture in SysML

Please send questions/comments to DigitalEngineering@saic.com

Introduction

Systems Engineers are responsible for producing requirements, including software requirements. This paper describes a method for developing, modeling, and measuring software requirements in a Model Based Systems Engineering (MBSE) environment using Systems Modeling Language (SysML). The output of the method is called a software component architecture.

This paper will demonstrate how to construct a software component architecture in SysML using the IFPUG method to perform the Systems Engineering process of Software Functional Analysis. This Analysis activity pertains directly to the CMMI Process Areas of Requirements Development, Measurement and Analysis, and Estimation.

Background

SysML is a language for Systems Engineers that supports the analysis and specification of a System using partitions called Blocks as elements of its composition.

- Blocks are modular units of system description.
- Blocks provide a general-purpose capability to model systems as trees of modular components.
 - These include modeling either the logical or physical decomposition of a system, and the specification of software, hardware, or human elements

The language was developed by leveraging certain concepts from the Unified Modeling Language (UML).

UML is a language that supports the specification of Software Systems. SysML a derivative, or dialect of UML, with extensions, to precisely capture characteristics of a System that do not necessarily pertain to software.

Our goal is to identify the Blocks that represent Software Components, as these are the elements that comprise the Software Functional Architecture. We then apply software engineering best practices to the specification, measurement, and analysis of the set of Blocks that represent the “modular components of software” in the System.

Software Architecture

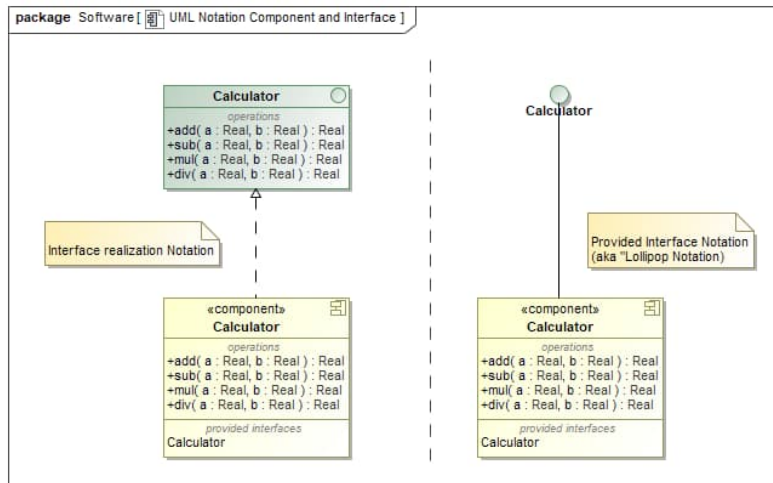
Software Architecture is an organization of software components that comprise a software system. The modeling elements required to produce a Software Architecture are: Components, Interfaces, and Data Types.

As defined in UML 2.5.1, these architectural elements represent “a set of constructs that can be used to define software systems of arbitrary size and complexity.”

“[A] Component [specifies] a modular unit with well-defined Interfaces that is replaceable within its environment. The Component concept addresses the area of component-based development and component-based system structuring, where

a Component is modeled throughout the development life cycle and successively refined into deployment and run-time.” – UML 2.5.1

The following illustration depicts a UML Component and its provided Interface, using two common notations.



From the software engineering perspective, the Component specifies what is what is built, tested, and deployed. This represents the Software Boundary, from the Measurement Perspective. The Interface specifies what the component does, in terms of Operations.

