**Narrowing the Gap Between System and Software Engineering by Integrating Computations into Object-Process Methodology**

Natali Levi

**Narrowing the Gap Between System and Software Engineering by Integrating Computations into Object-Process Methodology**

Research Thesis

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Natali Levi

Submitted to the Senate of  
the Technion - Israel Institute of Technology

|  |  |  |
| --- | --- | --- |
| HAIFA | Shvat, 5781 | January 2021 |
|  |  |  |

The research thesis was done at the Faculty of Industrial Engineering and Management under the supervision of Professor Dov Dori and Dr. Ahmad Jbara.

The generous financial support of the following organizations is gratefully acknowledged:

Technion – Israel Institute of Technology

Bernard M. Gordon Center for Systems Engineering at the Technion – Israel Institute of Technology

**Publications:**

1. Dov Dori, Ahmad Jbara, Natali Levi, and Niva Wengrowicz, Object-Process Methodology, OPM ISO 19450 – OPCloud and the Evolution of OPM Modeling Tools. *Systems Engineering Letters*, Project Performance International (PPI) SyEN 61, January 30, 2018. <https://www.ppi-int.com/wp-content/uploads/2018/01/SyEN_61.pdf>
2. Linwen Li, Natali Levi Soskin, Ahmad Jbara, Moti Karpel, and Dov Dori, Model-Based Systems Engineering for Aircraft Design with Dynamic Landing Constraints Using Object-Process Methodology. *IEEE Access*, pp. 61494-61511, 2019. Open Access: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8710233>
3. Ahmad Jbara, Arieh Bibliowicz, Niva Wengrowicz, Natali Levi, and Dov Dori, Toward Integrating Systems Engineering with Software Engineering through Object-Process Programming. *International Journal of Information Technology* – *SN Computer Science*, May 2020. Open Access: [https://rdcu.be/b5rB5](https://eur01.safelinks.protection.outlook.com/?url=https%3A%2F%2Frdcu.be%2Fb5rB5&data=02%7C01%7Cdori%40technion.ac.il%7C830396261af5438fad1408d8215623b0%7Cf1502c4cee2e411c9715c855f6753b84%7C1%7C0%7C637296001132137260&sdata=xKt%2FI9bqVQCCgBHwT%2BXk2cwGE5ymxK7KHVYUOSdm0dM%3D&reserved=0)
4. Natali Levi-Soskin, Ron Shaoul, Hanan Kohen, Ahmad Jbara, and Dov Dori, Model-Based Diagnosis with FTTell: Assessing the Potential for Pediatric Failure to Thrive (FTT) During the Perinatal Stage. In: Wrycza S, Maślankowski J, eds. *Information Systems: Research, Development, Applications, Education*. Cham: Springer International Publishing; 2019:37-47.
5. Natali Levi Soskin, Ahmad Jbara, and Dov Dori, The Model Fidelity Hierarchy: From Text to Conceptual, Computational, and Executable Model. IEEE Systems Journal, 2020. *IEEE Systems Journal*, 15(1), pp. 1287-1298, March 2021.[DOI: 10.1109/JSYST.2020.3008857](https://doi.org/10.3390/app10217417)

Acknowledgement

Throughout the writing of this dissertation, I have received a great support and assistance.

I would first like to thank my supervisors, Professor Dov Dori and Dr. Ahmad Jbara, whose expertise was invaluable, starting with formulating the research questions and methodology, through instructing me in individualistic thinking and developing my way as a researcher, ending with writing this Ph.D. work. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. I am deeply grateful to you for your time, knowledge sharing, guiding and support.

I would like to extend my sincere thanks to colleagues from our ESML lab for their wonderful collaboration. Thank you for your insightful comments and suggestions, support, and friendship.

In addition, I would like to thank my parents for their endless, unconditional support and helping me gaining the required time and optimal environment for performing this research.

Finally, I would like to express my gratitude to my family. My husband Roee, who supported me through this meaningful period and my beloved children – Daniel, Yehonatan and Romi that hugged, kissed and raised a smile on my face in every hard moment. Having you makes everything possible.

Table of Contents

[List of Figures 6](#_Toc62570785)

[Acronym Glossary 8](#_Toc62570786)

[Abstract 1](#_Toc62570787)

[**1** **Introduction** 2](#_Toc62570788)

[**1.1** **Model-Based Systems Engineering** 2](#_Toc62570789)

[**1.2** **OPM ISO 19450** 2](#_Toc62570790)

[**1.3** **The System-Software Engineering Gap** 5](#_Toc62570791)

[**1.4** **Research Goal** 6](#_Toc62570792)

[**1.5** **OPCloud** 7](#_Toc62570793)

[**1.6** **Model Fidelity Hierarchy** 8](#_Toc62570794)

[**1.7** **Usage and Applicability Examples** 8](#_Toc62570795)

[**1.7.1** **The Braking System Model** 8](#_Toc62570796)

[**1.7.2** **The Landing Gear Model** 9](#_Toc62570797)

[**1.8** **Diagnostic Models: A Third Kind added to Descriptive Prescriptive Models** 9](#_Toc62570798)

[**1.8.1** **FTTell – Failure To Thrive Diagnostic Model** 10](#_Toc62570799)

[**1.9** **Contribution** 10](#_Toc62570800)

[**1.10** **Thesis Outline** 11](#_Toc62570801)

[**2** **Background and Literature Review** 12](#_Toc62570802)

[**2.1** **The System–Software Gap** 12](#_Toc62570803)

[**2.2** **OMG Attempts at Computational Executable Modeling** 13](#_Toc62570804)

[**2.3** **Different Modeling Approaches** 13](#_Toc62570805)

[**2.4** **OPM Compared with Other Modeling Approaches** 14](#_Toc62570806)

[**2.5** **Medical Knowledge Representation** 15](#_Toc62570807)

[**3** **MAXIM—Methodical Approach to Executable Integrated Modeling** 16](#_Toc62570808)

[**3.1** **Computational Objects** 16](#_Toc62570809)

[**3.2** **Computational processes** 17](#_Toc62570810)

[**3.3** **Computational States** 19](#_Toc62570811)

[**3.4** **Procedural Links** 20](#_Toc62570812)

[**3.5** **Structural Links** 21](#_Toc62570813)

[**4** **OPCloud** 25](#_Toc62570814)

[**4.1** **Computational Object** 25](#_Toc62570815)

[**4.2** **Computational Process** 27](#_Toc62570816)

[**4.2.1** **Predefined functions** 28](#_Toc62570817)

[**4.2.2** **User-defined** 29](#_Toc62570818)

[**4.2.3** **External functions** 29](#_Toc62570819)

[**4.3** **Model Execution in OPCloud** 30](#_Toc62570820)

[**4.3.1** **A single execution** 30](#_Toc62570821)

[**4.3.2** **Stop execution** 33](#_Toc62570822)

[**4.3.3** **Repeated execution** 34](#_Toc62570823)

[**4.3.4** **Export results** 34](#_Toc62570824)

[**4.3.5** **Import values** 35](#_Toc62570825)

[**5** **Industrial Cooperation: A Running Case Study from The Aircraft Industry** 36](#_Toc62570826)

[**5.1** **OPM Conceptual Model** 36](#_Toc62570827)

[**5.2** **Adding Computational Elements** 38](#_Toc62570828)

[**5.3** **Model-Based Computation Demonstration** 44](#_Toc62570829)

[**6** **Model Fidelity Hierarchy – The Landing Gear Case Study** 46](#_Toc62570830)

[**6.1** **First Fidelity Hierarchy Level: An OPM Conceptual Model** 46](#_Toc62570831)

[**6.2** **Second Fidelity Hierarchy Level: An OPM Computational Model Extension** 53](#_Toc62570832)

[**6.2.1** **Static load and load rating calculation** 54](#_Toc62570833)

[**6.2.2** **Tire parameters defining** 56](#_Toc62570834)

[**6.3** **Third Fidelity Hierarchy Level: OPM Computational Model Execution** 57](#_Toc62570835)

[**7** **Diagnostic Model – FTT Case Study** 60](#_Toc62570836)

[**7.1** **FTTell Perinatal Stage Evaluation and Results** 69](#_Toc62570837)

[**7.2** **FTTell Postnatal Stage Evaluation and Results** 69](#_Toc62570838)

[**8** **SUMMARY** 76](#_Toc62570839)

[**9** **CONCLUSION** 80](#_Toc62570840)

[**10** **FUTURE WORK** 82](#_Toc62570841)

[**Bibliography** 84](#_Toc62570842)

[תקציר iii](#_Toc62570843)

List of Figures

Figure 1: Example of an OPM object (left) and process (right) 2

Figure 2: Example of an OPM objects and processes having different essence 2

Figure 3:Example of an OPM objects and processes having different affilliation 2

Figure 4: An image of OPCloud 7

Figure 5: A **Car** object with a **Weight** attribute having a specific value **3500**, Measurement Unit **[kg]** and as alias **{w}** 17

Figure 6: Process specializes into Physical Process and Informatical Process and the latter specializes into Computable and Non-Computable Process 18

Figure 7: Exemplifying the **Computability** property of a **Process**. Left: **Walking** is a physical, **non-computable** process. Center: **Thinking** is an informatical, **non-computable** process. Right: **Word Counting ()** is a **computable** process 18

Figure 8: **Time to city center** is a **computable** process wich gets two input values from two objects and writes the result into the output object **TimeTo City Center** 19

Figure 9: object **Bank** with situational states **open** and **closed** (left); object **Velocity** with computational state having the value **60** (right) 19

Figure 10: Difference between opl sentences that describe **Bank** object with situational states and **Velocity** object with a computational state. 20

Figure 11: different usages for procedural links 20

Figure 12: opl with marked sentences for instrument, comsumption and result links 21

Figure 13: effect link affects the value of the object **Parameter** 21

Figure 14: OPD of model-specific type for **Aircraft** 22

Figure 15: model-specific type for **Aircraft** with features (attributes and operation) 23

Figure 16: model-specific type for **Aircraft** with object and process specializations added 23

Figure 17: Creating a new object 25

Figure 18: Naming the object **Number** 25

Figure 19: Object-related operations in the toolbar 25

Figure 20:Object-related operations in the halo next to the object 26

Figure 21: Popup window for value inserting 26

Figure 22: Popup window for units inserting 26

Figure 23: Popup window for alias inserting 27

Figure 24: Toolbar and halo for a computational object 27

Figure 25: Creating a new process 27

Figure 26: Naming the process **Adding** 28

Figure 27: Object-related operations in the toolbar 28

Figure 28:Object-related operations in a frame next to the object 28

Figure 29: A menu for selecting the type of computational functionality for the process 28

Figure 30: Selecting a basic arithmetic operation for a process 29

Figure 31: Popup window for User Defined function 29

Figure 32: External function definition 30

Figure 33: "Execute" sigh circled by a solid black line 30

Figure 34: Execution menu 30

Figure 35: Input and output with same **kg** units 31

Figure 36: **Fruits Set** units changed from **g** to **kg** automatically because the units of the **Weight** of **Apples Set** and the **Weight** of **Bananas Set** are **kg** 31

Figure 37: **Weight** of **Apples Set** is in **g** while **Weights** of **Bananas Set** and **Fruits Set** are in **kg** 32

Figure 38: Popup for choosing the units that the user would like to use 32

Figure 39: Execution with **g** units 32

Figure 40: Execution with **kg** units 32

Figure 41: Executing **Multuplying()** process four times 34

Figure 42: Export file example 35

Figure 43: place to choose the number n which means that every n runs a csv file exported 35

Figure 44: **Aircraft Braking** – the main process of the system’s function 36

Figure 45: An Agent Link from the object **Pilot** to the process **Aircraft Braking** 36

Figure 46: An Instrument Link fr om the object **Aircraft Braking System** to the process **Aircraft Braking** 37

Figure 47: An Effect Link from **Aircraft Braking** to **Aircraft** 37

Figure 48: SD – the top-level OPD of the **Aircraft Braking System** 37

Figure 49: SD1 - **Aircraft Braking** in-zoomed. A Result Link from the **Commanding** process to the **Command** object indicates that **Command** is created in one of its four states. In this execution, the state is **pedal braking**, as shown by the circle along the link from that state to the **Pedal Braking** process. 38

Figure 50: SD1.1 – **Pedal Braking** in-zoomed, showing three engineering domains in the same model and same diagram: **Pedal Pressing** is in the mechanical engineering and human factors engineering domain, **Braking Force Applying** is in the mechanical engineering domain, and **Decelerating** is in the computational domain. 40

Figure 51: **Decelerating** with a tooltip showing its computational function **V=V₀-F/m\*t** and the informatical objects **Mass**, **Time**, and **Braking Force** with their units, used for the **Speed** calculation. 40

Figure 52: SD1.1.1 - **Braking Force Applying** in-zoom showing two conceptual processes and one that will be in-zoomed and calculate the **Braking Force** 41

Figure 53: SD 1.1.1.1 - **Braking Force Calculating** is a computational process with input **Press Force** and **Press Angle** in degrees, and output **Braking Force** in **Newton** 41

Figure 54: SD1.1.1.1.1 - **Signal Converting** in-zoomed 42

Figure 55:SD1.1.1.1.2 - **Signal Processing** in-zoomed 42

Figure 56: SD1.1.1.1.3 - **Wheel Lock Detecting** in-zoomed 43

Figure 57: SD1.1.1.1.4 -**Pulse Set Generating** in-zoomed 43

Figure 58: SD1.1 - After 5 s of **Decelerating**, the value of **Speed** is updated from **500.00 m**/s in Figure 37 to **497.42** m/s 45

Figure 59. Integrating two OPM models into a single model 46

Figure 60: SD1 - **Aircraft Defining** in-zoomed 47

Figure 61:SD2 - **System Defining** in-zoomed 47

Figure 62:SD3 - **Item Defining** in-zoomed 48

Figure 63: SD4 - **Item Implementing** in-zoomed 49

Figure 64: SD5 - **Item Realizing** in-zoomed 49

Figure 65:SD6 - **System Realizing** in-zoomed 50

Figure 66: SD7 - **Aircraft Realizing** in-zoomed 50

Figure 67. OPL sentences describing the initial and final states of **Aircraft** 50

Figure 68: SD8 - **Cargo Transporting** in-zoomed 52

Figure 69: SD8.1 - **Aircraft Landing** in-zoomed 52

Figure 70. The **Landing Shocks Absorbing** process in-zoomed 53

Figure 71. The fixed part of Figure 69, where we added **Main Landing Gear** and **Nose Landing Gear** as specilizations of **Landing Gear**,and refined the links 55

Figure 72. **Landing Gear Parameters Defining** in-zoom for calculating the **Static Load** for each landing gear 56

Figure 73. **Tires Selecting** in-zoom for calculating the required **Load Rating** for tires 56

Figure 74. **Tire Parameters Defining** in-zoomed 57

Figure 75. The value of **Main Static Load {Fsm}** is **38340** but the value of **Nose Static Load {Fsn}** is **0** 58

Figure 76. calculated parameters of the **Nose** and **Main Landing Gear** and **Tire** 58

Figure 77. calculated parameters of the **Nose** and **Main Landing Gear** and **Tire** 59

Figure 78: SD of the FTTell system, showing the main process which is called **Failure To Thrive (FTT) Diagnosing & Treating**, the involved object set that serve as agents, and resulting objects. One of the three outputs of this process is the object **Diagnosis** (at the center left), which consist of three parts, each for a different developmental period. 60

Figure 79: Main process **FTT Diagnosing & Treating** in-zoomed(left) and its **Diagnosing** subprocess in-zoomed further (right) 62

Figure 80: SD1.1.1 - **Perinatal Growth Examining** in-zoomed 62

Figure 81: **Postnatal Growth Examining** in-zoomed 64

Figure 82: **Postnatal Growth Examining** in-zoomed – with real values 65

Figure 83: **Weight Gain Deceleration Analyzing** process in-zoomed 66

Figure 84. Model Fidelity Hierarchy diagram 81

Acronym Glossary

|  |  |
| --- | --- |
| CPG | Cyber-Physical Gap |
| CPS | Cyber-Physical Systems |
| FTT | Failure To Thrive |
| fUML | Foundational UML |
| INCOSE | The International Council on Systems Engineering |
| ISO | International Organization for Standardization |
| MAXIM | Methodical Approach to Executable Integrative Modeling |
| MDA | Model-Driven Architecture |
| MFH | Model Fidelity Hierarchy |
| OMG | Object Management Group |
| OOSEM | Object-Oriented System Engineering Method |
| OPD | Object-Process Diagram |
| OPL | Object-Process Language |
| OPM | Object Process Methodology |
| SysML | Systems Modeling Language |
| UML | Unified Modeling Language |
| xUML | Executable UML |
| DFS | Depth-First Search |

Abstract

Lack of a modeling framework that integrates systems with software engineering is a major cause of product development problems. The current model-based systems engineering approach applies a variety of model kinds, each with its own fidelity level, with disparate or loosely integrated models. The gap between the system model and the software levels leads to a great fidelity gap, as the software is often not aligned with the model of the system it is supposed to control. This doctoral thesis aims to overcome this system-software modeling gap by integrating computational, software-related, and model execution capabilities into Object Process Methodology – OPM ISO 19450-based conceptual modeling, resulting in a holistic unified executable qualitative-quantitative modeling framework. The gap is bridged by extending OPM with a Methodical Approach to Executable Integrative Modeling (MAXIM). We exemplify MAXIM usage by a model of a civil aircraft landing gear braking system, showing that engineers from different domains can collaborate during early system development phases to jointly construct a holistic model, combining qualitative and quantitative aspects. During the research, we came across the model fidelity hierarchy, which is presented via a case study of modeling an aircraft landing gear. OPM with MAXIM enables continuous, seamless modeling approach with increasing accuracy. As errors are revealed during early system lifecycle stage, they are exponentially less costly to correct than those revealed downstream. The principles of MAXIM are presented and demonstrated within OPCloud—a web-based collaborative conceptual OPM modeling framework.

Models have traditionally been mostly either prescriptive, expressing the function and structure of a system-to-be, or descriptive, specifying a system so it can be understood and analyzed. MAXIM provides a new methodology for developing and engaging with a new, third family of models—diagnostic models. As a case in point, we have built a model for assessing potential pediatric failure to thrive (FTT). As part of this Thesis, we present FTTell—an executable model-based medical knowledge aggregation and diagnosis tool, in which the qualitative considerations and quantitative parameters of the problem are modeled using MAXIM. The efficacy of the tool is demonstrated on data collected from 100 children, providing 87% correct diagnosis. Pediatricians can use this model-based standardized approach to improve their FTT diagnosis for appropriate timely intervention.

The development of MAXIM as a major extension of OPM, which promotes it from a conceptual modeling language and methodology to a combined conceptual-computational modeling approach, lays out a solid foundation for extending it further to support full-fledged simulation and execution of systems of all kinds. Moreover, the ability to inject code into model processes will potentially allow smoother reverse engineering. In case we do not know how to make the translation between code and graphical representation we can retain the code as is and simply inject it into the process with slight adaptations. Another direction is teaching programming for novices by starting with visual programming and progressively moving to traditional coding. MAXIM provides us with an approach that combines traditional coding with visual programming, benefiting from both modalities.

# **Introduction**

## **Model-Based Systems Engineering**

The basic tenet of model-based systems engineering (MBSE) is that the conceptual model of the system-to-be is expressed as early as possible, ideally from the requirements stage, at which a solution-neutral model of the problem at hand can be constructed. MBSE is often defined as the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities, advocates that formal models are the authoritative set of artifacts that anchor product and system development processes and evolve to reflect the evolution of the design.

The formal model evolves along the system lifecycle stages, and each project-related discipline continues the system design using its domain-specific language and toolset, while the systems engineer keeps the holistic system model updated.

## **OPM ISO 19450**

OPM [1], [2], [3] – ISO 19450:2015 is a systems modeling language and methodology that represents the function, structure, and behavior of any system using a minimal universal (upper) ontology with only two kinds of *things*, shown in Figure 1: *stateful objects*—things that exist, possibly with states, and *processes*—things that transform objects by creating or consuming them, or by changing their state.



Figure 1: Example of an OPM object (left) and process (right)

Any OPM thing has an attribute called Essence whose values, as shown in Figure 2, can be informatical (not shaded) or physical (shaded). In addition, it has Affiliation, which can be systemic (solid border) or environmental (dashed border), as shown in Figure 3.