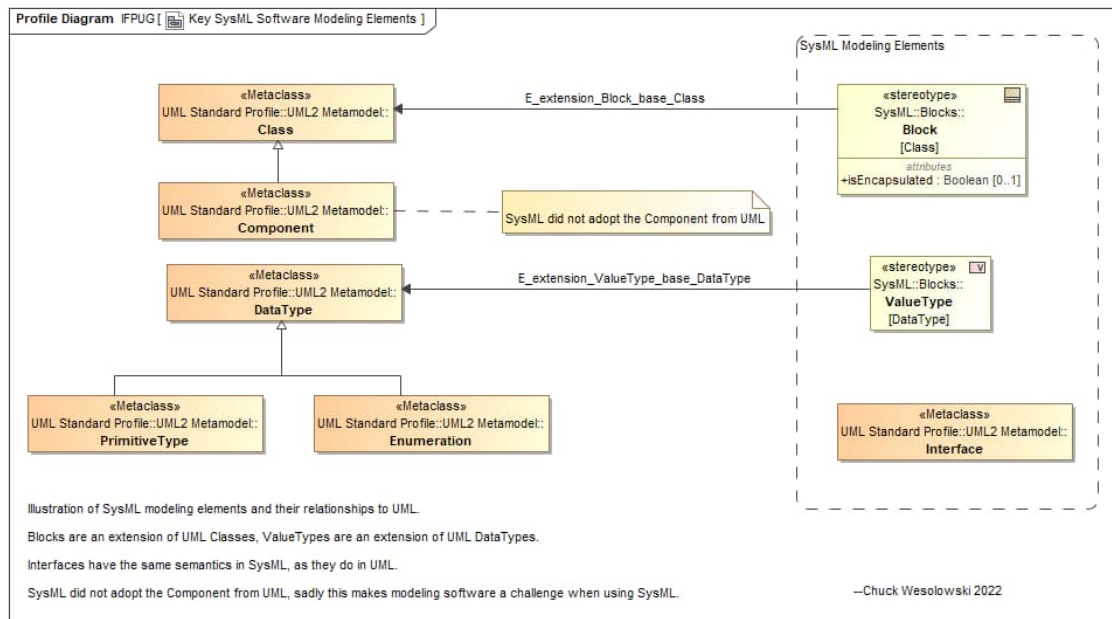
The Component Operations represent the implementation of the Interface Operations. There must be a Component Operation that corresponds with each Interface Operation.

In summary,

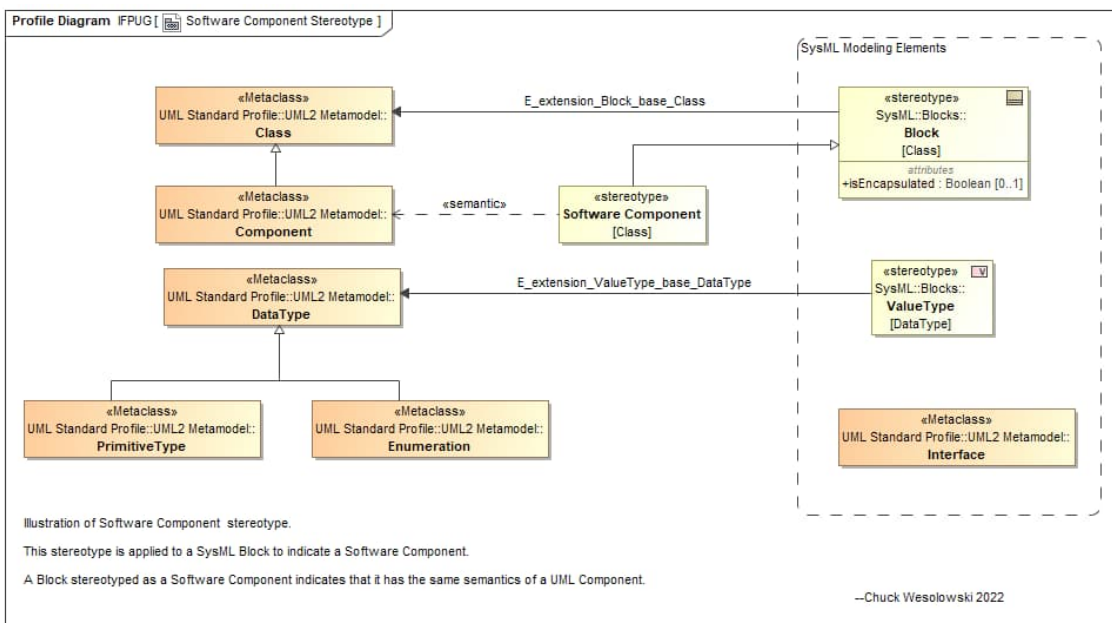
- Software Components and Interfaces are well-defined UML modeling elements that are used to construct “Software Component Architectures.”
 - Software Component Architectures are the result of a Software Engineering best-practice for specifying complex software systems.
 - Software Component Architectures have the following Characteristics:
 - Modular
 - Flexibly deployed
 - High Cohesion
 - Reusable
 - Manageable
 - Measurable