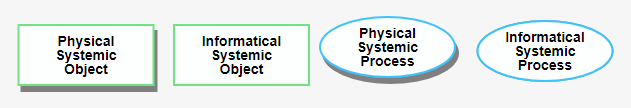


Figure 2: Example of an OPM objects and processes having different essence

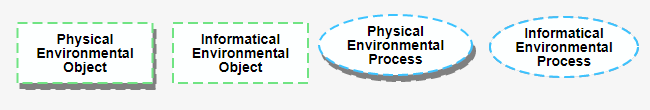


Figure 3:Example of an OPM objects and processes having different affilliation

OPM things are connected by links, which graphically express relations. There are two kinds of links: procedural (Table 1) and structural (Table 2). Any OPM model consists of two parts, which express the same set of model facts in two modalities: (1) the graphical part – the OPD set: a hierarchically-organized set of one or more Object Process Diagrams (OPDs) and, (2) the textual part – the OPL spec: a collection of sentences in a subset of English called Object Process Language (OPL). The OPD set is a set of OPDs related to each other in a tree structure, such that each diagram lower in the hierarchy refines its ancestor, usually by zooming into one of the processes in the ancestor OPD. OPD is the only kind of diagram of an OPM model. It can contain things (objects and processes), with links connecting them to express structural and procedural relation. OPL is the counterpart textual representation of the OPD set. Each OPD construct—two or more things connected by one or more links—is reﬂected textually in one or more OPL sentences. This bimodal representation caters to the dual channel assumption [4], [5], [6]. An OPM model can be presented at various levels of detail in different, interconnected views, each being an OPD. The top-level OPD is called System Diagram (SD). SD usually consists of one systemic process and its operand – the object which that process transforms. Together, the process and the operand are the function of the system. SD also includes the beneficiary group – the person or people benefiting from the system’s function, and enablers – agents (humans) and instruments (non-humans). Enablers are linked to the main process and enable it, but unlike transformees (input and output objects), they are not transformed by it. Each OPD can be reﬁned to expose deeper levels of detail by one of the following three refinement–abstraction complexity management mechanisms: (1) In-zooming – out-zooming: In-zooming reﬁnes a thing, usually a process, to show subprocesses, of which the main process consists and their temporal (sequential or parallel) execution order. Out-zooming is the inverse of in-zooming—it reduces a set of things into a more abstract thing. (2) Unfolding – folding: Unfolding reﬁnes a thing, usually an object, to show parts (sub-objects), of which the main object consists, or features (attributes and operations), or specializations of that thing. Folding is the inverse—it abstracts an unfolded thing by hiding its parts, features, or specializations. (3) State expression – suppression: Each object can have one or more states, and each state can be expressed (shown) or suppressed (hidden), based on what the modeler would like to emphasize in the diagram.

Table 1: Procedural links in OPM

|  |  |  |
| --- | --- | --- |
| **Link name** | **Example** | **OPL** |
| Result |  | **Creating** yields **File.** |
| Consumption |  | **Eating** consumes **Food**. |
| Effect |  | **Pedal Pressing** affects **Speed.** |
| Agent |  | **Person** handles **Eating**. |
| Instrument |  | **Eating** requires **Fork.** |
| Invocation |  | **Test Finishing** invokes **Test Submitting.** |
| Instrument Condition |  | **Buying** ocuures if **Store** is at state **open**, otherwise **Buying** is skipped. |
| Instrument Event |  | **Whistle** initiates **Running**, which requires **whistle.** |

Table 2 : Structural links in OPM

|  |  |  |
| --- | --- | --- |
| **Link name** | **Example** | **OPL** |
| Aggregation-Participation |  | **Animal** consists of **Head** and **Tail**. |
| Generalization-Specialization |  | **Cat** is an **Animal**. |
| Classification-Instantiation |  | **My Cat** is an instance of **Cat**. |
| Exhibition-Participation |  | **Cat** exhibits **Color**. |
| Unidirectional Tagged |  | 4 **People** live in **House**. |
| Bidirectional Tagged |  | **Exam** contains **Question**.  and  **Question** appears in **Exam**. |

The entire OPM model of a complex system is expressed in a tree of OPDs, each created by zooming into or unfolding some process or object in its ancestor OPD. The first OPD, SD, is the root of the OPD tree, and it is the only OPD at detail level 0. This is the bird-eye’s view of the system, which provides a quick overview of the system’s function and benefit. Lower-level OPDs are denoted SDn1.n2. … nm, where nj, j=1,2.. m, is the number of the detail levels (layer number in the OPD tree) and m is the number of the refined (in-zoomed or unfolded) subprocess in the ancestor OPD.

There is a long-standing ontological debate and disagreement between two schools of philosophers about what exists, or what it means to persist: The perdurance theory (perdurantism) and endurance theory (endurantism). Perdurantists think that objects have both spatial and temporal parts, while endurantists think that they only have spatial parts. Perdurantists believe that things (e.g., animals, airplanes, stars, which are OPM objects) have ‘temporal parts’ in addition to spatial ones. Endurantists believe that things do not have temporal parts, but whenever they exist, they are present as a whole. This disagreement is resolved by the OPM definition of a thing as a generalization of object and process, with object being a thing that exists over time and can be inspected at points in time, and a process being a cognitive pattern which happens to an object over time and transforms it (e.g., by changing one of its attribute values), so it can be inspected only as long as the object which the process operates on—the transformee—undergoes transformation. Hence, OPM objects are endurants: They only have spatial parts, not temporal ones, while OPM processes are ‘kind of’ perdurant: They only have temporal parts, not spatial ones. We say ‘kind of’ because while a perdurant thing has both spatial and temporal parts, a process does not have any spatial parts; all its parts are temporal. In OPM, state is a situation or position of an object at some point in time. Hence, state mediates between space and time: An object inspected at a given point in time can be at exactly one of its states or in transition between states, and it takes a process to affect the object by changing its state. Effect, which is a change of an object state, is one of three kinds of object transformations, the other two being generation and consumption. If we assign to each object by default the two states “existent” and “non-existent”, then generation amounts to changing the state of that object from non-existent to existent, while consumption is the reverse.

## **The System-Software Engineering Gap**

Systems engineering and software engineering are sister, complementary disciplines, yet their evolution paths have been largely separate and disconnected, preventing integration between the hardware and software system aspects. Accurate modeling and design of the system-to-be is necessary for proper development and smooth interfacing [7]. While operating the system and executing the software program that controls it using real-life data, new insights are gained [8], [9] and problems in industrial product development are discovered [10], [11]. These insights often reveal that important issues are modeled incorrectly or not modeled at all, requiring significant rework, which delays the project’s successful completion.

The major problem arising from the current de-facto development process is that the transition from a system-level model to software and other disciplinary models, creates a “Grand Canyon” [12]—a gap that causes the various disciplinary engineers to lose the common big picture expressed in the original conceptual model, along with critical information related to design considerations associated with this model. Problems caused by misalignments between early and late design start to emerge, resulting in functionality and quality problems, project delays, and cost overruns. These phenomena are most acute for the system-software transition due to the difference between the “real” physical parts of the system and the software that is supposed to control it, which is informatical, intangible, “cybernetic”, and does not yield to the laws of physics.

## **Research Goal**

The primary goal of this doctoral research is to overcome the widening hardware-software modeling gap [10], [13], [14] by bringing systems engineering and software engineering closer together in a stepwise fashion [15]. To this end, we present MAXIM—a Methodical Approach to Executable Integrative Modeling. MAXIM enables integrating computational, software engineering capabilities into ISO 19450:2015 OPM [1], [2], extending OPM’s power as a model-based systems engineering methodology. The implemented integration of MAXIM is part of OPCloud[[1]](#footnote-1) [16], [17].

In MAXIM, the system model, which comprises both physical (human and hardware) and informatical (software) components, represents not only the structure and behavior of the system, which enable it to perform its intended function, but also its quantitative requirements and computational, software-driven aspects. This combined humans-hardware-software model constitutes a complete and accurate executable speciﬁcation of the system from its top-level abstract view, which expresses the system’s main function, beneficiaries, and value, all the way to a set of the most detailed views, which may specify how leaf-level, mostly computational processes transform the system’s atomic parts. This detailed information provides a solid basis for model simulation and execution. The domain-neutral nature of OPM, which does not favor systems engineering, software engineering, or any other engineering domain, such as mechanical, aerospace, or electrical engineering, is conducive to cross-domain collaboration not just among engineers with various educational backgrounds but also among system architects and designers on one hand and other system stakeholders, such as beneficiaries and users, on the other hand. From a software engineering viewpoint, our approach positions MAXIM as a bridge between standard, traditional textual programming and visual programming. MAXIM offers a smooth, seamless transition that ideally blends these two approaches; There is always a certain point along the detail level spectrum, beyond which it is no longer desirable or “cost-effective” to stick to visualization. At that point, the transition to text-based program specification makes more sense, as demonstrated in [18]. In addition, adopting purely visual programming might decrease model comprehension. For example, a simple arithmetic expression can be represented in a few tokens, whereas visually it can capture a large area on the canvas. MAXIM enables the visual and textual approaches to be complementary rather than competing, so systems and software engineering benefit from both: Visualization is good at expressing high-level and abstract aspects of the system, while text is good for concrete, low-level computations.

To the best of our knowledge, this work is the first to introduce a unifying framework that facilitates integrative modeling of different domains governed by several engineering disciplines. Being OPM-based, MAXIM is not limited by the kinds of domains that comprise the underlying system, because OPM is based on a minimal universal ontology, which has indeed been applied in multiple domains, from civil aviation [19] to molecular biology [20].

By using MAXIM, we built a new kind of model – a diagnostic model. Medical knowledge in general and pediatric medical knowledge in particular have been increasing significantly over the past decades. Therefore, it is hard or maybe even impossible for a pediatrician to be updated even in her or his field of expertise. In case the etiology of a medical problem is not well defined and has no consensus, models are subject to frequent changes. New factors might dynamically be added, and existing ones might be changed or removed. Such modifications affect the model structure and will eventually invalidate the dynamic part due to lack of synchronization. The dynamic nature of medical issues makes it difficult to develop a long-term write-once tool. A single tool that is based on a simple, coherent methodology, rather than a chain of disparate tools, will enable medical doctors to be actively involved in the tool development and its tuning as new factors and considerations emerge. Based on this motivation, we developed FTTell—a model-based improved, simple “one-stop-shop” medical diagnosis tool for assessing (“Telling”) the potential for pediatric FTT during the perinatal and postnatal stages.

## **OPCloud**

OPCloud is a real-time collaborative web-based environment for model-based systems engineering (MBSE) that implements and enables modeling in OPM ISO 19450 (Figure 4).

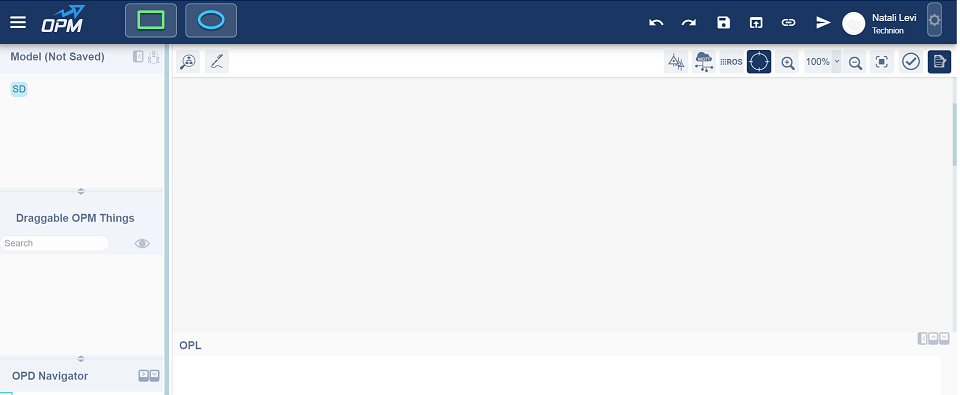


Figure 4: OPCloud screenshot

Since MAXIM is implemented as part of OPCloud, we briefly describe OPCloud “under the hood” and the design considerations leading to its architecture. An OPM model in OPCloud is composed of three layers: Drawn, Visual, and Logical. The user interacts with OPCloud via the Drawn layer, which uses Rappid [21]. The Visual layer is mapped one-to-one onto and decoupled from the Drawn layer. Every Drawn element (thing or link) has a Visual element, which reflects its corresponding Drawn element in a package-independent manner. As a model might have many Visual (and Drawn) elements relating to the same logical one, every element in the Logical layer can point to many Visual elements. The rationale behind this separation of concerns is twofold: (1) the Drawn layer is dependent on the Rappid diagramming package. Replacing Rappid with a different diagramming package will affect only the Drawn layer rather than the entire system. (2) Changing the name or similar properties of Drawn elements is done at the Logical layer to facilitate their propagation to all the other appearances of these OPM elements.

OPM is both a language and a methodology. OPM’s methodological part includes guidelines for limiting the size of a single OPD and refining the model in lower-level OPDs, catering to the second multimedia assumption of Mayer [22] on humans’ limited channel capacity. OPM’s language part is implemented in OPCloud, ensuring correct-by-construction OPM models while the diagram is constructed, as it constantly validates the OPM language syntax. It does so by offering the modeler who seeks to connect two things in the model only the subset of links that produce syntactically correct constructs, and therefore might make sense semantically, preventing the modeler, whether junior or senior, from introducing wrong constructs into the OPM model. Moreover, the ability to execute the model at any stage increases the model correctness, since execution is the highest level in the model fidelity hierarchy [9].

## **Model Fidelity Hierarchy**

The design process of a system usually starts with the most obvious and simple way by writing a specification document [23]. In this step, a designer and a field expert or a product client are involved. They often start with eliciting requirements and specifications verbally. This is a major source of miscommunication, since often the parties mean different things when they use the same words. A slight improvement is achieved when the spoken words are written into a formal document, as putting requirements in writing usually requires more discipline, especially when authoring a legally binding document. Yet, a major problem with such requirements specification documents is that they almost inevitably prone to containing potential mistakes, because there is no automated verification that the document can be passed through, and the writer may write a wrong fact or even two opposite, contradicting facts or requirements in two different places in the same document [24]. It is likely that a reader of a sufficiently long requirements document written in free text would encounter difficulties remembering all the facts and spot all the contradictions within it [25]. Given these severe drawbacks of a free text document, the next, more formal way to design a system is to adopt an MBSE approach – creating a conceptual model of a system. The model is expected to be based on facts and requirements from the specification document and therefore to faithfully reflect what needs to be designed and implemented. By using a proper modeling methodology, language, and tool, some of the contradictions, deficiencies, and redundancies in the free text may be resolved during the modeling process. However, there is a limit to what qualitative conceptual modeling can resolve in terms of overcoming the above shortcomings of free text documents.

As part of this research, we present the model fidelity hierarchy (MFH), which includes two new stages that are enabled by MAXIM: computational modeling and execution. These stages advance the state-of-the-art in ensuring accurate design that meets as many of the original requirements as possible. Each level in the MFH helps to improve the model achieved at the previous level, making it more accurate, as mistakes are detected and eradicated from the model. The overall result is that the system represented by the model becomes less error-prone, which contributes to increasing its soundness. Modeling mistakes can be discovered and fixed by MAXIM.

## **Usage and Applicability Examples**

## **The Braking System Model**

To demonstrate the applicability of MAXIM, we model and analyze a real-life case study of an aircraft braking system designed by Airbus. Each subsystem of this system is currently modeled and designed by experts of some engineering domain without having a full picture of the other subsystems. Our joint overall model provides all the stakeholders from disparate engineering domains with a holistic, common view of the entire system. As the system is executed, and as the results are presented in real time, the interactions between elements from its different subsystems enable deep understanding of the system’s function, structure, and behavior under different conditions.

## **The Landing Gear Model**

To exhibit the MFH we used MAXIM to model a conceptual Vee model of an aircraft's landing gear system and compute its design parameters. In our previous work [26], before the development of MAXIM, the model was purely conceptual and qualitative. As we drilled down into the details, we started to populate the model with quantitative elements as the needs arose. However, the computational part in [26], which used Matlab, was not integrated into the model. To achieve this integration, we used MAXIM to expand the original qualitative model with computational and executable capabilities that are embedded into and spread across the originally conceptual model. With MAXIM, these capabilities are added naturally, effortlessly, and seamlessly. The result is a unified model that combines qualitative and quantitative aspects of the landing gear system. The uniqueness of this approach is that it enables one to numerically calculate the various landing gear design parameters as computations that are embedded naturally in their wider qualitative context, rather than as separate, disconnected mathematical equations.

Moreover, as OPM has a built-in mechanism for dealing with complexities, modeling complex systems of systems (SoSs) is feasible. While in the previous work [26], the Vee model, the cargo system, and the associated computations were three separate models, in this research, this complex system is represented in a single unifying, seamless model, which includes the three previously separate models. Our approach contributes to developing of SoSs, which is desirable in many contexts because different systems that collaborate to produce a global behavior are less likely to fail when modeled as a larger SoS. Our approach ensures that all systems are modeled using the same language constructs and the same environment regardless of inherent differences between these systems.

One of the new traits we discovered as we were engaged in the new modeling approach is that by adding units, values, and computational functions to the model, we detected and corrected inaccuracies in the conceptual model of the aircraft's landing gear. Not less importantly, we were able to find mistakes in the calculations and correct them. In the next, top level of the model fidelity hierarchy, model execution, we were able to verify whether the new model is correct. At this stage too, we discovered additional conceptual and computational mistakes. It seems that synergy between the different aspects of the landing gear OPM model yields better results and helps detect flaws when developed as a whole rather than in separate, disjoint consecutive parts. Importantly, the continuum of the levels in the model fidelity hierarchy enables us to trace engineering decisions to business requirements, enabling us to convey the “big picture”. For example, the high-level business requirements of transporting goods through air percolates downstream through the model layers till it reaches purely technical details, such as the need to calculate the values of the airplane’s wheel nominal diameter and tire nominal diameter.

## **Diagnostic Models: A Third Kind added to Descriptive Prescriptive Models**

Models are currently used as descriptive or prescriptive. In this research, a third kind of models is proposed – diagnostic models. System failures are sometimes easily missed or not diagnosed on time. To diagnose a failure, system experts must be updated on the relevant literature related to all the subject aspects and examine the system from all its relevant aspects. Given the knowledge explosion over the past decades, it is difficult and sometimes impossible for an expert to be updated in her or his domain. MAXIM enables representing both the qualitative and quantitative aspects of the diagnosing process in a model using up-to-date knowledge and execute it.

The diagnosis process usually involves both qualitative and quantitative aspects. The relevant data for it can be represented in a model and executed by using MAXIM. Decisions can be made based on the diagnostic model and it's the execution results, and the model can be updated with knowledge gained via new research.

## **FTTell – Failure To Thrive Diagnostic Model**

As an example of a diagnostic model that uses MAXIM, we have developed FTTell – a system that diagnoses children’s Failure to Thrive – FTT, which is sometimes easily missed or not diagnosed on time. While different reasons may cause a child to deviate from the normal stature or weight for age and gender, it is not always the case that low weight or stature implies that a child fails to thrive [27], [28], as some of the reasons may be genetic- or nutrition-related. To diagnose FTT, the pediatrician should examine the child from many aspects, including parental (deprivation syndrome), prenatal growth, birth weight, and postnatal growth. Information about the parents, such as their heights, mother’s nutrition status, and her emotional stress during the pregnancy, are also highly relevant. For making the best and accurate diagnosis, the pediatrician must be updated in relevant medical research related to all FTT aspects. It is difficult and sometimes impossible for a pediatrician to be updated even in her or his field of expertise, as the medical knowledge in general, and pediatric medical knowledge in particular have been increasing significantly over the past decades.

Our model represents succinctly and consistently the knowledge about the child at the perinatal stage – the period immediately before and after birth, and at the postnatal stage – the period from birth until the child turns five years old. FTTell provides for collecting the required data, including the weight and length at different pregnancy stages for getting an assessment of the child’s FTT potential, and weight for different stages after birth for indication of whether the child suffers from FTT, and if so – the FTT severity. Since diagnosis of FTT involves both qualitative and quantitative aspects, we have used MAXIM for representing the data in a model and executing it. Any member of the medical staff can insert the required data into the model. The model with the data can be easily executed in a single step, so the staff can watch the results, track and analyze issues, and let the pediatric expert use it as a decision-support tool with no need for a computer professional’s help. The model can be easily updated and synchronized with new relevant data and knowledge that are likely to emerge as the research on FTT is evolving. We have evaluated our model using real data taken from 100 cases of children, about 70% of whom suffer from FTT at different severity levels. As we show, FTTell correctly indicated the diagnosis of FTT in most of the cases, sometimes even where experts may have had problems diagnosing it.

## **Contribution**

The contribution of this doctoral thesis is manifested in three dimensions: (1) Theoretical: we established the concept of model fidelity hierarchy (MFH) and added to it two new stages: computational modeling and model execution. As a result, we get a combined hardware-software-humans model that constitutes a complete and accurate executable speciﬁcation of the system from its top-level abstract view all the way to the most detailed view. In addition, we introduced a new type of model – a diagnostic model which joined the descriptive and prescriptive model types. (2) Methodological: we presented a methodical approach to executable model creating with MAXIM, which enables describing the structure and behavior of the system, as well as its quantitative requirements and computational, software-driven aspects. (3) Practical: MAXIM has been implemented as part of OPCloud – the web based OPM modeling environment. This work can also advance both round-trip engineering and teaching programming for novices.

## **Thesis Outline**

The rest of this thesis is organized as follows:

Chapter 2 presents and discusses current software and systems engineering integration approaches. Chapter 3 discusses OPM and its OPCloud modeling environment. Chapter 4 describes MAXIM in detail. Chapter 5 presents a conceptual model of an aircraft braking system, which serves as a case in point. Chapter 6 shows a discussion and presents the limitations of MAXIM. Finally, Chapter 7 is a summary of this work and suggestions for future research direction.

# **Background and Literature Review**

## **The System–Software Gap**

Almost any contemporary system consists of intertwined hardware and software elements, so system and software engineering must be fully integrated [13], [11]. One of the conclusions is that the content of a system or different sub-systems in a complex system should be modeled jointly so they are interoperable, well organized, and consistent, but this goal is hard to achieve [29]. Several solutions to the interoperability problem have been proposed, including levels of conceptual interoperability [30] and openMBEE, which serves as a first step of capturing consistent system data [31]. However, these efforts ended at the conceptual model level, without software integration and model execution; hardware and software are still far from this ideal. Domain-speciﬁc tools, such as Modelica [32][,](#11) Simulink [33], UML [34], and SysML [35], are used for modeling different system domains in both the system and the software aspects, but integrating these languages and tools for joint usage in all the system domains persists to be a challenge [36], [37][.](#11) A signiﬁcant reason for the gap between system and software engineering is their different development tempo [13]. Agile software development is in sharp contrast to the long lifecycle of hardware development. Systems engineering has started to adapt agile software development approaches, and OPM is certainly poised to provide an excellent platform for agile systems engineering, but we are still far from the point at which the software engineering agile approach is embraced by systems engineering. The present work can catalyze such adoption. Another meaningful challenge to bridging the systems-software gap stems from academic education traditions: Academic programs of systems engineering are newer and separated from those of software engineering, producing professionals that are regrettably experts in only one of these two ﬁelds and are only vaguely familiar with the other.

Different approaches to integrating systems and software engineering or adding computations to conceptual models have been suggested, but they all divide the system into several parts, or domains, and model each domain separately. One such approach is aspect-oriented modeling [38], which integrates modeling methods instead of tools [39][.](#11) When modeling a new system, only the main functionality is usually modeled, with the common wisdom being to model non-functional aspects, such as cost, communication, veriﬁcation, and security, separate from the main model [8].

Integrating systems engineering and software engineering has been examined in computational science—the use of computers and communication hardware, software, algorithms, analytics and simulations for solving scientific and engineering problems. After reviewing about 50 publications [11], the authors’ conclusion was that more research needs to be done, as programming languages are general purpose languages, while the combination of software engineering with computational science is expressed in domain-speciﬁc languages. Another research [10] aimed to integrate two separate ISO standards for systems engineering and software engineering: ISO/IEC 12207, which deﬁnes the software engineering process, and systems engineering processes, which are deﬁned in ISO/IEC 15288. Software Platform Embedded Systems Modeling Framework (SPES MF) is suggested as an artifact-oriented, model-based approach, which supports compliant recursive engineering of embedded systems[.](#1ksv4uv) SPES MF proposes dividing the system into four domains: requirements, functional, logical and technical, and modeling each domain separately. The model is decomposed into smaller problems, which are recursively treated in the same way in a new layer. On the software engineering side, however, according to ISO/IEC 12207, the granularity layer concept is not applied, so only one layer of artifacts exists. To integrate the system and software aspects in these two standards, the authors suggest mapping the artifacts in lower layers of the system model to those in the software model.

## **OMG Attempts at Computational Executable Modeling**

The Object Management Group (OMG) [40] has issued a call for proposals to SysML v2 RFP [41], which will replace the current SysML [35]. New required features emphasize that the current SysML does not have computational and execution capabilities; currently, the parametric diagram is the only relevant one, but it is static and not executable. OMG has tried to integrate conceptual modeling with computational and execution capabilities by developing xUML [42] and fUML [34], [43], [44], [45], but none succeeded to include the entire system without dividing it into several subsystems that have little or no interaction between them. The acute need to integrate conceptual modeling with computational and execution capabilities has been clearly demonstrated in our prior work [26], in which we could not seamlessly extend the conceptual model to include the computations. OMG has proposed Unified Modeling Language (UML) [46], [47] as a software modeling language that serves as the main basis for Model-Driven Architecture (MDA). The full infrastructure [46] and superstructure [47] definitions of UML, which add up to well over 1000 pages of technical reading, is a high barrier of entry to full-fledged UML-based modeling. This size and complexity of UML has been shown to have quite a few negative aspects, including operational semantics problems [48], semantic duplications among the various diagram kinds [49], behavioral diagrams that are hard to understand and with value that does not justify the cost [50], time-consuming and frustrating meta-model navigation [51], numerous modeling concepts with poorly defined semantics and lightweight extension mechanisms that make learning and applying UML difficult [52], lack of system-theoretical ontological foundation and lack of support for integrating structure and behavior in a single model, which puts the intellectually-demanding burden of flipping back and forth between at least two diagram kinds entirely on the developer's shoulders [53]. Another problematic aspect of MDA is that the model and the system are connected only by forward transformations, after which the parts of the system that were not defined in the model are implemented manually by the software developers. As the implementation process continues, the model of the system quickly becomes outdated [54], so its usefulness decreases sharply.

## **Different Modeling Approaches**

According to the international standard ISO 15288, a system engineering process can be divided into requirements development, architectural design, technical evaluation, and synthesis [55]. The International Council on Systems Engineering (INCOSE) has adopted the Vee model to define and develop arbitrary systems. The left leg of the Vee model represents the decomposition of requirements and creation of system specifications. The right leg represents integration of parts and validation of the system [26].

The common wisdom is that modeling and simulation are valuable solutions for system analysis and improve the development of complex cyber-physical systems [56]. Another approach for creating an executable model has shown the limitations of system simulation and execution [57]. The simulation lacks information about the main purpose of the system, and achieving a model that is amenable to simulation or execution requires four steps: (1) starting with the abstract "target system", (2) translating it to an "abstract model" that consists of formal definitions based on the target model (3) building a "computational model" that represents the abstract model and is founded on formal rules and definitions of agents, and (4) "software model", which is a translation of the computational model. Having performed these steps, we end up with four models, but we execute only the last one. Although the final model is executable, it might not be consistent with the original purpose, as in each step some piece of information may be lost or changed [57].

SysML, the OMG systems modeling language, is intended to facilitate the application of the modeling approach to create a model of the system [58]. Most SysML models are created for documentation purposes, while others enable better understanding of the underlying system and validation of desirable behavior of that system. In the former, the focus is on syntax and notation, while in the latter, the focus is on execution semantics [59]. Executable models require an engine that “understands” the execution semantics.

## **OPM Compared with Other Modeling Approaches**

In contrast with UML and its SysML profile, OPM is not just a language; it is also a methodology. OPM captures the three main aspects of any artificial system: function – the utilitarian aspect – what the system does; what value it provides and for whom, behavior – the dynamic aspect – how the system operates, and structure – the static aspect – what is the system comprised of, and how its parts relate structurally. A UML activity has similar semantics to that of an OPM process. An activity diagram is a UML diagram that, along with sequence diagram and state machine diagram, captures the system’s dynamic aspect by specifying the chain of activities from one to the next, optionally adding the input and output objects for some activities. OPM has a single kind of diagram – Object-Process Diagram (OPD). The system aspect described by a UML Activity diagram can be easily specified in an OPD by an invocation (lightening-like) link from one process to the next. Moreover, within an in-zoomed process, the timeline OPM principle is used. Like UML Sequence diagram, this principle asserts that within an in-zoomed process, the order of execution of processes is dictated by their vertical location, so a process located higher is executed before one below it. Concurrency, which can be modeled in Activity diagrams, is expressed in OPM by positioning processes at the same height. Activity diagrams also support branching, which in OPM is modeled by using stateful objects (objects with two or more states) and condition links emanating from each state to a corresponding process to be executed. We are not aware of UML modeling aspects or capabilities of any of the 14 UML diagram kinds that OPM with its single diagram kind, OPD, does not support. The fact that the structural and behavioral aspects are modeled in the same and only kind of OPM diagram, OPD, is an advantage, because it explicates how processes transform objects at various levels of detail. OPM has key modeling concepts and constructs that UML and SysML [60] do not have. For example, the OPM notion of enabler (agent – human or instrument – non-human), which is an object that is required for a process to occur but is not transformed by that process – is missing in UML and SysML. Object Constraint Language (OCL) [61] and Object Process Language (OPL) of OPM are two different languages with different motivations and purposes. OPL is an equivalent textual representation of an OPM diagrammatic model. As such, it conveys to the modeler the same facts that he models using graphical elements. The importance of this seemingly redundant view is threefold: it helps the modeler to verify the fact she has just added by reading its meaning in a natural language, it helps non-technical stakeholders to understand the facts and take part in the early stages of the system requirements elicitations, and it caters the dual-channel assumption of multimedia learning, which states that humans process verbal representation and graphics in different channels with some people being more text-oriented while others are more visually inclined [4]. OCL is a complementary language that supplements UML models with constraints and queries. As such, its main purpose is to close the gap that UML has while OPL purpose is to improve human comprehensibility and model verification.

There are other modeling languages, tools and methods, which are, for the most part, domain-specific or aspect-specific: AmmA [62], StateML+ [63], MECHATRONIC UML [64], Domain-Driven Design (DDD) [65], and LML [66]. The major advantages of OPM over these languages and methods is that OPM is domain-agnostic, because it is founded on the universal minimal ontology of objects as things that exist and processes as things that transform objects by creating or consuming them, or by changing their states. Other unique advantages include OPM’s bi-modal representation (graphics and text), its diagram kind singularity (only one kind of diagram, OPD), which alleviates the cognitive load of traversing different diagram kinds to mentally combine the structural and behavioral system aspects, and its refinement-abstraction mechanisms (in-zooming – out-zooming, unfolding – folding, and state expressing – state suppressing) which provide for systems’ complexity management. A few works have used OPM to model cyber-physical systems (CPSs). In particular, a recent work [67] that uses OPM provides a solution for the cyber-physical gap (CFG). I5 is an OPM-based framework to enable systems of systems architecting [68]. An attempt at integrating computations into conceptual modeling in OPM yielded an integrative model using MATLAB and Simulink [18]. In that work two approaches were demonstrated: (1) AUTOMATLAB, which generates a MATLAB code based on the OPM model, and then enhance it further using MATLAB. The main drawback of this method is that OPM semantics may be breached while enhancing the code using MATLAB. In addition, the qualitative functionality is limited to several predefined functions that can be expressed by an OPM model. (2) OPM/CS, which allow using MATLAB code stored in MATLAB directory. The code is called from an in-zoomed process by specifying the folder in which the user-defined function or diagrams are saved, using the same name for both the OPM process and the MATLAB function or Simulink diagram. The main problem with this method is that the code execution is performed in separate tool and the functions are predefined. In this work, we show how OPM-based conceptual modeling and computations are combined naturally and seamlessly, opening the door for an end-to-end engineering environment that does not mandate abandoning the conceptual model when transitioning to detailed design, in which computations are necessary and prevalent.

## **Medical Knowledge Representation**

To provide a solution, medical knowledge has been translated into computer interpretable formats as models or pseudo-code. Examples include GLIF [69], Arden [70], PROforma [71], EON [72], and GLARE [73]. The major problem with these frameworks is that they are not intuitive, and therefore medical doctors find them difficult to use. A 2016-published executable tool for representing medical knowledge and treatment protocols [74], which, to the best of our knowledge, is the state-of-the-art, translates medical data into Statechart models using the open-source tool Yakindu [75]. Since this tool is not executable and therefore does not provide for verification, an additional tool, Y2U [76], translates the Statecharts model from Yakindu to UPPAAL timed automata. To use this framework, medical doctors have to build the Statecharts model using Yakindu, then run Y2U to transform the model to UPPAAL timed automata, which can finally be executed, usually requiring help from a computer professional. Any change in the input data or the model structure, mandates updating the Statecharts model, re-running the Y2U tool, and executing the UPPAAL timed automata.

# **MAXIM—Methodical Approach to Executable Integrated Modeling**

MAXIM extends OPM [1], [2] ISO 19450:2015 [3] with computational capabilities, using the same concepts and principles of OPM. An OPM model expresses constructs—assemblies of things (objects and processes) connected by links. This OPM’s minimal universal ontology has proven to successfully capture conceptual models of systems from various domains. One of MAXIM’s design criteria has been to extend OPM with computational capabilities without augmenting its minimal, yet universal ontology. Therefore, MAXIM is built of computational constructs comprising objects, processes, states (which can have numeric or symbolic values), and relations between them that convey meanings that are analogous to their non-computational counterparts, as will be elaborated below. While the conceptual model is represented at increasing levels of detail and refinement, computations are performed only at the most detailed, atomic, leaf-level processes. Similarly, the inputs and outputs of these computational processes come from and are stored in computational objects, which are discussed next.

## **Computational Objects**

An attribute is an object that describes another object – the exhibitor of the attribute. An attribute optionally stores a value. This value shall be recorded in the value window — the rountangle (rounded-corner rectangle) used to designate a state in an object and a value in an object that is designated as an attribute of another thing (object or process)—the exhibitor of that attribute. Graphically, an object is defined as an attribute if it is linked in at least one OPD to the exhibitor – the thing that exhibits the attribute – via an exhibition-characterization link.

The state or value symbol—the rountangle—represents a placeholder for a value. The text inside an attribute shall be of the form **Name**, optionally followed by [Measurement Unit]and {alias}: Name [[Measurement Unit]] [{alias}].

Measurement Unit is replaced with the name of the measurement unit, which shall be a reserved OPL word (and therefore be recorded in non-bold Arial font). Measurement units shall use the International System of Units, abbreviated SI [77]. The names of the seven SI base units shall be written in brackets as follows:

[m] for meter (length),

[kg] for kilogram (mass),

[s] for second (time),

[A] for ampere (electric current),

[K] for Kelvin (thermodynamic temperature),

[mol] for mole (amount of substance), and

[cd] for candela (luminous intensity).

All the other units are derived units. For example, velocity shall be denoted [m/s] or [mph] (miles per hour). Other units can be used as well, for example [mile], [mph], [gallon], [liter], [deg C], [deg F], etc. Where necessary, a converting process shall be incorporated as part of the model. An example is the process deg C to deg F Converting, with the appropriate converting process provided as a combination of mathematical operations inside the in-zoomed view of that process or as a formula in the process text (or code) attribute (field). Attributes can be unitless, i.e., not have a measurement unit specification. Objects in general can be unitless as well and used as arguments in numerical computations, in which units are unspecified, because they are not relevant.

Alias is replaced by the alias of the object. An alias shall be a single word or an acronym, which will make it shorter and easier to refer to the object. An alias shall be written in curly brackets after the units which are written after the object's name. An alias is mostly used for referring it in a software code as one of the input variables, especially when the object name includes spaces and therefore cannot be used for reference in software code.

In Figure 5 we can see an example of an informatical, computational object called **Weight**, which is an attribute of the physical object **Car**. The object **Weight** has a value equals to **3500**, Measurement Unit [**kg**] and an alias {**w**}.

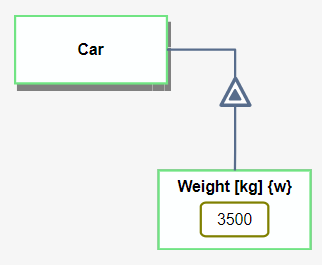


Figure 5: A **Car** object with a **Weight** attribute having a specific value **3500**, Measurement Unit **[kg]** and as alias **{w}**

## **Computational processes**

A computational process is a process to which a computation is attached. Following this addition, a new **Computability** property is added, whose possible values are **computable** and **non- computable**. If the process is a regular process, as defined in the current ISO 19450:2015, the value of the **Computability** property of that **Process** is **non- computable**. Alternatively, a **Process** whose **Computability** value is **computable** is a computational process — a process to which a computation is attached. The computation may contain formulas, which uses input values from one or more computational objects to perform a calculation and provide a value for the (computational) resulting object. This computation shall be executed if the process is invoked and its precondition is met. The computation in a **computable** process can be one of the predefined basic arithmetic operations, a user-defined function (currently expressed in TypeScript), or a link to an external designated tool, such as MATLAB/Simulink or Modelica. The predefined basic arithmetic operations are the following: Abs, Adding, Average, Concat, Cos, Dividing, Exponent, Log, Max, Min, Multiplying, Power, round, Sin, Sqrt, Subtracting, and Tan.

The naming convention shall follow the conventions defined in ISO 19450 with the additional **Computability** property. The default value of the **Computability** property is **non- computable**, i.e., by default, the process has no computational function. A process whose **Computability** property is **computable** is a **computational** processand it is a specialization of an informatical process. A **computable** process has a property called **Container**, which holds the computation that is specified for that process.

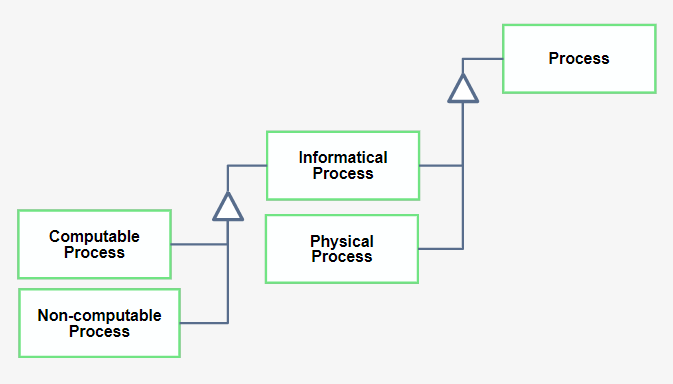


Figure 6: Process specializes into Physical Process and Informatical Process and the latter specializes into Computable and Non-Computable Process

A **computable** process is represented visually by adding a pair of adjacent open and close parentheses to the right of the process name, separated from the name by one space. Figure 7 depicts the difference between a physical process (**non-computable**) and a **computable** process.

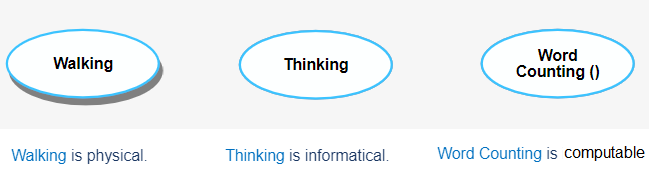


Figure 7: Exemplifying the **Computability** property of a **Process**. Left: **Walking** is a physical, **non-computable** process. Center: **Thinking** is an informatical, **non-computable** process. Right: **Word Counting ()** is a **computable** process

The **Essence** property of a **computable** process shall be informatical since it is computational by definition. A **computable** process shall be a leaf process, i.e., a process that is not and cannot be in-zoomed, as it is a basic function without OPM subprocesses. Hence, an in-zoomed process or a process that the modeler plans to in-zoom shall not be **computable**. If during the modeling process a **computable** process must be in-zoomed, its computation can be moved to one of its subprocesses or distributed among two or more subprocesses, with adequate changes to the computation as needed. Figure 8 demonstrates an example of a **computable** process **Time To City Center Calculating ()** which gets as an input two objects: **Velocity and Distance From City Center** and writes the result in **Time To City Center**.