Meta Model Analysis

Unfortunately SysML did not adopt the UML Component as a modeling element, as shown in the following illustration. Note that this expression is a view of the Cameo/MagicDraw Meta Model.

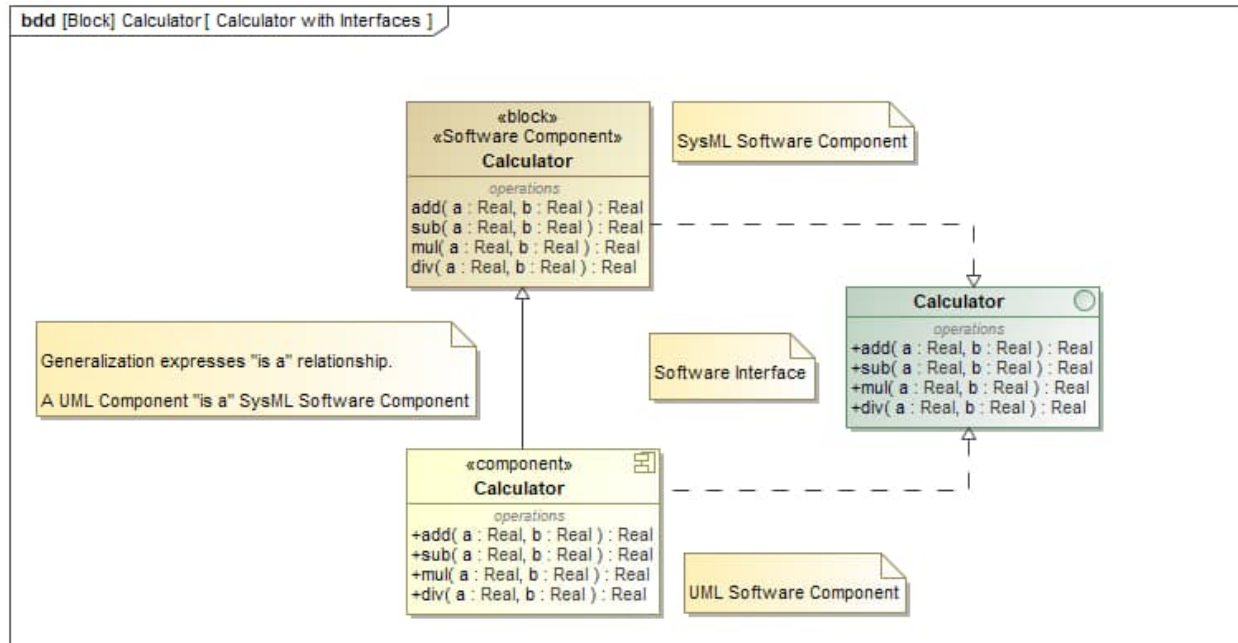


Consequently, the specification of Software Requirements and Architecture is sometimes unclear in SysML models. This situation is easily cured by the introduction of the Software Component stereotype. When applied to a SysML Block, this stereotype gives the semantics of a UML Component to the Block.