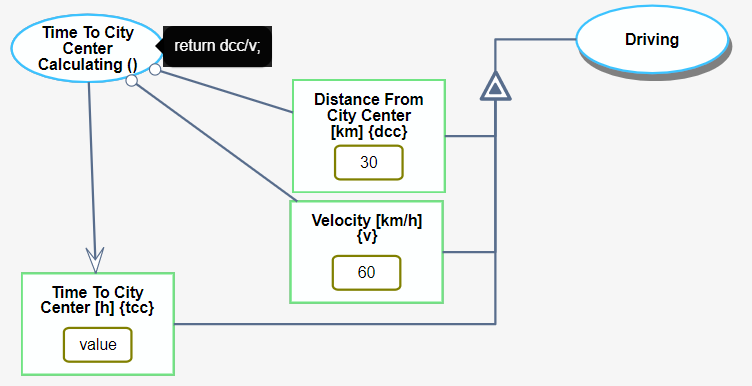


Figure 8: **Time to city center** is a **computable** process wich gets two input values from two objects and writes the result into the output object **TimeTo City Center**

A computational process may change a value of an object. In this case, the OPL sentence shall change in real time during the execution. For example, before execution:



While after execution it can be:



## **Computational States**

States represent possible situations an object can be at, or, if it is an attribute, possible values that it can be assigned, during its lifecycle. Thus, a state might be one of two kinds: (1) Situational state (or simply state): a situation description, which can be applicable to any object; (2) A value state (or simply value): a state of an attribute. Situational states shall be defined as in ISO 19450. A value can be one of three types: (a) numeric, (b) string, (c) user-defined: a value which is applicable to the type that was selected from one of the (user-defined) type object classes.

Figure 9 shows the difference between situational and computational states. Figure 10 demonstrates the difference in OPL between describing an object with situational states and an object with a computational state.



Figure 9: object **Bank** with situational states **open** and **closed** (left); object **Velocity** with computational state having the value **60** (right)



Figure 10: Difference between opl sentences that describe **Bank** object with situational states and **Velocity** object with a computational state.

## **Procedural Links**

A new functionality is added to procedural links, related to objects with values. A process can consume a quantitative object or use it as an instrument to calculate a result. A consumption link is used to consume an object containing a specific value. It is used for an object that represents a temporary variable, the same way as in a software code a temporary variable is deleted at the end of a function. For example, in Figure 11, for calculating the time in minutes that will take to a car to move 30 km, we will first calculate it in hours in the process called **Time To City Center Calculating ()**, then translate it to minutes in the process called **Hours To Minutes Converting ()**. The final result, **Minutes To City Center**, is used further, but the temporary variable, in hours, is not needed and therefore is consumed. An instrument link is used to transfer the value of the computational object to a computational process but the object persists and may be used later for a different computational process. In our example, **Velocity** and **Distance From City Center** objects are connected to the computational process **Time To City Center ()** using instrument links which means that their values are used for the computation but the objects are not deleted after the computation is done. Finally, for assigning a result of a computation to a computational object we use a result link. The result link changes the value that is written in the computational state after the execution of the model which will be described in section 6.

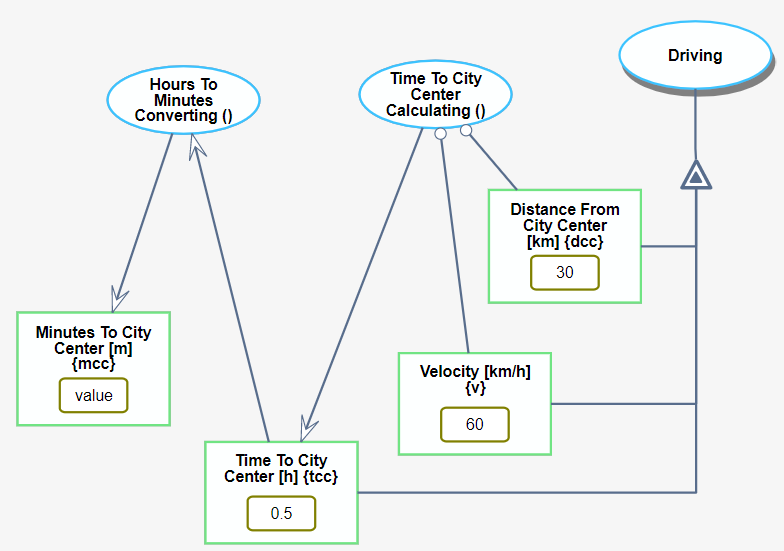


Figure 11: different usages for procedural links

The OPL of the example in Figure 11, with marked sentences for the instrument, consumption and result link is show in Figure 12.

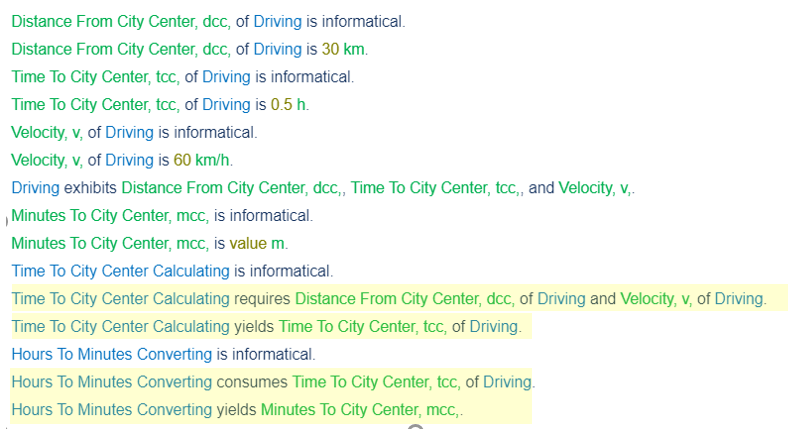


Figure 12: opl with marked sentences for instrument, comsumption and result links

The last procedural link that is used in the computational context is the effect link. An effect link between a computational process and a computational object means that the function that is written in the process reads the value of the object and updates it, as shown in the example in Figure 13.

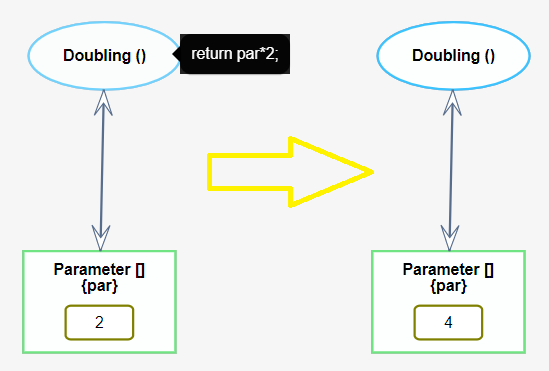


Figure 13: effect link affects the value of the object **Parameter**

## **Structural Links**

A new functionality reserved for an OPD consisting only of structural links is model-specific type definition. In addition to number and string, new model-specific types can be defined by an OPD which will contain only structural links and be named and stored as model-specific type. When adding a new object with a value, one would be able to set the type to be numeric, string, or one of the model-specific types. A model-specific type is analogous to class definition in software.

Using Aggregation-participation relation link fields of a class can be defined. Figure 14 demonstrates a definition for **Aircraft** class having the fields: **Body, Wheel Set** and **Wing Set**. In pseudo code, it would be defined as:

**class** Aircraft {  
 **wheelSet** : **number**;  
 **body** : **any**;  
 **wingSet** : **number** ;  
}

A possible OPM implementation is to define the objects **Wheel Set** and **Wing Set** as having a values that represents the number of wings and wheels respectively. A more verbose way is to have each set a default attribute called **Size**, whose value is the (unitless) number of members in the set, e.g., **3** for **Wheel Set** and **2** for **Wing Set**. This is a more adequate way to model such facts, because the OPL sentences would be:

**Wheel Set** exhibits **Size.**

**Wing Set** exhibits **Size.**

**Size** of **Wheel Set** is **3.**

**Size** of **Wing Set** is **2.**

An aggregation-participation relation with a partial refinee specification (part) is analogous in software to a list to which the user can add fields. Similarly, an exhibition-characterization relation with a partial refinee specification (attribute) is also analogous in software to a list to which the user can add fields. Indeed, in software, there is usually no distinction between parts and attributes, and both are considered “data members”. In OPM, however, there is a semantic difference between parts, which comprise an object, and attributes, which characterize it.

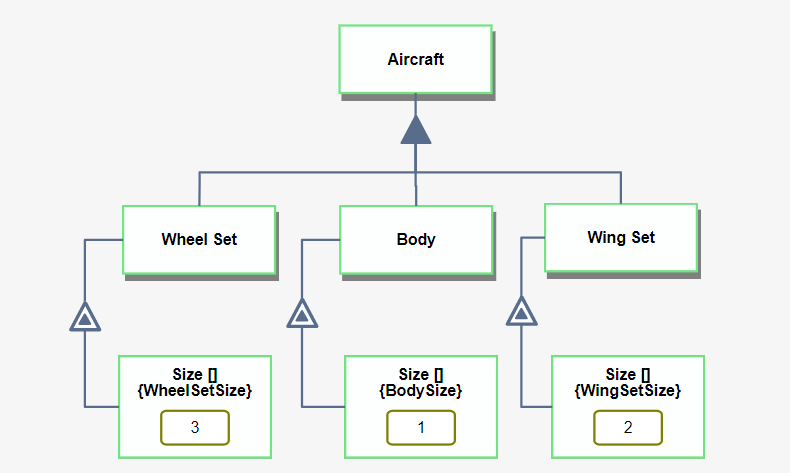


Figure 14: OPD of model-specific type for **Aircraft**

The exhibition-characterization relation may link things in four possible combinations: (1) Both the refineable and the refinee are objects. In this case, the refineable will be the class while the refinee is an attribute in OPM and a field of that class in software. (2) The refineable is an object while the refinee is a process. In this case the refinee is an operation in OPM and a method of the exhibiting class. (3) The refineable is a process and the refinee is an object. In this case, the refinee is an attribute of the process in OPM, while in software it is a parameter of the function exhibited by the refineable. The distinction between input and output parameters is based on the procedural link that connects the attribute with the process: An instrument or consumption link designates an input parameter while a result link—an output parameter. (4) Both the refineable and the refinee are processes. In this case, the function exhibited by the refineable uses the function exhibited by the refinee as an auxiliary function. Figure 15 demonstrates the new addition.

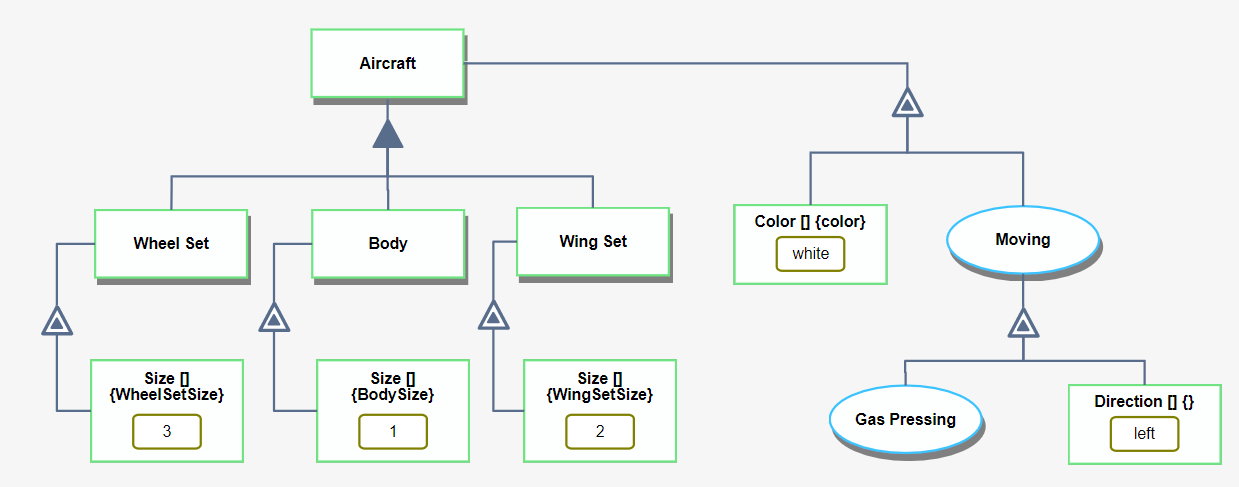


Figure 15: model-specific type for **Aircraft** with features (attributes and operation)

The pseudo code is getting more detailed now. The new part is colored in red:

**class** Aircraft {  
 **wheelSet** : **number**;  
 **body** : **any**;  
 **wingSet** : **number** ;

**color**: **string**;  
 moving (direction){  
 gasPressing();  
 }  
 }

A generalization-specialization relation induces inheritance. Having this link type in OPD of a model-specific type can have two meanings: (1) The refineable and the refinees are objects. In this case, each refinee is an additional constructor in the class represented by the refineable. Another possibility is that each refinee is an inheritor of the refineable. (2) The refineable and the refinees are processes. In this case, each refinee is an overriding function of the function represented by the refineable. Figure 16 demonstrates the new addition.

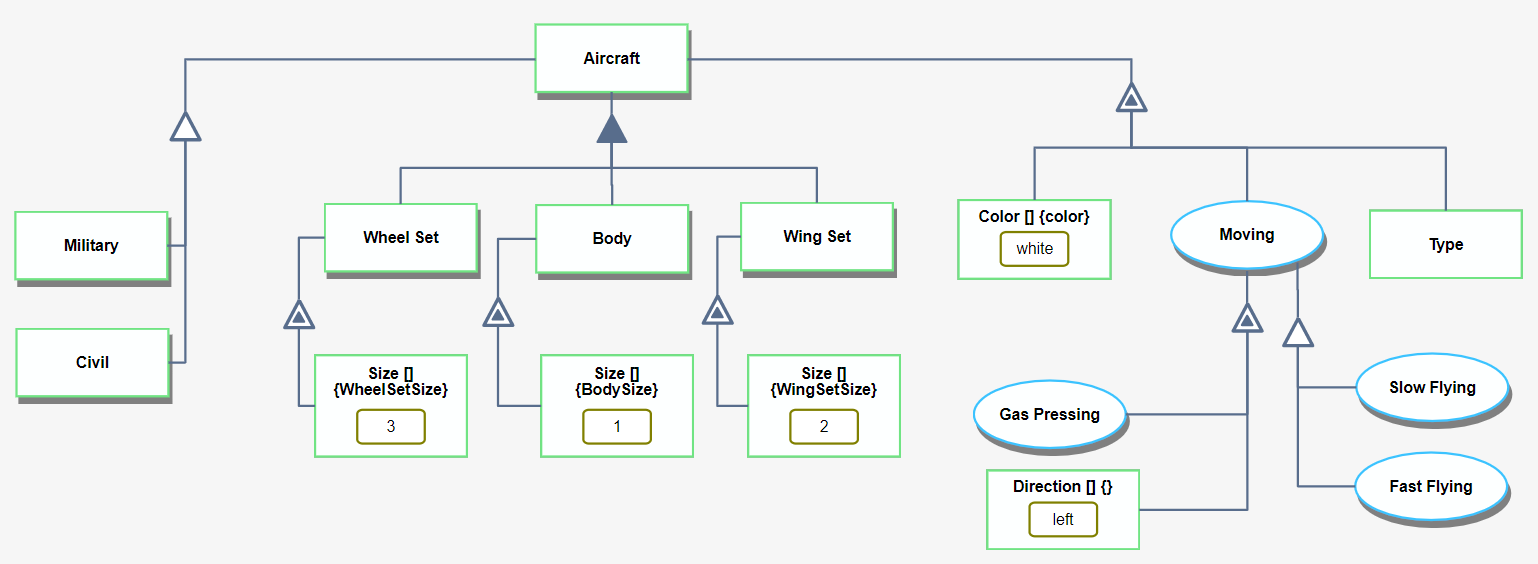


Figure 16: model-specific type for **Aircraft** with object and process specializations added

The pseudo code is getting even more detailed now. The new part is colored in red:

**class** Aircraft {  
 **constructor**(){

this.type = this.constructor.name;b

};  
  **wheelSet** : **number**;  
  **body** : **any**;  
  **wingSet** : **number** ;  
  **color**: **string**;

**type**: **string**;  
  
 moving (direction){  
 gasPressing();  
  **if**(type == **'civil'**)  
 slowFlying();  
  **else if** (type == **'military'**)  
 fastFlying();  
 }  
 }

class Civil : Aircraft {

moving (direction) {

parent.moving(direction);

slowFlying();

}

}

class Military : Aircraft {

moving (direction) {

parent.moving(direction);

fastFlying();

}

}

A classification-instantiation relation cannot appear in an OPD representing a model-specific type, as it designates a specific instance rather a class definition.

# **OPCloud**

In this section we show technically how to define and use OPCloud in general and computations in particular, including model execution.

## **Computational Object**

For creating a computational object, one should first create a regular object by dragging it from the upper left corner to the main working area, called paper, as shown in Figure 17.

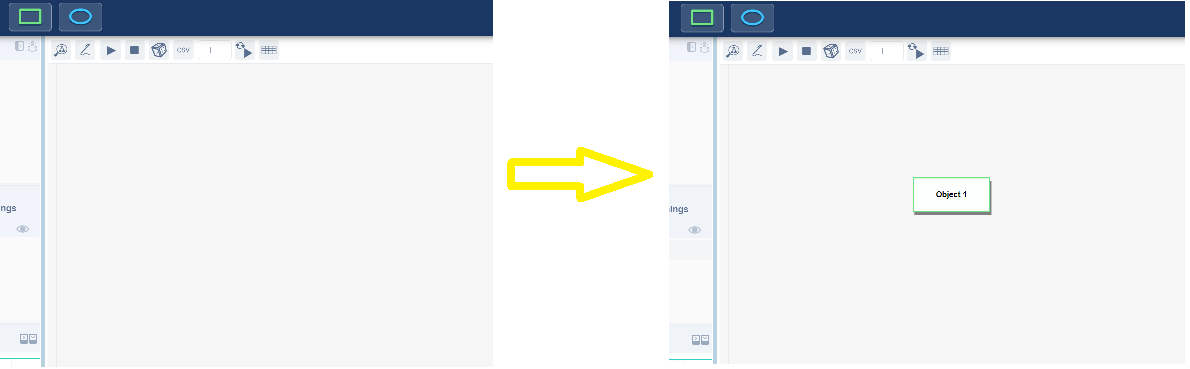


Figure 17: Creating a new object

By double-clicking on the object, a popup window for changing its name appears. In Figure 18, the name is changed to **Number**.

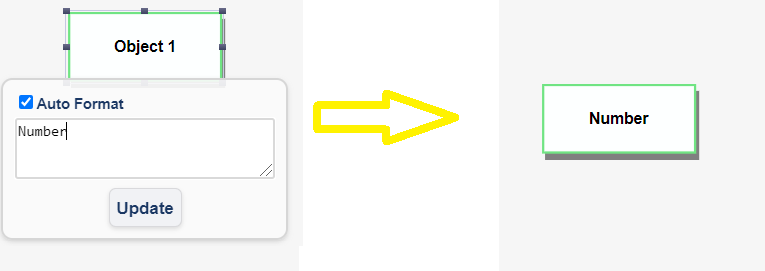


Figure 18: Naming the object **Number**

Clicking on the object opens a menu with object-related operations. The related operations appear in two places: (1) the toolbar – see Figure 19; (2) the halo – a circular context menu (pie menu) next to the element with options for editing – see Figure 20.



Figure 19: Object-related operations in the toolbar

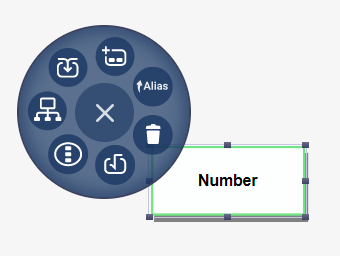


Figure 20:Object-related operations in the halo next to the object

For defining the computational attributes of the object, we click on the square root icon which is marked in Figure 19 and in Figure 20 by a yellow circle. A popup window opens for inserting a value (Figure 21). The default value is "value", and if we do not change it, then during the execution the computational object will be considered as not having a value (usually used for result objects).

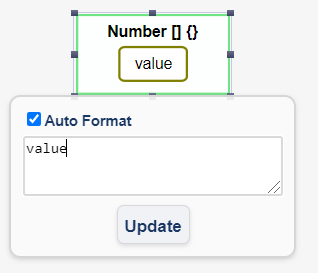


Figure 21: Popup window for value inserting

After writing a value or leaving it with its default, we click on the "update" button (or enter) and get a new popup window for inserting units (Figure 22). Units can be selected manually from a list or left blank, which means that the object is unitless.

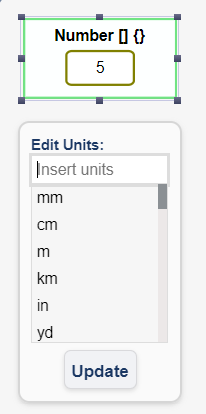


Figure 22: Popup window for units inserting

After another click on the "update" button, the last popup window is opened for choosing an alias, as shown in Figure 23. If left blank, there is no alias to the object, and the name of the object is taken as its alias, provided it does not contain any spaces. If it does, an alias must be defined for use in computations.

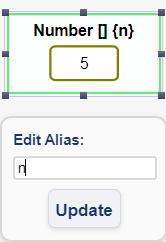


Figure 23: Popup window for alias inserting

For editing the value, units, alias or making the object non-computational, the required option can be selected from the toolbar or the halo (Figure 24).

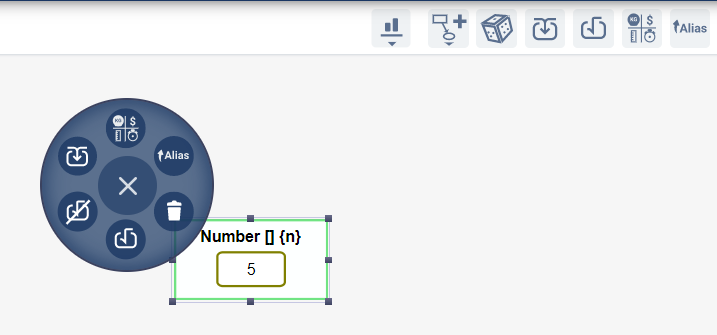


Figure 24: Toolbar and halo for a computational object

## **Computational Process**

For creating a computational process, one should first create a regular process by dragging it from the upper left corner to the paper, as shown in Figure 25.

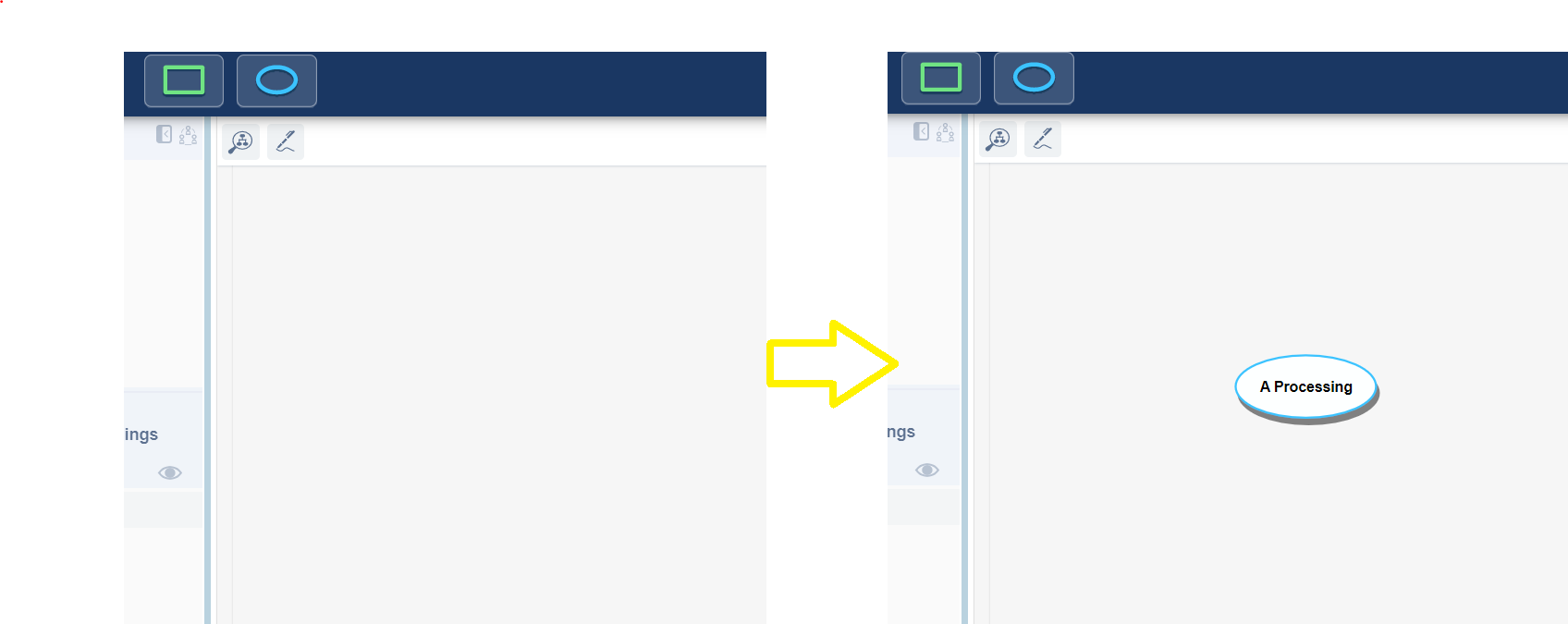


Figure 25: Creating a new process

By double-clicking on the process, a popup window will appear for changing its name. In Figure 26, the name is changed to **Adding**.



Figure 26: Naming the process **Adding**

Clicking on the process opens a menu with process-related operations. The related operations appear in two places: (1) the toolbar - see Figure 27; (2) the halo – a circle next to the element with options for editing - see Figure 28.



Figure 27: Object-related operations in the toolbar

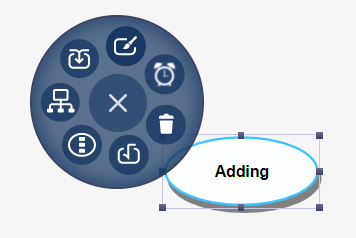


Figure 28:Object-related operations in a frame next to the object

For defining the computational attributes of the process we should click on the square root icon which is marked with a yellow circle in Figure 27 and Figure 28. A new menu will be opened (Figure 29) in which we select the type of the functionality we want to use: Predefined, User Defined or External. (ROS and MQTT are out of the scope of this doctoral thesis)

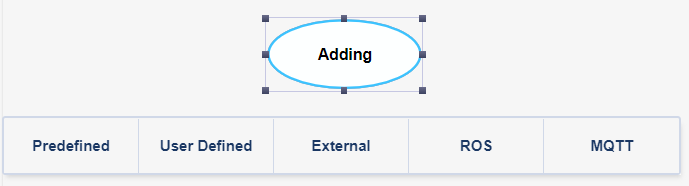


Figure 29: A menu for selecting the type of computational functionality for the process

## **Predefined functions**

When clicking on the Predefined option in the menu shown in Figure 29, a new menu is opened, where one of a list of basic mathematical operations can be selected (Figure 30) and stored by clicking on the "update" button. The implementation of these operations is part of OPCloud software. After selecting the desired operation, the menu is closed, a pair of parentheses is added to denote that the underlying process is computational, and the selected operation can be seen as a tooltip when hovering over the process name (see Figure 30).

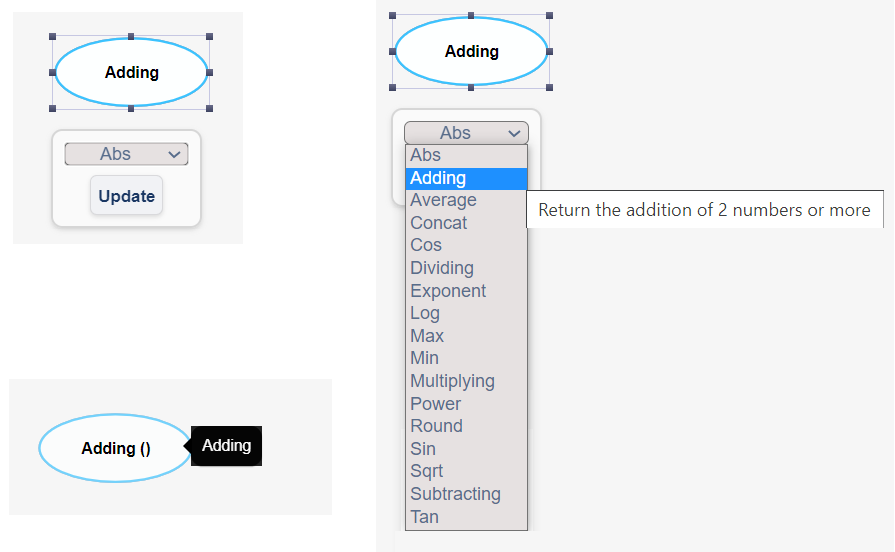


Figure 30: Selecting a basic arithmetic operation for a process

## **User-defined**

When clicking on the User Defined option in the menu in Figure 29, a popup window will be opened, as shown in Figure 31, where the modeler can write a TypeScript code. The length of the code is unlimited, and the variables name are related to the input variables. In the example in Figure 31, two computational objects with names or aliases **a** and **b** must be connected to the process **Adding**. The result of this process will be the sum of the values of **a** and **b**. After writing the function code, the window will be closed, a pair of parentheses will be added, and the function code will be seen as a tooltip when hovering over the process name.



Figure 31: Popup window for User Defined function

## **External functions**

When clicking on the External option in the menu shown in Figure 29, a new popup window is opened, and the user has to select the URL of the external program and the parameters that will be transferred to that program. In the example in Figure 32, two computational objects with names or aliases **a** and **b** must be connected to the process **Adding**. The result of this process is not known because the computation is done in some external program on an external server (localhost:4000 in this example). After defining the Url and the Parameters, the window is closed, a pair of parentheses is added and the Url connection will be seen as a tooltip when hovering over the process name.

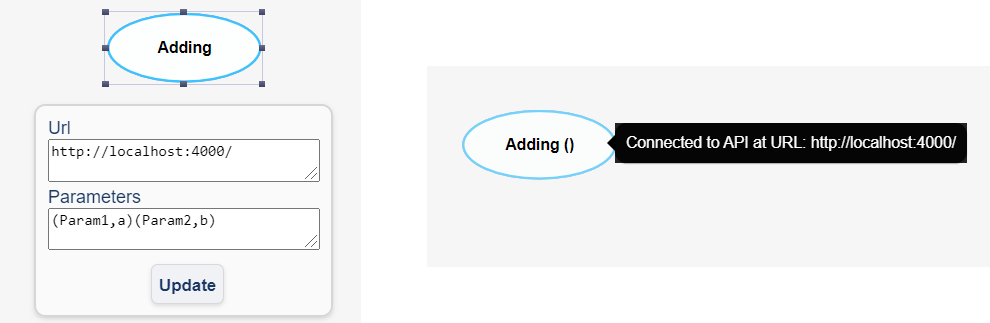


Figure 32: External function definition

## **Model Execution in OPCloud**

For executing a model in OPCloud, one should first create the model on the paper area and then click on the Execute icon on the upper right corner, as shown in Figure 33, circled by a solid black line.



Figure 33: "Execute" sigh circled by a solid black line

Clicking on the Execution sign opens a new menu dedicated for execution options, as shown in Figure 34. Using this menu, the user can perform several kinds of operations: Single execution, Stop execution, Repeat execution more than one time, Export the results to a csv file and Import computational values from a csv file.



Figure 34: Execution menu

## **A single execution**

By clicking on the "play" sign in the Execution menu, three functions, that were developed during this research, run behind the scenes.

First, a function called **checkUnits** is executed. This function scans all computational processes, their inputs and their outputs and verifies the consistency of the units. In case of inconsistency, there is an alert to the user and the user can decide if he wants to keep the units as they are or change them to be consistent. In case of consistency, the system checks the units of the result and if they are missing, it is automatically updated.

Example 1: For calculating the total sum of a fruits bag, we have to sum the **Weight** of the **Apples Set** and the **Bananas Set**. Both written using **kg** units, as well as the result object, like in Figure 35. Therefore, we have no alert.

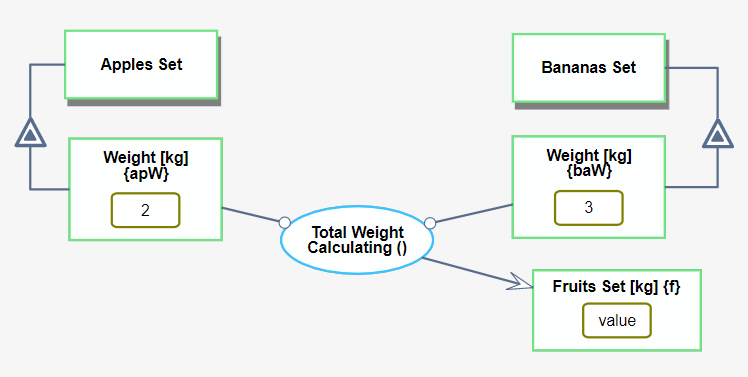


Figure 35: Input and output with same **kg** units

Example 2: The result object has different units, which are not consistent with the input units. In this case, the units of the result object are updated automatically to fit the input, as shown in Figure 36. In this example, the **Weight** of the **Apples Set** and the **Bananas Set** has **kg** unit, but the **Weight** of the **Fruits Set** has **g** units. The **g** is changed to be **kg** automatically.

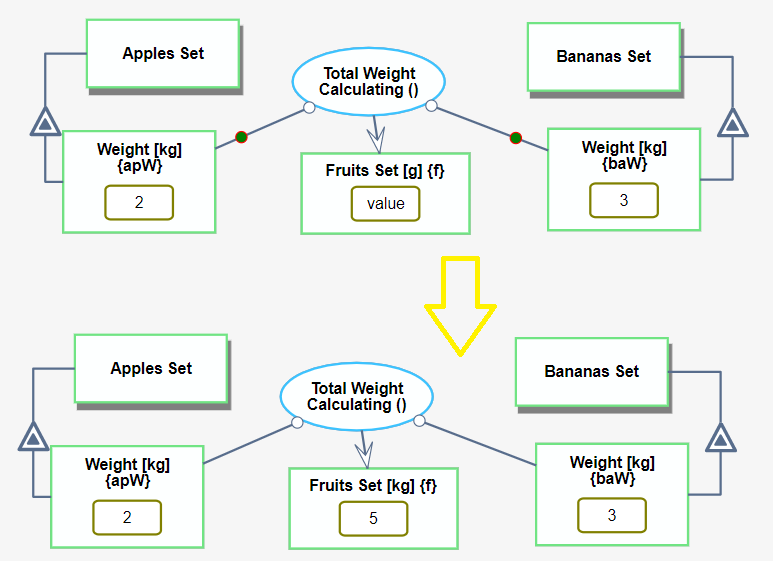


Figure 36: **Fruits Set** units changed from **g** to **kg** automatically because the units of the **Weight** of **Apples Set** and the **Weight** of **Bananas Set** are **kg**

Example 3: Input objects have different units' type. The system asks the user if he would like to convert the units of one of the input objects. If the user wants to convert, he has to choose to which units. Finally, the function is calculated according to the desired units and make the required conversion. In Figure 37 **Weight** of **Apples Set** is in **g** while **Weights** of **Bananas Set** and **Fruits Set** are in **kg**. After clicking on the "play" button, the user is required to choose if he would like to use **g**, **kg** or leave it as it is. The choice popup is shown in Figure 38.

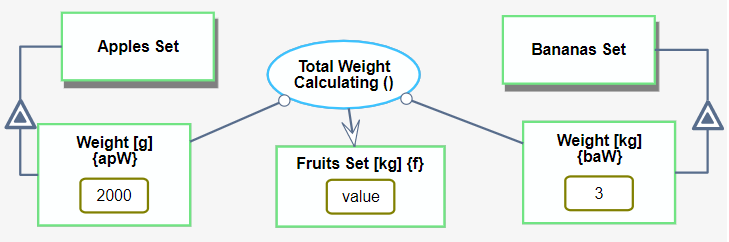


Figure 37: **Weight** of **Apples Set** is in **g** while **Weights** of **Bananas Set** and **Fruits Set** are in **kg**

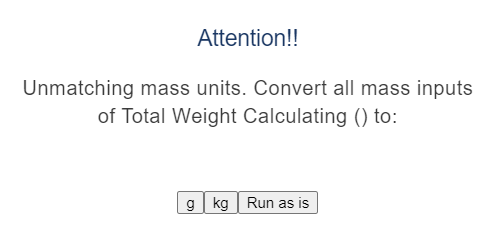


Figure 38: Popup for choosing the units that the user would like to use

If the user chooses **g**, the calculation is made in **g** and the result object's unit becomes **g** (Figure 39)

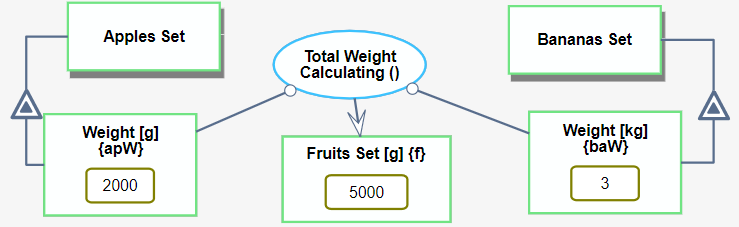


Figure 39: Execution with **g** units

If the user chooses **kg**, the calculation is made in **kg** and the result object's unit becomes **kg** (Figure 40).

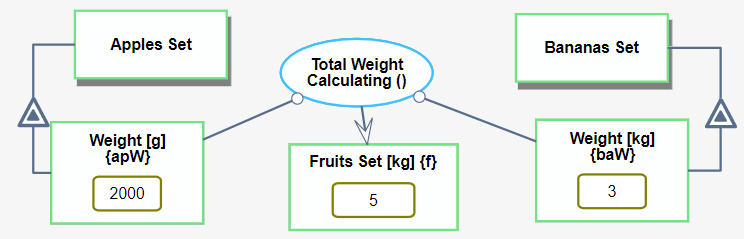


Figure 40: Execution with **kg** units

When **checkUnits** ends, the second function which is executed called **execute**. This function scans all processes in a Depth-First Search (DFS) order, performs the calculations that are written in the computational processes and stores the results of these computations in the logical representation of the result object. For both computational and conceptual processes the function stores the input and output links in dedicated arrays.

The code of **execute** function is shown in Function 1.

|  |
| --- |
| Function 1. Execute function  **function** *execute*(linksArray, initRappid, opdId) {  **const** currentOpdElements = initRappid.**opmModel**.getOpd(opdId).**visualElements**;  *// get all processes in the OPD* **let** currentOpdProcesses = currentOpdElements.filter(element => (element **instanceof** OpmVisualProcess));  *// sort processes from top to bottom* currentOpdProcesses = currentOpdProcesses.sort((p1, p2) => p1.getPosition().**y** - p2.getPosition().**y**);  *// go over all processes* **for** (**let** i = 0; i < processes.**length**; i++) {  *// If the user clicked on stopExecution during the execution*  **if** (initRappid.**Executing** === **false**) {  **return**;  }  **const** inbound = *getInboundLinks*(processes[i].**id**, initRappid.**opmModel**.getOpd(opdId));  *// if it is an in-zoomed process then need to go to the in-zoomed OPD and execute it first* **if** (processes[i].**refineeInzooming**) {  **const** inzoomedOpd = initRappid.**opmModel**.getOpdByThingId(processes[i].**refineeInzooming**.**id**);  *// recursive execution. now the in-zoomed graph will be executed* **await** *execute*(linksArray, initRappid, inzoomedOpd.**id**, runner);  } **else if** (processes[i].**logicalElement**.code !== code.*Unspecified*) {  **try** {  **await** *compute*(linksArray, processes[i], initRappid.**opmModel**.getOpd(opdId), initRappid);  } **catch** (e) {  *handleExeExceptions*(e.toString(), processes[i]);  } }  **const** outbound = *getOutboundLinks*(processes[i].**id**, initRappid.**opmModel**.getOpd(opdId));  **const** continueLoop = runner.checkInzoomedInvocation(outbound);  **if** (continueLoop && continueLoop.**length**) {  i = -1;  processes = continueLoop;  } } |

Finally, when **execute** function ends, the **showExecution function is executed**. This function scans the input and output links arrays, draws green tokens that move along the links to demonstrate execution and update computational result objects, the drawn part that is seen on the screen, with the result that was stored in the logical representation during the **execute** function. Each token runs for one second. The **showExecution** function traverses smoothly along the links demonstrating the seamless integration between the conceptual and computational parts of the model.