Software Component Stereotype

The Software Component stereotype provides the way to distinguish between software, and the “hardware, and human elements,” that a SysML Block can represent in the “trees of modular components” that comprise a system. Furthermore, because a Software Component Block is semantically identical to a UML Component, software engineering methods and best-practices developed for UML Components can be applied to the SysML Block.



The stereotype provides a means to translate between the modeling languages. It adds a “word” to the SysML vocabulary that already has a precise meaning in the parent language of UML.

Notice that the Interface, which has the same meaning in both SysML and UML, is noted as a Software Interface. It is often called a “Functional Interface” as it captures the functionality available to the User. It is not to be confused with a SysML Interface Block, which is often called an “Interface” in SysML parlance.

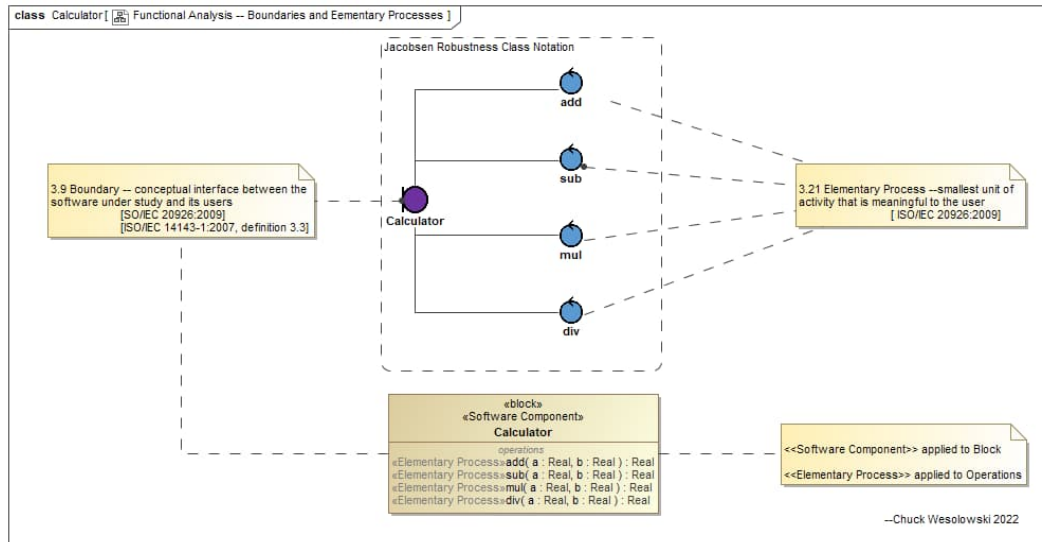
Software Functional Analysis and Architecture

Functional Analysis of software systems is an iterative process that specifies the required behavior of software components. The set of required behaviors and data required to support the behavior, constitutes the software functional requirements of the software component.

Software Functional Analysis preceded both UML and SysML, and has been practiced for decades, using many methods and techniques for documenting the resultant software functional architecture. The purpose of Software Functional Analysis is not only to specify quality software functional requirements, but also to measure these requirements.

Fortunately, UML and SysML provide a simple way to express the essential elements of software functional architectures, using surprisingly few modeling elements that clearly relate to the vocabulary of Software Functional Analysts and Measurement Specialists.

Software Functional Analysis involves the identification of a Boundary that encapsulates the Elementary Processes of the software. The following diagram illustrates these ideas using Jacobsen Robustness Classes, to indicate the Boundary and the Elementary Processes, as well as a fully specified SysML Software Component.



Note that the Operations of a SysML Software Component are stereotyped as Elementary Processes.

An Elementary Process represents a functional requirement. It is analyzed and measured to yield its Functional Size in Function Points (FP). This Measurement and Analysis is conducted from a “Black Box,” or logical view of the Boundary, without regard to Non-Functional, or Technical Requirements.

Quick Guide to Software Requirements

Functional Requirement

- What does it do? (name, description)
- What does it need? (inputs , memory references)
- What does it produce? (outputs, memory updates)
- What can go wrong? (errors)

Non-Functional Requirement

- How many ...?
- How much ...?
- How fast ...?
- How often ...?
- When ...?