## **Stop execution**

During the showExecution function execution, while tokens are run and values of objects are updated, the user can click on the "stop" button, which appear on the execution menu next to the "play" button, to stop run tokens and restore all objects’ values to be as they were before the current, last execution.

By clicking on the "execute" sign, circled by a solid black line in Figure 33, the user exits the execution mode and all values restored to be as they were before entering this mode. Undefined values become again undefined.

## **Repeated execution**

The user can configure the system so that the execution will repeat more than one time by entering the number of runs in the execution menu shown in Figure 34. It is used mostly when the value of an object updated in each execution. All iterations will be calculated at the background, i.e., in the **execute** function and animates only the last iteration, i.e., **showExecution** function will update the final value. In Figure 41, the value **2** in object **Number** is multiplied by two four times using an effect link. Only the last time result is shown to the user and the value is updated on the screen to be **32**.

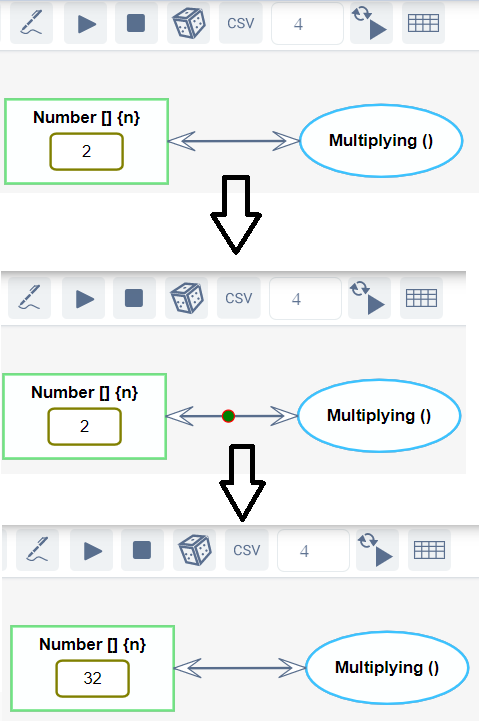


Figure 41: Executing **Multuplying()** process four times

## **Export results**

The results of the execution may be exported to Excel file. The file includes all computational objects and their values as shown in Figure 42.

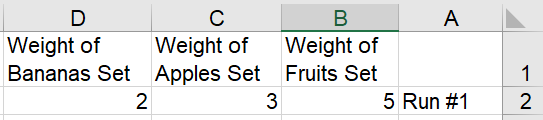


Figure 42: Export file example

In case of repeated execution, the user can choose to export the intermediate results to a new csv file every n runs. For example, for n=5 and number of total executions of 23, the result will be 5 files. The first, second, third, fourth and fifth files will include values of first 5, 10, 15, 20 and 23 runs respectively. The place for choosing the number of runs is opened after clicking on the csv button and shown in Figure 43.



Figure 43: place to choose the number n which means that every n runs a csv file exported

## **Import values**

For executing the model with different values, the user can import the values from Excel file. The names of the columns should be identical to the names or aliases of the computational objects. For doing it, the user should click on the rightmost button in Figure 43.

By enabling redundant execution and importing the input values from an Excel file, we provide an option for many executions that can be later analyzed to make better decisions regarding the model and errors that might be detected. This option enables diagnostic models, error detection models and big data analyzing.

# **Industrial Cooperation: A Running Case Study from The Aircraft Industry**

To demonstrate our integrated OPM-based hardware-software modeling methodology with MAXIM integrated into OPCloud, we use an OPM model of an Airbus aircraft braking system as a case in point. The development of the braking system requires involvement of engineers from various engineering disciplines: (1) mechanical engineers, for designing the **Pedal Pressing** process; (2) mechanical, electrical and software engineers, for the ABS system; (3) aerospace engineers, for the **Decelerating** process; and (4) human factor engineers to design and test the display of **Speed Indicating** on the Cockpit's **Display System**. These engineering domains have been using different languages and tools to model their parts of the system separately, while the computations were not modeled at all at this early conceptual design phase. Our goal is to model and execute the entire simulated system, integrating its conceptual and computational aspects in a single model. This enables discovering and investigating the interactions between different domains within the same system at this early design phase, in which discovering and correcting design errors is quick and inexpensive, preventing costly and disruptive changes downstream.

## **OPM Conceptual Model**

We start with the main process—the procedural part of the system’s function, called **Aircraft Braking**, as shown in Figure 44.

Figure 44: **Aircraft Braking** – the main process of the system’s function

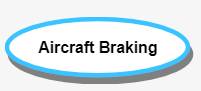


Figure 44: **Aircraft Braking** – the main process of the system’s function

The **Aircraft Braking** process is operated by a pilot, so we need an object called **Pilot**. The object **Pilot** is connected to the main process by an agent link (Figure 45). An OPM agent is a human or a group of humans that enable and are in charge of or control the process. In addition, **Aircraft Braking** requires a braking control system. To express this, we add an instrument link from the object **Aircraft Braking System** to the process **Aircraft Braking** (Figure 46). An instrument is a non-human enabler. The **Aircraft Braking** process affects the **Aircraft** itself, because it slows its speed, so we use an effect link to express this (Figure 47).

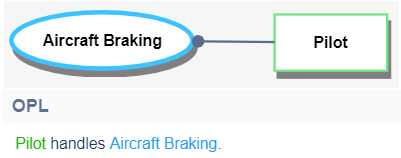


Figure 45: An Agent Link from the object **Pilot** to the process **Aircraft Braking**

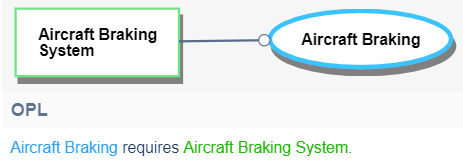


Figure 46: An Instrument Link fr om the object **Aircraft Braking System** to the process **Aircraft Braking**

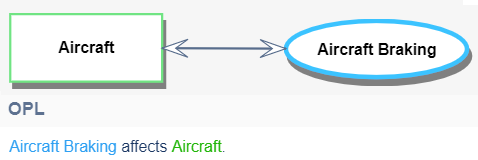


Figure 47: An Effect Link from **Aircraft Braking** to **Aircraft**

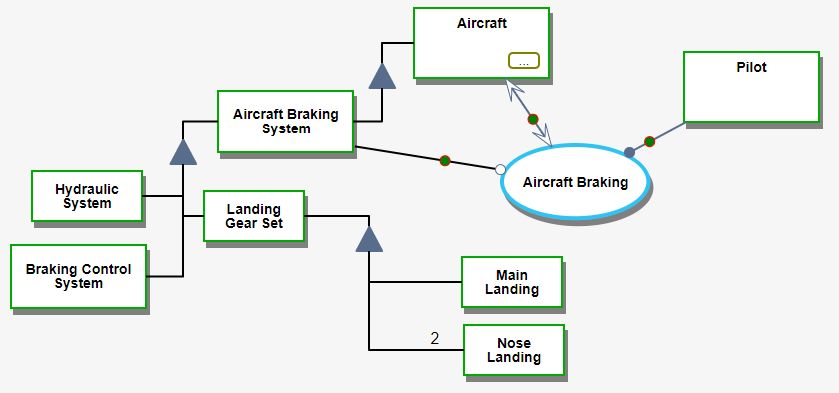
To describe the structure of the Aircraft Braking System, we use an aggregation-participation link, drawn as a black triangle, expressing that a whole object, connected to the tip of the triangle, consists of one or more other objects, its parts, which are connected to the triangle’s base. The same link denotes that **Aircraft Braking System** is part of **Aircraft**. The red-green circles shown in Figure 48 along the procedural links demonstrate the execution of the model, as explained in Section 5.3. 

Figure 48: SD – the top-level OPD of the **Aircraft Braking System**

After deﬁning the main function of the system, we reﬁne the resulting SD in Figure 48 by zooming into the **Aircraft Braking** process, getting SD1, as shown in Figure 49. In-zooming is an OPM refinement mechanism that allows the modeler to specify the subprocesses of a process and their temporal order, with the timeline going from top to bottom.

According to Figure 48, the **Pilot** handles **Commanding**, which generates a **Command** at one of its four possible states using a result link. Each state represents a possible command that the **Pilot** can give. At any point in time, **Command** can exist in exactly one of its states. Here we focus on modeling the **Pedal Braking** command, since the state of **Command** is **pedal braking**.

The **pedal braking state of the Command object** serves as a condition to start the process **Pedal Braking**. This semantics is expressed by the addition of the control modifier c next to the white lollipop of the link between pedal braking and Pedal Braking, denoting an instrument condition link. Following the condition link semantics, the remaining three subprocesses in Figure 49, **Auto-Braking**, **Park Braking** and **Aircraft Directing**, are skipped.

The **Pedal Braking** process, which is physical (as denoted by the shading of this process ellipse), transforms three objects: (1) it changes the state of the **Aircraft** from **taxiing** to **stopped**, (2) it updates the **Speed**, which is an attribute of **Aircraft**, as denoted by the exhibition-characterization link between them, and (3) it changes the state of **Status Display** from **no braking indication** to **pedal braking**. The physical process **Pedal Braking** transforms the conceptually modeled physical objects **Aircraft** and **Status Display**, as well as the computational object **Speed** of **Aircraft**. This is an example in which a physical process bridges the gap between conceptual and computational modeling constructs, connecting systems and software engineering in the same unifying OPM model. This fusion also helps validate the conceptual model and detect logical errors.

The last subprocess in SD1 is **Speed Indicating**, which uses (and consumes or eliminates) the value of **Speed**, **500 m/s**, and updates the value of the **Displayed Speed** attribute of **Primary Flight Display**, which is part of the **Cockpit Display System**.

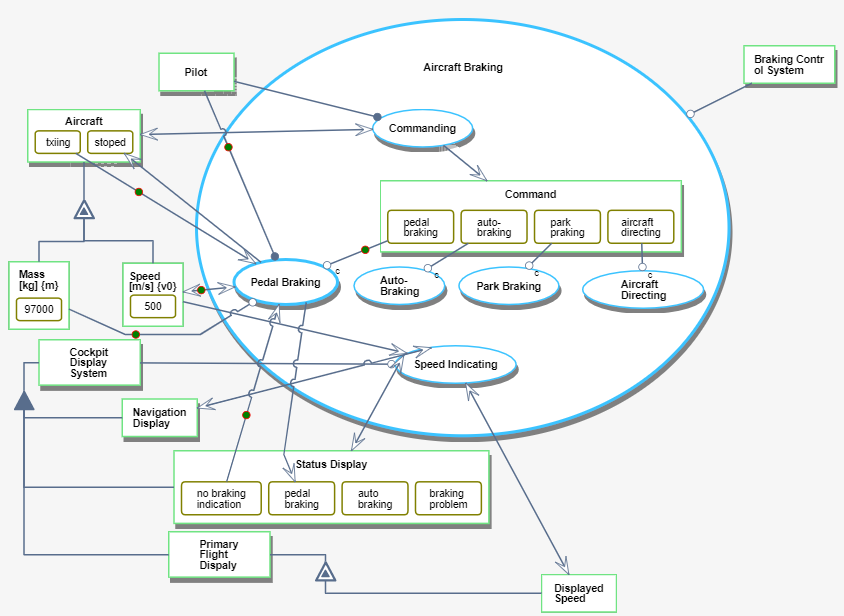


Figure 49: SD1 - **Aircraft Braking** in-zoomed. A Result Link from the **Commanding** process to the **Command** object indicates that **Command** is created in one of its four states. In this execution, the state is **pedal braking**, as shown by the circle along the link from that state to the **Pedal Braking** process.

To model finer details of the system, specifically the changing value of **Displayed Speed**, we must calculate the **Speed** of the **Aircraft** periodically, as it decelerates over time due to the **Aircraft Braking** process, till it becomes zero. Here we encounter for the first time the need for the new computational and model execution capabilities. In what follows, we elaborate on the computational aspect as part of the model execution.

## **Adding Computational Elements**

In a conceptual modeling language, such as UML or SysML, computation is not readily possible. There is an option to present equations in a SysML parametric diagram, but not to execute the model and actually compute values as they change during the execution. Therefore, to perform computations based on a conceptual model, we have to resort to using external tools such as MATLAB. MAXIM enables incorporating computations directly into the conceptual model with the same building blocks used in the conceptual model and verifying the model consistency at any time. If the need to update a conceptual part of the model arises, the computational part is updated automatically.

We wish to calculate the aircraft speed periodically, as it changes during the landing and the **Aircraft Braking** process. The modeled calculation must consider the **Press Force** and the **Press Angle**. Moreover, we want the calculated changing velocity to be displayed on the **Primary Flight Display** of the **Cockpit Display System**. This way, we handle concurrently the human factors aspect, demonstrating the viability of the model to integrate various aspects of the system. Additionally, we handle the mechanical engineering aspect, as we elaborate below. Traditionally, as argued, conceptual modeling for system engineering and computations for software engineering have been done separately, treating three different domains of the system in isolation:

(1) **Pedal Pressing** – This is the physical process done by the **Pilot**, who applies force on the braking pedal to stop the aircraft.

(2) **Braking Force Applying** – This is a process which combines physical and computational subprocesses as we show below when we zoom into this process. In OPM, physical essence is dominant over informatical essence, so when a process involves both physical and informatical (including computational) subprocesses, it is designated as physical. The same rule applies to objects comprised of physical and informatical parts. One of the subprocesses performs calculations of the force that is applied on the aircraft, the antilock braking system (ABS), and the new speed after a short period of time, which is repeated till the aircraft stops and the landing is complete.

(3) **Speed Indicating** – This is the physical process of displaying the new speed on the pilot’s display, so the pilot can accurately know the aircraft’s speed at any point in time during the aircraft landing and taxiing processes.

To model the combination of these domains—mechanical engineering, human factors engineering and computations—we zoom into the **Pedal Braking** process, which causes the creation of SD1.1 (Figure 50). Computations are done inside the model, reflecting part of the software engineering system aspect.

The first subprocess within **Pedal Braking** is **Pedal Pressing**, which the **Pilot** handles as the agent of the process. **Pedal Pressing** results in two values: **Press Force** and **Press Angle**. From this point on, we move from pure conceptual hardware modeling to integrated conceptual-computational modeling. Figure 50 shows the objects involved in the **Braking Force Applying** process: one enabler, two consumees – the inputs, and one resultee – the output. The enabler of **Braking Force Applying** is the (conceptual representation of the) physical **Brake Assembly**, which enables this process only if it is at state active, as expressed by the instrument link from the state to the process. The two consumees (consumed objects), **Press Force** and **Press Angle**, are informatical and quantitative. The result of this process is a computed **Braking Force**, which is used to compute the new **Speed** of **Aircraft** by **Decelerating**. The first computational function, **Decelerating (),** is based on Newton’s second law, as shown in Figure 51. **Mass** and **Time** are constants; **Speed** (both and ) is changing according to the formula in the **Decelerating ()** process. Therefore, both and which appear in the formula represented by **Speed** object, keep changing. Here we model just one iteration, where the time gap is 5 seconds.

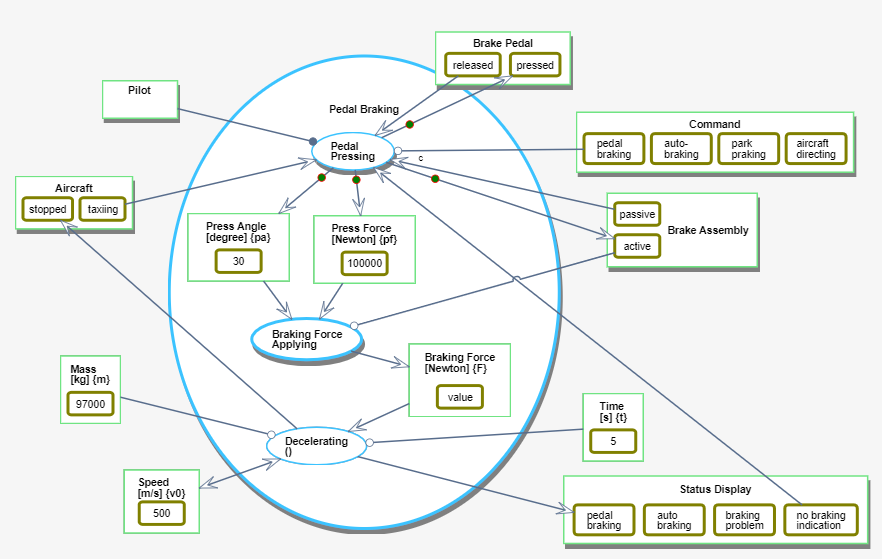


Figure 50: SD1.1 – **Pedal Braking** in-zoomed, showing three engineering domains in the same model and same diagram: **Pedal Pressing** is in the mechanical engineering and human factors engineering domain, **Braking Force Applying** is in the mechanical engineering domain, and **Decelerating** is in the computational domain.

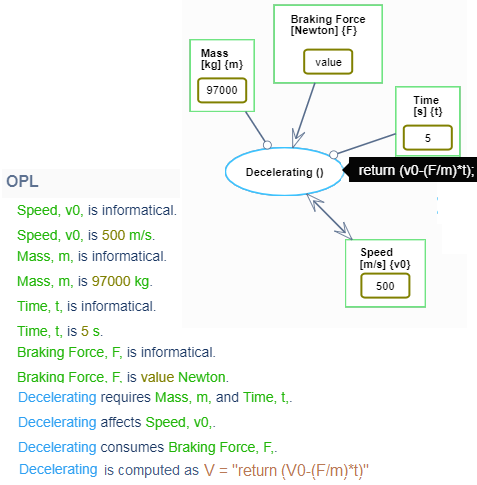


Figure 51: **Decelerating** with a tooltip showing its computational function **V=V₀-F/m\*t** and the informatical objects **Mass**, **Time**, and **Braking Force** with their units, used for the **Speed** calculation.

To obtain the value of **Braking Force**, we zoom into the **Braking Force Applying** process (Figure 52). First we see the conceptual process **Signal Detecting** which requires **Sensor Subsystem** object and put the result in an object called **Signal Set**. Next, we have an in zoomed process called **Anti-Lock Force Calculating** and finally the conceptual process **Actuating**.

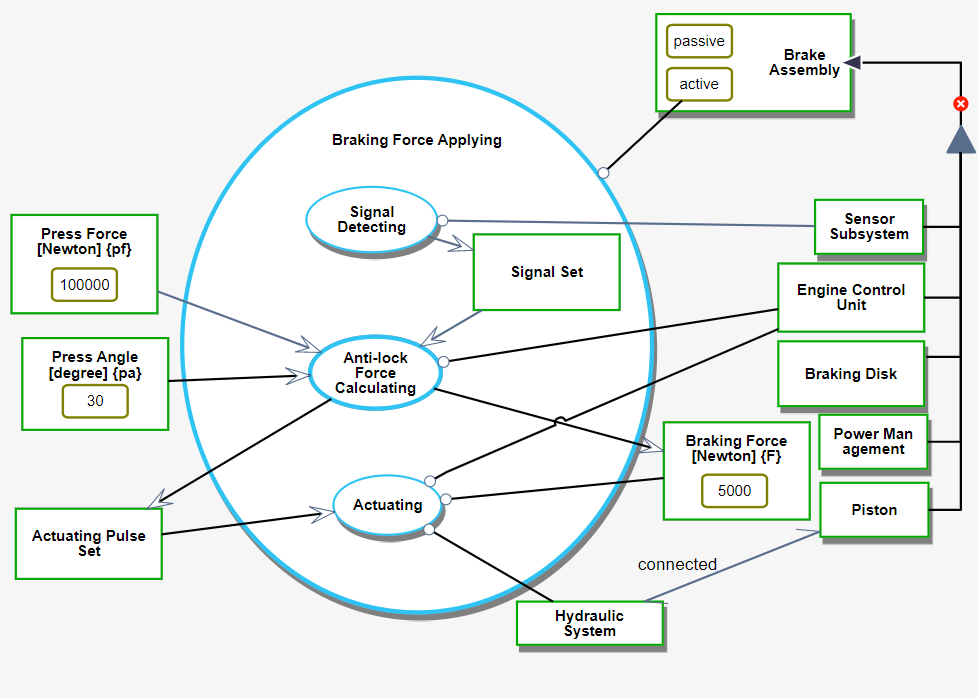


Figure 52: SD1.1.1 - **Braking Force Applying** in-zoom showing two conceptual processes and one that will be in-zoomed and calculate the **Braking Force**

Next, we focus on the **Anti-Lock Force Calculating** subprocess within the **Braking Force Applying** process (Figure 53). In this process we have five sub processes that allow us to see again the smooth integration between conceptual and computational modeling.

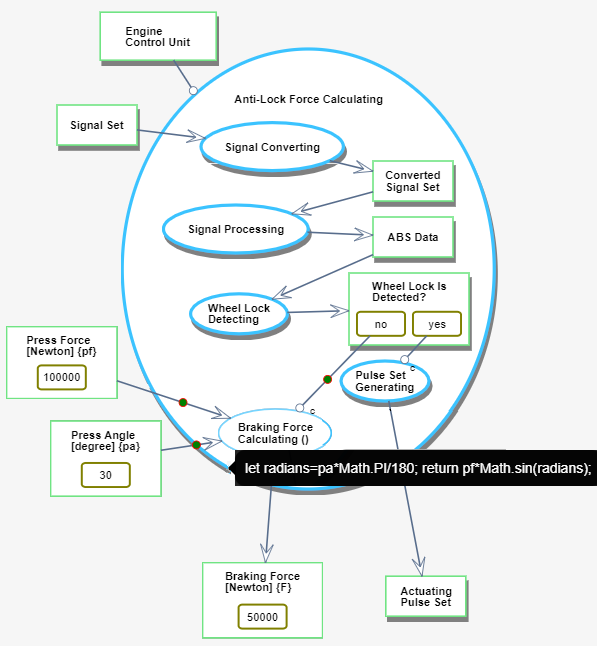


Figure 53: SD 1.1.1.1 - **Braking Force Calculating** is a computational process with input **Press Force** and **Press Angle** in degrees, and output **Braking Force** in **Newton**

The first four processes are pure conceptual while the last one is computational. In Figure 54 we see the in zoom of the first conceptual process named **Signal Converting**, in Figure 55 we see the in zoom of the second conceptual process named **Signal Processing**, in Figure 56 we see the in zoom of the third conceptual process named **Wheel Lock Detecting**, and finally in Figure 57 we see the in zoom of the fourth conceptual process named **Pulse Set Generating**. The fifth process is computational, where **Braking Force** is calculated, based on **ABS Data**: if wheel lock is not detected (**Wheel Lock is Detected?** is at state **no**), then the **Braking Force** is calculated, otherwise there is a wheel lock, which means that the aircraft is skidding on the runway. In this case, **Pulse Set Generating** is triggered, yielding **Actuating Pulse Set**.

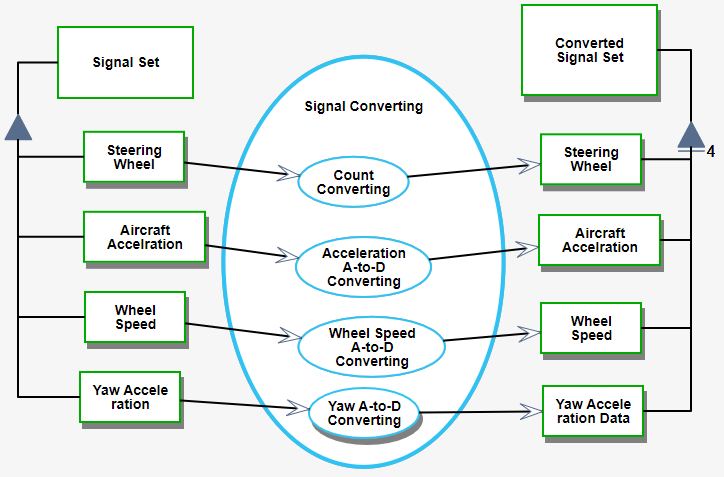


Figure 54: SD1.1.1.1.1 - **Signal Converting** in-zoomed

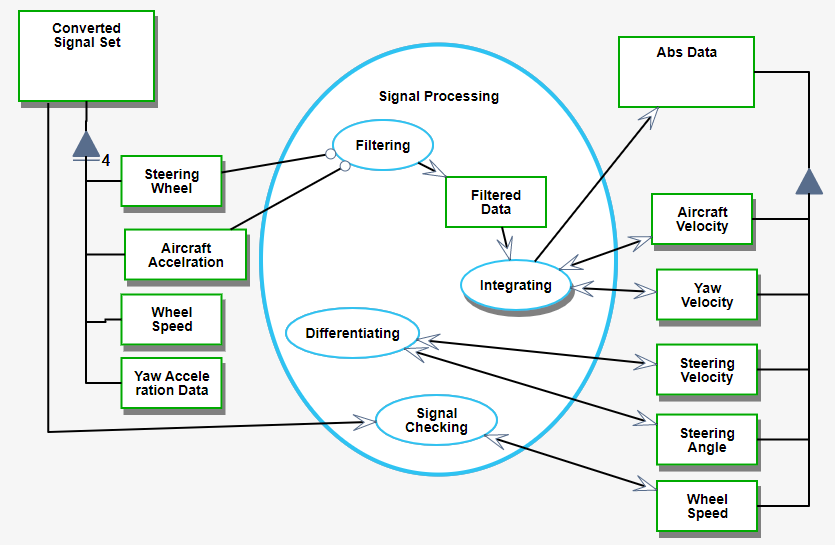


Figure 55:SD1.1.1.1.2 - **Signal Processing** in-zoomed

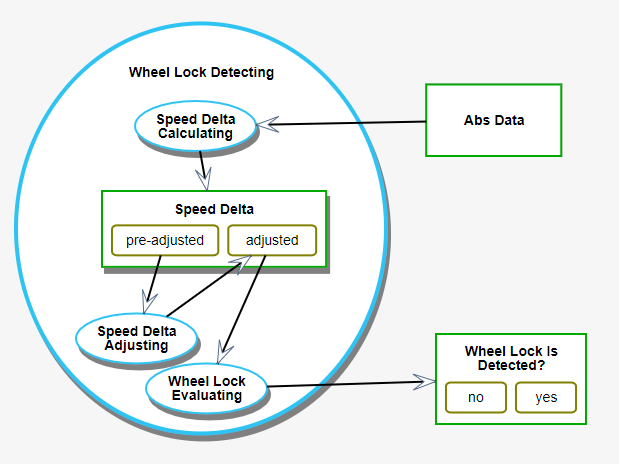


Figure 56: SD1.1.1.1.3 - **Wheel Lock Detecting** in-zoomed

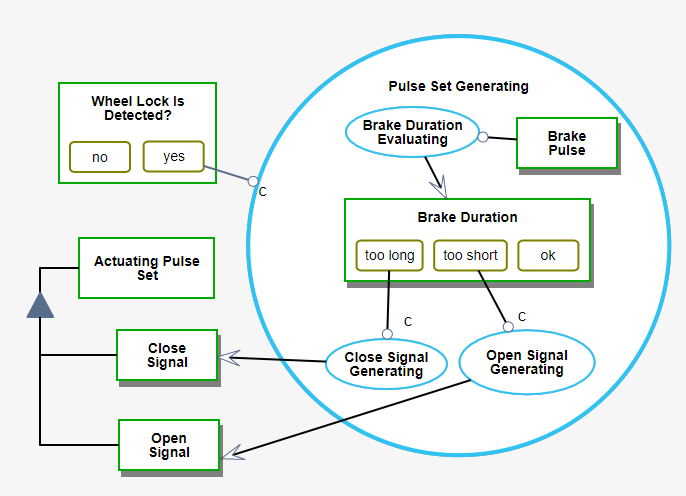


Figure 57: SD1.1.1.1.4 -**Pulse Set Generating** in-zoomed

Back to SD1.1, in Figure 58, the effect link between **Decelerating ()** and **Speed** means that **Decelerating ()** affects the value of **Speed**: it consumes the old (initial in the first iteration) **=500** and results in the new value, the velocity after 5 seconds,  **= 497.42**.

## **Model-Based Computation Demonstration**

To see the integrated model in action, in this section we execute the model, watch calculated values, and use them in more than one domain, such as displaying the speed on the **Cockpit Display System**, as shown in Figure 49. To visualize the execution, tokens are shown moving along the links as values are passed or as matter or signals are sent and received along those links. If the process is computational, the tokens can be thought of as transferring input data from a consumee (where the value is consumed, i.e., read and then deleted from the source) or from an instrument (as read-only data, i.e., where the value is read and remains recorded in the source) to the process, and output data from that process to the resultee – the resulting object, whose value is updated.

We start the execution at SD (System Diagram), the highest, most abstract OPD level, level 0, which is the root of the OPD tree. Three tokens, shown in Figure 48, are moving along the three links from the objects **Pilot**, **Aircraft** and **Aircraft Braking System** to the **Aircraft Braking** process.

At this point, the animated execution takes us through the OPD tree in a depth-first order. Hence, the next OPD to be visited is SD1. We skip several execution steps and jump directly to **Pedal Braking**. This process is handled by a **Pilot,** the agent of the process. It gets a **taxiing Aircraft**, i.e., **Aircraft** at state **taxiing**, a **no braking indication Status Display**, i.e., **Status Display** at state **no braking indication**, which conditionally requires a **pedal-braking Command** and affects the **Speed,** whose initial value is **500 m/s**. These four tokens are shown in Figure 49 as they move towards **Pedal Braking**. When all four tokens reach their destination – the **Pedal Braking** process, the depth-first traversal of the OPD tree continues, and the execution changes focus to the in-zoomed **Pedal Braking** process. The tokens continue moving according to the execution path. Focusing on the computation-related scenarios, in Figure 50, the tokens move from **Pedal Pressing** to two computational objects – **Press Angle** and **Press Force** – and two conceptual physical objects: **active Brake Assembly** and **pressed Brake Pedal**. This is the first time in which both conceptual and computational things (objects and processes) are involved in the execution. At this point, the object **Braking Force** has no value yet; this is going to change after the process is done executing, as we describe below. We skip a few more detail levels of the OPD tree and get to the first computational process – **Braking Force Calculating**, presented in Figure 53. The process gets two values: **30** degreesand **100000** Newton, and it calculates the **Braking Force**. Once **Braking Force** is calculated, a token moves along the result link from the **Braking Force Calculating** process to the **Braking Force** object, whose value is updated in the model. Once a value of an object is updated in some OPD, it is stored in the current model's logical layer, explained earlier, when we discussed the architecture of OPCloud. This value is now readily available, known and updated in all the Visual and Drawn appearances of the same object in all the model OPDs.

Having executed the computation and obtained a value for **Braking Force**, we continue the depth-first OPD tree traversal. We were at the leaf of the tree, so we now go back and climb to an upper-level OPD, SD1.1, in which the **Pedal Braking** process is in-zoomed. Now, **Braking Force** has a value, so the **Speed** can be calculated.

In Figure 58, we see that the **Braking Force** is updated to **5000** Newtonand the **Speed** was reduced from **500** m/sto **497.42** m/sduring **5** s due to the **Decelerating ()** process. Finally, the **Speed** is displayed on the **Cockpit Display System**, using the **Speed Indicating** process as shown in Figure 49.

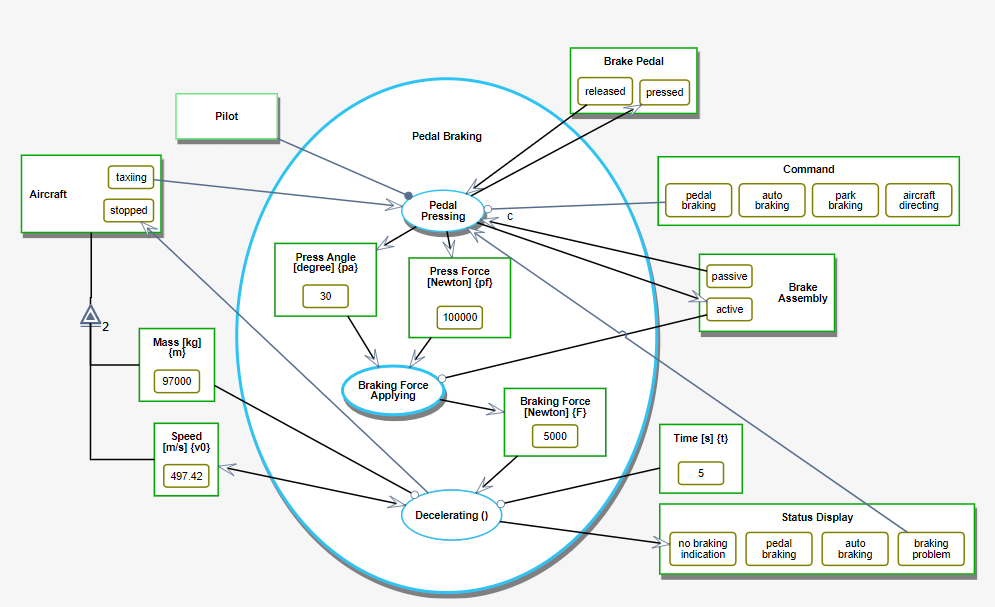


Figure 58: SD1.1 - After 5 s of **Decelerating**, the value of **Speed** is updated from **500.00 m**/s in Figure 37 to **497.42** m/s

# **Model Fidelity Hierarchy – The Landing Gear Case Study**

The landing gear is the part of the aircraft designed to enable its takeoff and landing. The latter requires absorbing great dynamic loads. A typical landing gear has three parts: one nose landing gear and two main landing gears. Each landing gear consists of a set of two or more tires and an oleo-pneumatic shock absorber strut. During aircraft landing, some of the loads are transmitted to the airframe through the landing gear. By determining the dynamic landing loads correctly, the landing gear weight can be reduced, enabling better aircraft performance.

What we have found after constructing the conceptual model was that the modeling process is also instrumental in finding in the free text document incomplete information and possibly contradicting requirements or wrong, unfounded basic assumptions.

## **First Fidelity Hierarchy Level: An OPM Conceptual Model**

We start with creating a high-level conceptual Vee model [26], [58], [59], [78] of the entire aircraft, which is circled by a solid black line on the left side of

Figure 59. We then focus on the Cargo Transporting process and the objects involved in it, circled by the dashed black line in

Figure 59. Next, based on a work we did about the Vee mode or an aircraft [26], we will gradually shift to modeling the landing gear.

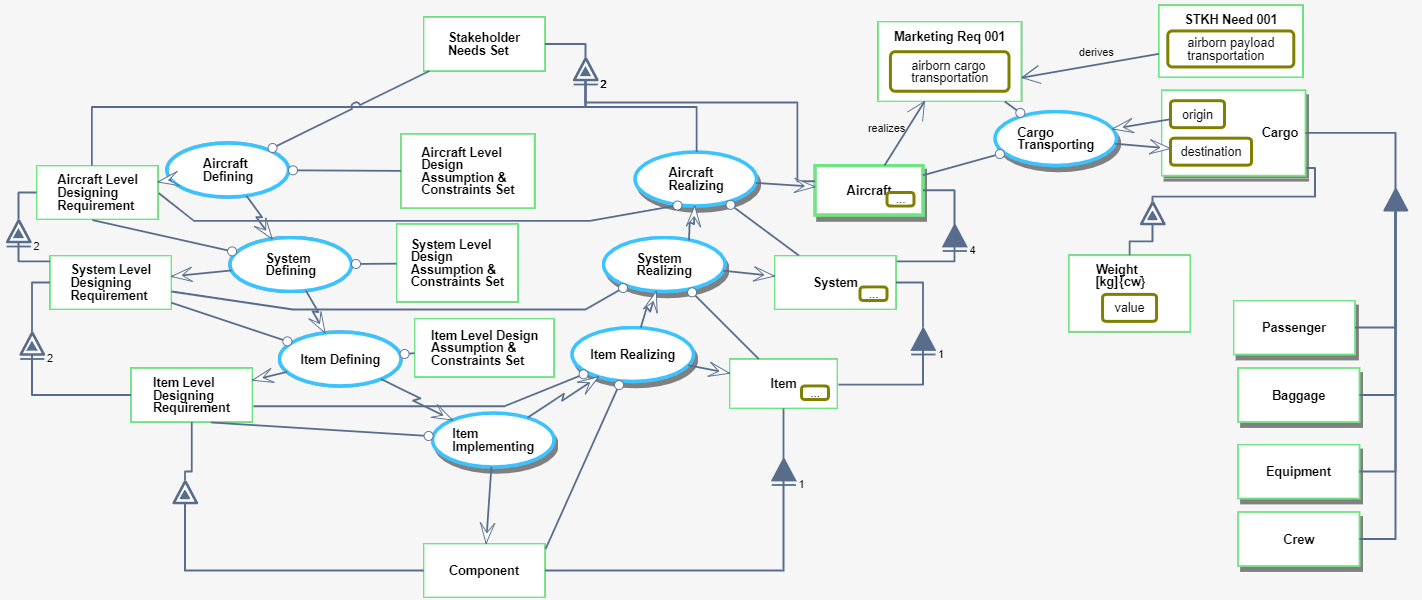


Figure 59. Integrating two OPM models into a single model

We first refine each of the processes in the OPM Vee model in

Figure 59 by zooming into it.

According to [26], the **Aircraft Defining** process is refined into five subprocesses, shown in Figure 60: **Aircraft Level Function Defining**, **Aircraft Level Requirements Identifying**, **Aircraft Level Requirements Allocating**, **Aircraft Designing**, and **Aircraft Level Requirements Validating**.

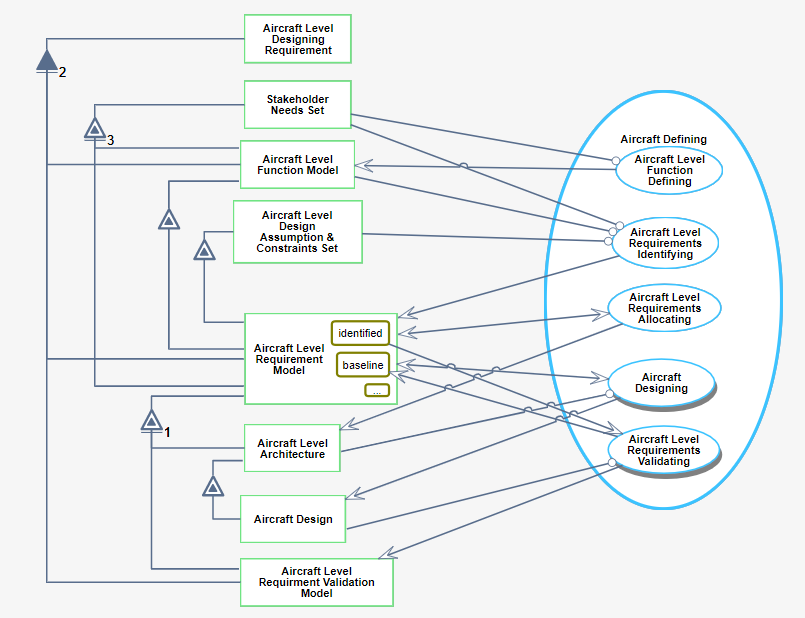


Figure 60: SD1 - **Aircraft Defining** in-zoomed

The **System Defining** process is also refined into five subprocesses, shown in Figure 61: **System Level Function Defining**, **System Level Requirements Identifying**, **System Level Requirements Allocating**, **System Designing**, and **System Level Requirements Validating**.