The SysML Operation captures the inputs and outputs of the required behavior as “parameters,” and the possible error conditions as “raised exceptions.” These are data types that “cross the boundary” between the user and the software. These data types form the “Transactional Data Model,” as they form the set of Data Types exchanged via the Component’s Interface.

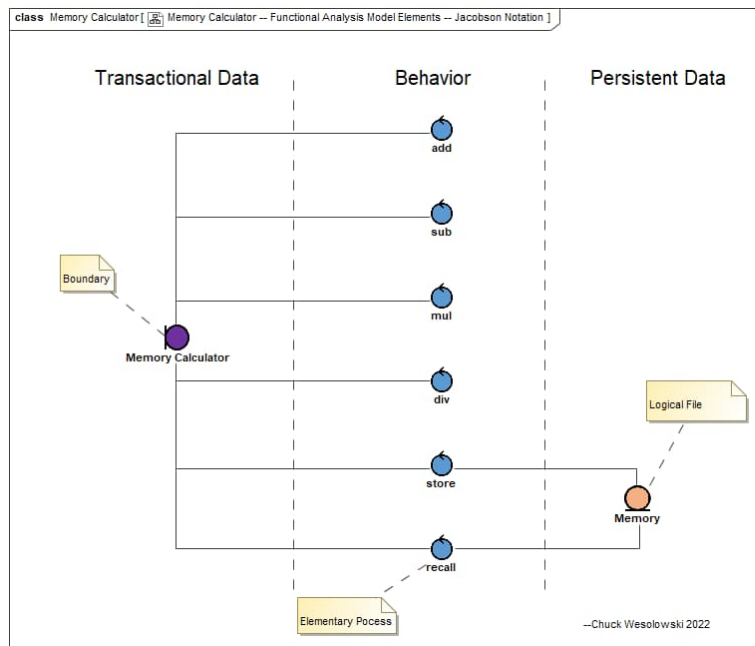
We turn our attention now to two specific questions from our Quick Review of Software Functional Requirements.

What does it need? (inputs , memory references)
What does it produce? (outputs, memory updates)

We know that inputs and outputs are contained in the operation signature, but what about memory references and updates?

Software Functional Analysts refer to “memory references and updates” as “Persistent Data,” meaning any data read or written from any type “storage” during the execution of an Elementary Process. Data “persists” from the analyst’s view in “Logical Files.” All Logical Files are forms of Persistent Data.

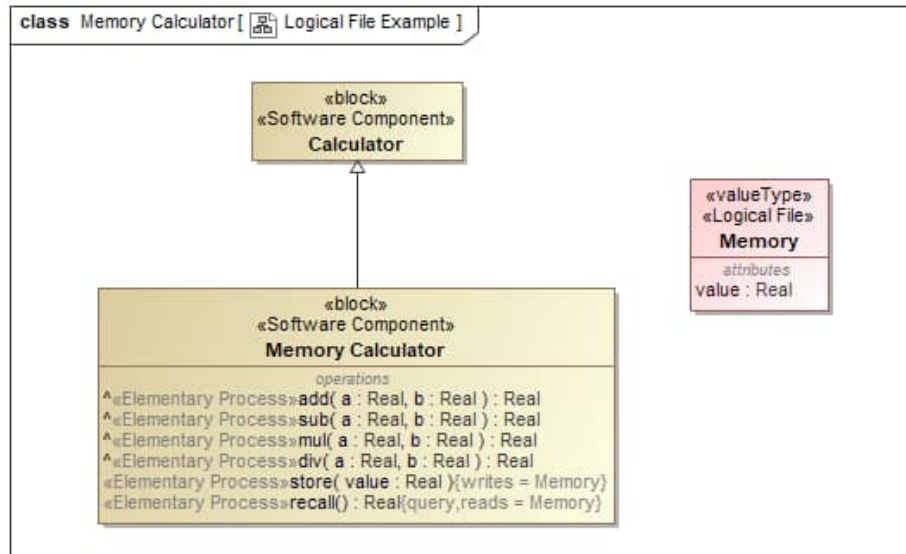
The following illustration depicts a Calculator with Memory.



A Logical File represents user relevant data, irrespective of technical, or other non-functional considerations. It matters not if the Logical File represents RAM in an embedded system, or relational database tables in an enterprise system.

A rule of thumb is: If the user can query it, and it can change from one query to the next, it is “Persistent Data.”

The SysML Specification of the Memory Calculator Software Component is indicated below.



Note first, that Memory Calculator inherits four Elementary Processes from the Calculator Component. These are indicated with a “^” preceding the Operation. Notice too, the properties displayed for the `store()` and `recall()` operations. The “reads” and “writes” properties are extensions provided by the Elementary Process Stereotype; “query” is a standard property of a SysML Operation.

IFPUG4SysML Profile

The IFPUG4SysML Profile supports Software Functional Measurement and Analysis by providing Stereotypes that identify key software elements in a SysML model.

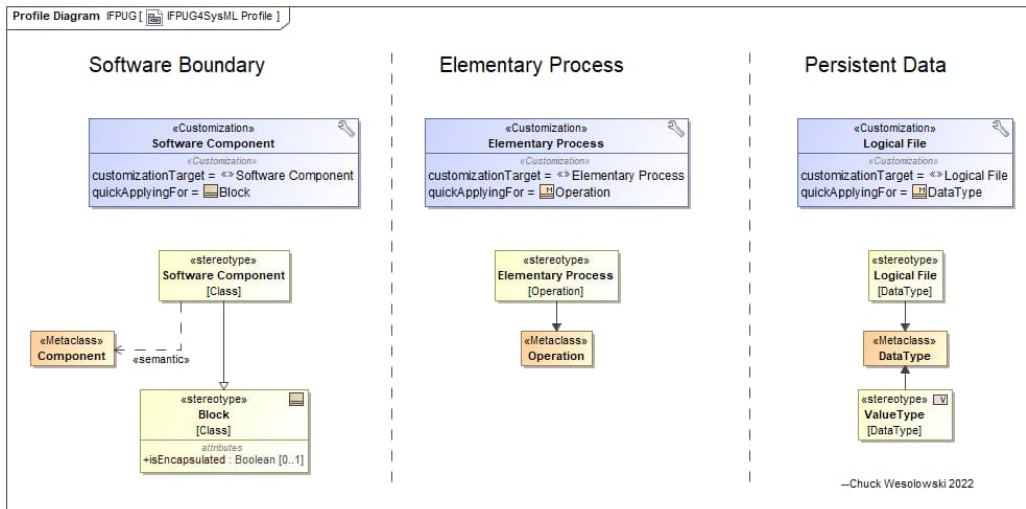
Functional Analysis Element	SysML Model Element	SysML Stereotype
Boundary	Block	<<Software Component>>
Elementary Process	Operation	<<Elementary Process>>
Logical File	ValueType	<<Logical File>>

SysML model elements that bear these stereotypes are subject to a set of validation rules for the setting of certain element properties.

The profile supports UI Customizations to facilitate the application of the Stereotypes to the appropriate SysML Elements using the “quickApplyingFor” feature, indicated in the following Profile Diagram.

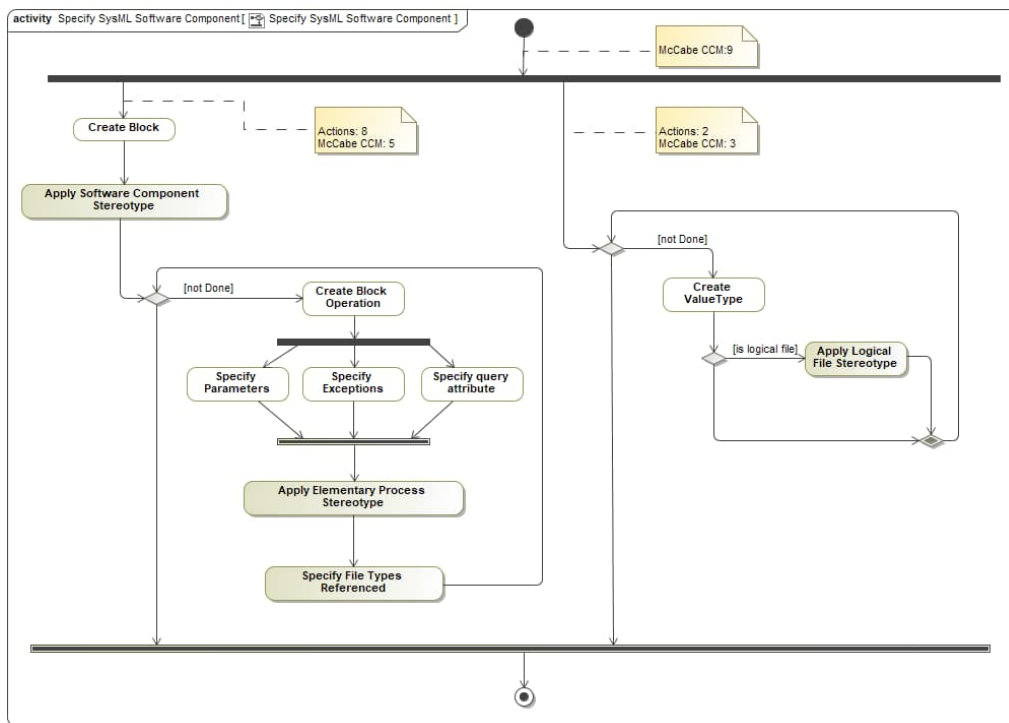
This means that the User can right-click on the modeling element in the Containment Tree or a Diagram to control the application of the stereotype.

The Quick Apply Feature is supported in Cameo/MagicDraw Modeling Tool.



Process for Specifying a Software Component

The following Activity Diagram describes the Systems Engineering Process for specifying a Software Component in SysML.



All Software Component Operations shall bear the Elementary Process stereotype. All File Types Referenced by the “reads” or “writes” properties of an Elementary Process shall bear the Logical File stereotype. Model validation rules enforce these requirements.

Software Functional Size

Software Functional Size is a way of quantifying the results of the Software Functional Analysis process. Each Software Component is measured independently for Functional Size.

A “Function Point Count,” is the common name for executing the IFPUG Standard Functional Size Measurement Method on a set of Software Functional Requirements and reporting the results.

Functional Sizing is quite simple provided that the required information is present. SysML provides an ideal mechanism for organizing Software Functional Requirements in a manner that facilitates Functional Sizing. The precision of specification inherent in the use of a formal modeling language enables us to use Cameo/MagicDraw Validation Rules to issue warnings and errors, when violations in model construction, or IFPUG counting rules are detected.

The following Table shows all SysML modeling elements required to conduct a Function Point Count of a Software Component.

Criteria

Element Type: ... Scope (optional): {} ... Filter:

Basis of Functional Measurement -- SysML Modeling Elements:

#	△ Owner	Name	Parameters	Errors	Reads	Writes	Documentation
1	Calculator	◇ add	◇ in a : Real ◇ in b : Real ◇ return : Real	Overflow			Perform the arithmetic addition (a+b) and return the result.
2	Calculator	◇ div	◇ in a : Real ◇ in b : Real ◇ return : Real	Overflow DivideByZero			Perform the arithmetic division (a/b) and return the result. Catch Divide By Zero
3	Calculator	◇ mul	◇ in a : Real ◇ in b : Real ◇ return : Real	Overflow			Perform the arithmetic multiplication (a*b) and return the result.
4	Calculator	◇ sub	◇ in a : Real ◇ in b : Real ◇ return : Real	Overflow			Perform the arithmetic subtraction (a-b) and return the result.
5	Memory Calculator	◇ recall	◇ return value : Real	MemoryError	Memory		Return the value of Memory
6	Memory Calculator	◇ store	◇ in value : Real	MemoryError		Memory	Store the input value in Memory

The Scope of the Count includes the Calculator and Memory Calculator Software Components. Note that the Key for the Element Type in the Table criteria is Elementary Process.

The IFPUG4SysML profile includes software to automatically count and report the Functional Size of a software boundary indicated by a SysML Software Component Block.

The software constructs a Measurement Model in the Software Component's Containment Tree. The Measurement Model is transient, and may be regenerated on-demand. The purpose of the model is to present a perspective that is especially useful in managing the life-cycle of Software Components in terms of their Functional Requirements.

The measurement model is composed of Jacobson Robustness Classes stereotyped to indicate the Software Boundary, Elementary Processes, and Logical Files. These are called the Base Functional Components (BFC) of the Measurement Model.

The following Table View is available as part of the IFPUG4SysML Profile. It contains the results of the Functional Size Measurement of the Software Boundaries specified in the Scope.

Criteria

Element Type:

...





















Scope (optional):

{Joy}

...

Filt

IFPUG Functional Size Report:

#	△ Owner	Name	BFC	FP	TFC	DFC	DET	FTR	RET
1	 Calculator	  Calculator	Boundary	16	16	0			
6	 Memory Calculator	  Memory Calculator	Boundary	29	22	7			
7	 Memory Calculator	 Memory	ILF	7			1		1
8	 Memory Calculator	 store	EI	3			3	1	
9	 Memory Calculator	 recall	EQ	3			3	1	
10	 Memory Calculator	 add	EO	4			5	0	
11	 Memory Calculator	 sub	EO	4			5	0	
12	 Memory Calculator	 mul	EO	4			5	0	
13	 Memory Calculator	 div	EO	4			5	0	

This view reports the Measurements and Metrics that represent the Functional Size of the Software. Results are reported throughout the software lifecycle beginning with Requirements Analysis. Changes in Requirements are reflected in both the measurements, and the Functional Size.

The Function Point Metric

Function Points represent the Functional Size of the SysML Software Component. The Transactional Function Contribution (TFC), is the sum of the Function Points contributed by each Elementary Process. The Data Function Contribution (DFC) is the sum of the Function Points contributed by each Logical File.

The Functional Size is expressed in Function Points (FP). Function Points are a Metric, meaning that they are derived from measurements. Each BFC in the Measurement Model has a Functional Size expressed in FP, however only the Functional Size of Elementary Processes and Logical Files are derived from measurements.

These measurements are made by counting characteristics of the Functional Architecture. These include the numbers of Data Element Types, references to Logical Files, and Record Element Types, indicated in the table as DET, FTR, and RET respectively.

DET counts apply to both Elementary Processes and Logical Files. FTR is unique to Elementary Processes, and RET is unique to Logical Files. The details of the process are documented in the Appendices.

Summary

Software Component Architectures are produced as a result of the Requirements Development Engineering process. This paper described a method for constructing Software Components in SysML.

Blocks that represent software in a SysML model are clearly indicated using the Software Component stereotype. This stereotype indicates that the SysML Block represents a UML Component in the model.

Each Block Operation on a Software Component is stereotyped as an Elementary Process. This stereotype enables the analyst to specify read and write references to structured Value Types stereotyped as Logical Files.

The Functional Size of a Software Component is reported from a Measurement Model of the Software Component that is generated on-demand within the modeling tool (Cameo). The profile includes pre-defined tables for reviewing the results.

Appendix A.

The following BPMN Diagram describes the process of a Software Systems Engineer specifying a Software Component using the SW4SysML Profile in Cameo.