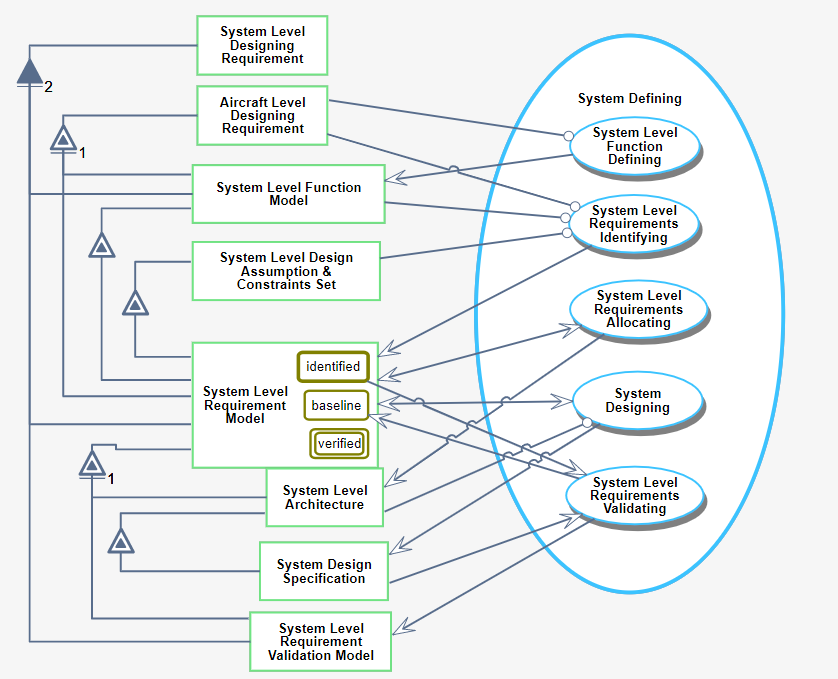


Figure 61:SD2 - **System Defining** in-zoomed

Next, in Figure 62, the **Item Defining** process from

Figure 59 is in-zoomed into five subprocesses: **Item Level Function Defining**, **Item Level Requirements Identifying**, **Item Level Requirements Allocating**, **Item Designing**, and **Item Level Requirements Validating**. After defining the processes according to the Vee model, we move to the implementation part.

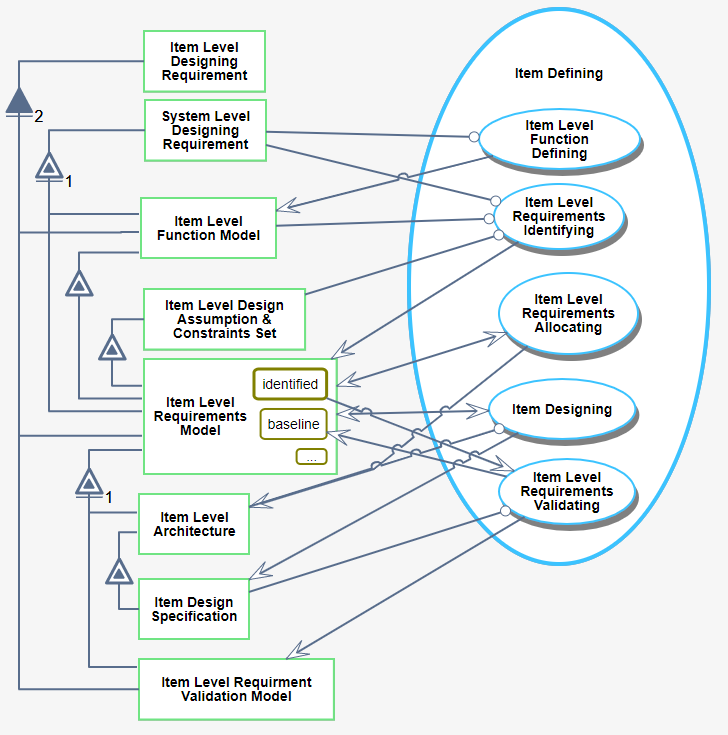


Figure 62:SD3 - **Item Defining** in-zoomed

We zoom into the **Item Implementation** process, shown in Figure 63, which is refined into two subprocesses: **Item Level Components Design** and **Item Level Components Manufacturing.**

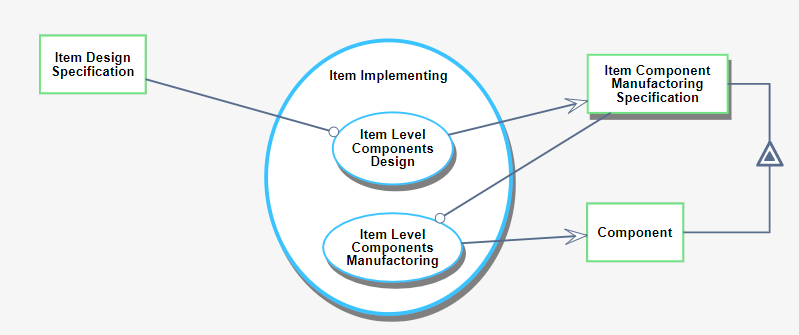


Figure 63: SD4 - **Item Implementing** in-zoomed

Next we move to three processes that are the right side of the V in the Vee model, each zoomed into three subprocesses. The First is **Item Realizing** that zoomed into **Item Level Component Assembling**, **Item Level Requirements Verifying**, and **Item Certifying** (Figure 64)**.**

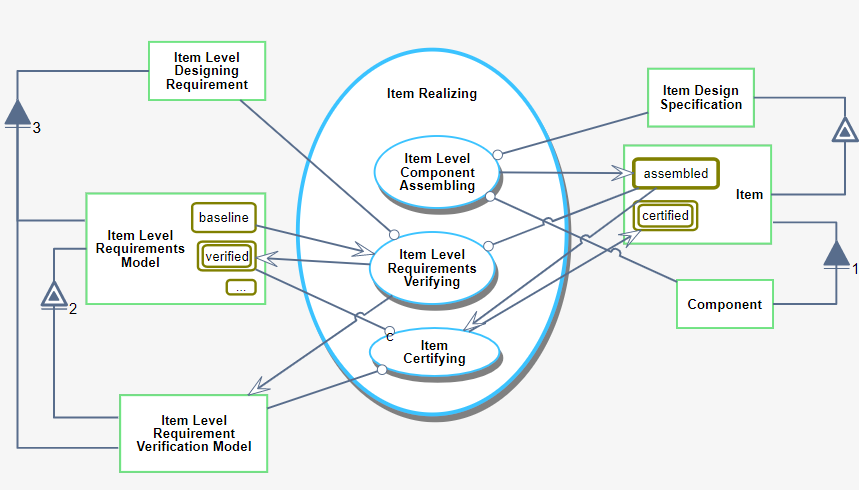


Figure 64: SD5 - **Item Realizing** in-zoomed

The second is **System Realizing,** that zoomed into **System Level Item Integrating**, **System Level Requirements Verifying**, and **System Certifying (**Figure 65**)**.

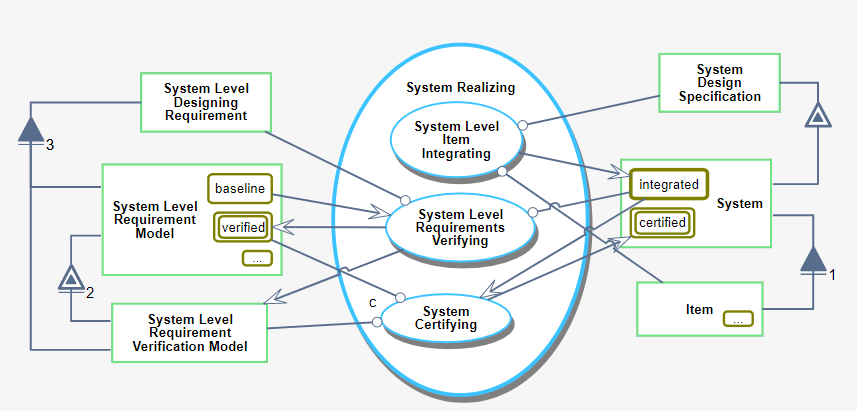


Figure 65:SD6 - **System Realizing** in-zoomed

And the last one is **Aircraft Realizing** that zoomed into **Aircraft Level Item Integrating**, **Aircraft Level Requirements Verifying**, and **Aircraft Certifying** (Figure 66).

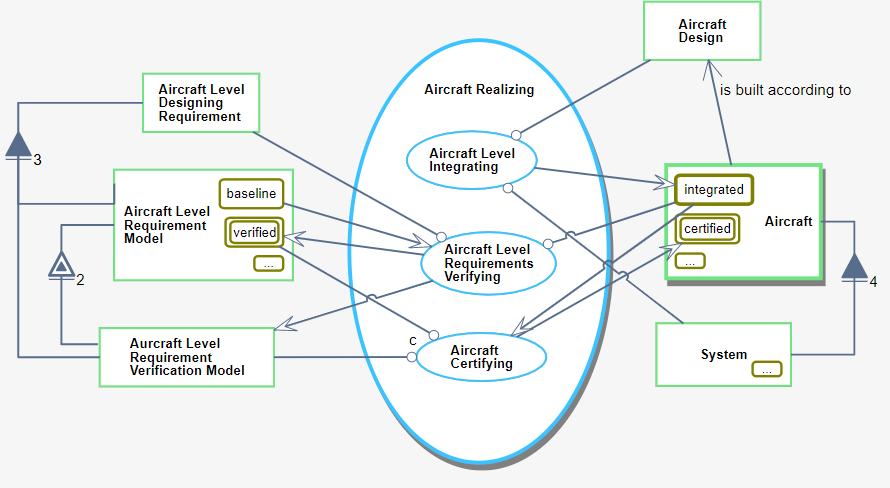


Figure 66: SD7 - **Aircraft Realizing** in-zoomed

In Figure 66, The object **Aircraft** has several states, two of which are visible: its initial state, **integrated**, and its final state, **certified**. This is expressed by the OPL sentences in Figure 67, which are generated automatically by OPCloud in response to the modeler’s graphic input.

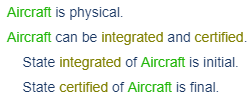


Figure 67. OPL sentences describing the initial and final states of **Aircraft**

The result of the **Aircraft Level Integrating** process is an integrated **Aircraft**. In order for the second subprocess, **Aircraft Level Requirements Verifying**, to start executing, **Aircraft** has to be at state **integrated**. This requirement is expressed by the instrument link from this state to **Aircraft Level Requirements Verifying**. **Aircraft Certifying** changes the state of **Aircraft** from **integrated** to **certified**. A new input regarding the aircraft is known at this stage – the **Aircraft** is built according to **Aircraft Design**.

In our other work [26], another, separate model was built in parallel to the Vee model for describing the **Cargo Transporting** process, circled in dashed black in

Figure 59. The purpose of that model was to describe the market need of a potential customer, identified as the object **STKH Need 001** (where STKH is an abbreviation for stakeholder), whose state airborne payload transportation expresses the content of the need. This need is expressed in marketing requirement terms as the object **Marketing Req 001**, whose value is **airborne cargo transportation**. The **Aircraft** realizes the formal need, and both are instruments for the **Cargo Transporting** process, which changes the state of the object **Cargo** from origin to destination. **Cargo** exhibits the attribute **Weight** and consists of four parts: **Crew**, **Passenger**, **Baggage** and **Equipment**.

While the Vee model (

Figure 59 left) and the **Cargo Transporting** model (

Figure 59 right) were built as two separate models, the fact that they have a common object, **Aircraft** (and its parts), has enabled us to integrate them into a single model.

We zoom into the **Cargo Transporting** process, as show in Figure 68, and see three subprocesses: **Aircraft Take-off Operating**, **Aricraft Cruising**, and **Aircraft Landing**. The first process, **Aircraft Take-off Operating**, gets the cargo at the origin, expresses by a consumption link from **Cargo** object at state **origin**. For performing this operation, required a **Landing Gear** and **A/c Req 001**, which are connected to the process by instrument links. The **Landing Gear** is part of the **Aircraft** who's move from the origin to the destination, expressed by changing the state of the **Aircraft** object from **origin** state to the **destination** state, by the second process, **Aircraft Cruising**. The **Aircraft** object has seven attributes that are connected to it by a double triangle and here we see at the first time a computational object in the model: **Weight**. The value of it is **71000**, the units are **kg** and the alias is **w**. The last process in this OPD, **Aircraft Landing**, requires the **Landing Gear**, **A/c Req 001**, and **A/c Req 002** and changes the state of the **Aircraft** from **in air** state to **on ground** state.

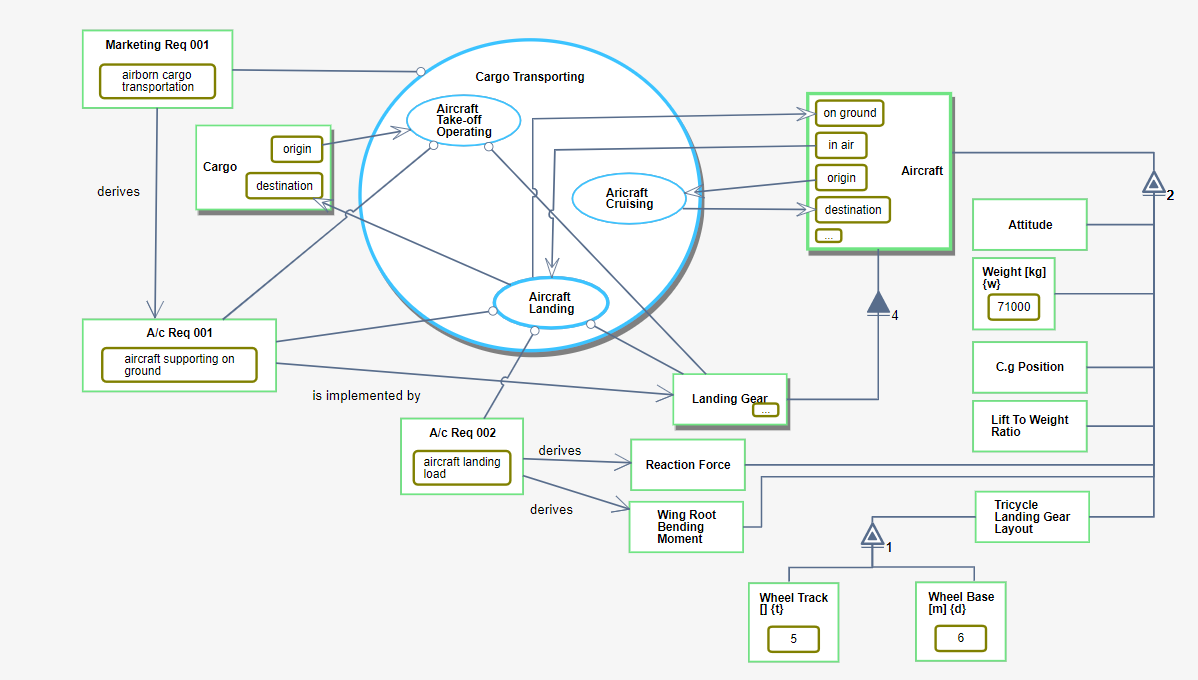


Figure 68: SD8 - **Cargo Transporting** in-zoomed

As our purpose is calculating the landing gear parameters, we zoom into the **Aircraft Landing** process (Figure 69), revealing two subprocesses: **Touchdown Landing** and **Landing Shocks Absorbing**. The result of the **Touchdown Landing** process is **Aircraft** at state **on ground**. The **Landing Shocks Absorbing** process requires **A/c Req 001**, **A/c Req 002**, **Sys Req 001**, **Sys Req 002**, **on ground Aircraft** and **Landing Gear** objects that are connected to it using an Instrument link. The process changes the state of the **Landing Gear** from **extended** to **compressed** and effects on the **Oleo-pneumatic Shock Absorber Strut** and the **Tire** which are parts of the **Landing Gear**.

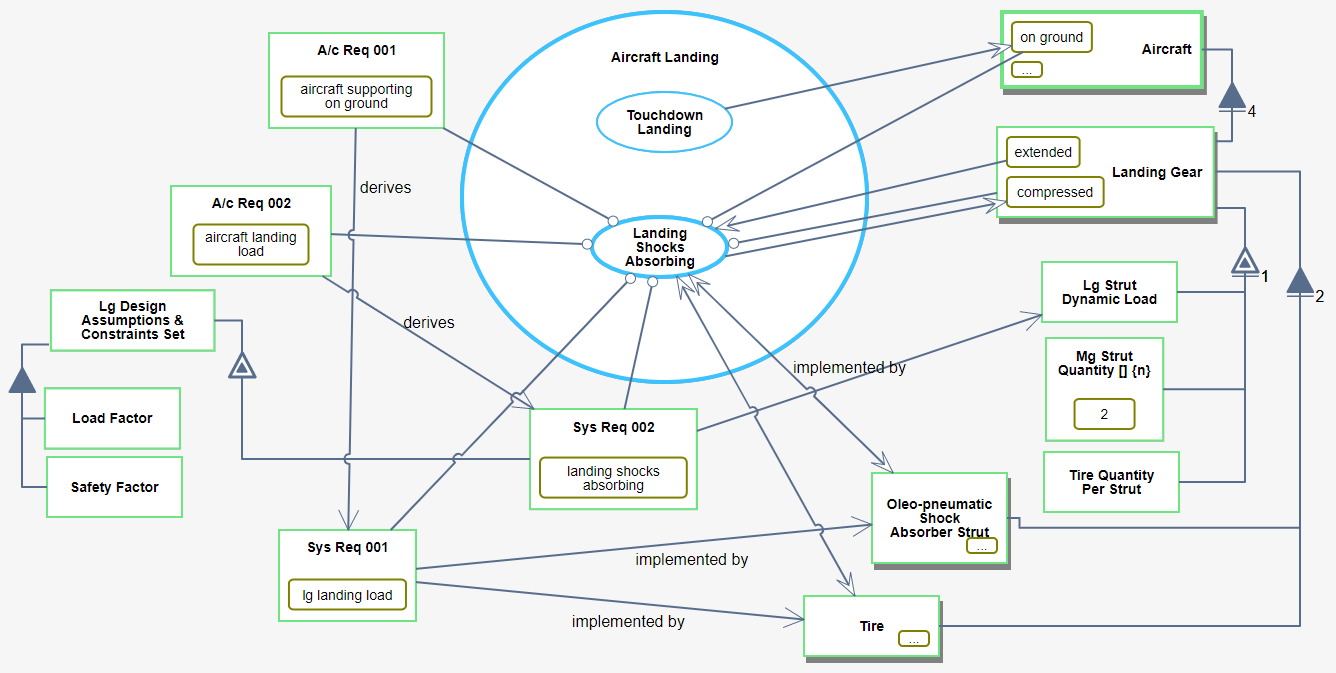


Figure 69: SD8.1 - **Aircraft Landing** in-zoomed

At the next stage we zoom into the **Landing Shocks Absorbing** process and we get to the most refined level presented in our previous work [26], which shows in Figure 70. In this doctoral work we focus on the **Energy Storing** subprocess of this process.

The **Energy Storing** process changes the state of the **Tire** from **inflated** to **compressed**. The instruments of this process are **Nose Tire** and **Main Tire**, which are parts of **Tire**, as expressed by the OPM aggregation-participation links from **Tire t**o **Nose Tire** and **Main Tire**. As explained in Section 6.2, we later realized that the aggregation-participation relation between **Tire** on one hand and **Nose Tire** and **Main Tire** on the other hand were incorrect, because the **Nose Tire** and the **Main Tire** are specializations of **Tire**, not parts of it, so a generalization-specialization relation (which gives rise to inheritance) should be used. Moreover, we modeled **Tire** as an object that exhibits five attributes: **Nominal Diameter**, **Wheel Nominal Diameter**, **Load Rating**, **Stiffness Coefficient**, and **Tire Pneumatic Spring Force**. In Section 6.2 we explain that this part of the model is problematic as well. Finally, the **Tire** and the **Oleo-Pneumatic Shock Absorber Strut are both** parts of the **Landing Gear**. The model contains additional objects and their attributes, but considering the main purpose of this section, which is expressing the model fidelity hierarchy by using computations and execution of a model, we focus on the ones specified above.

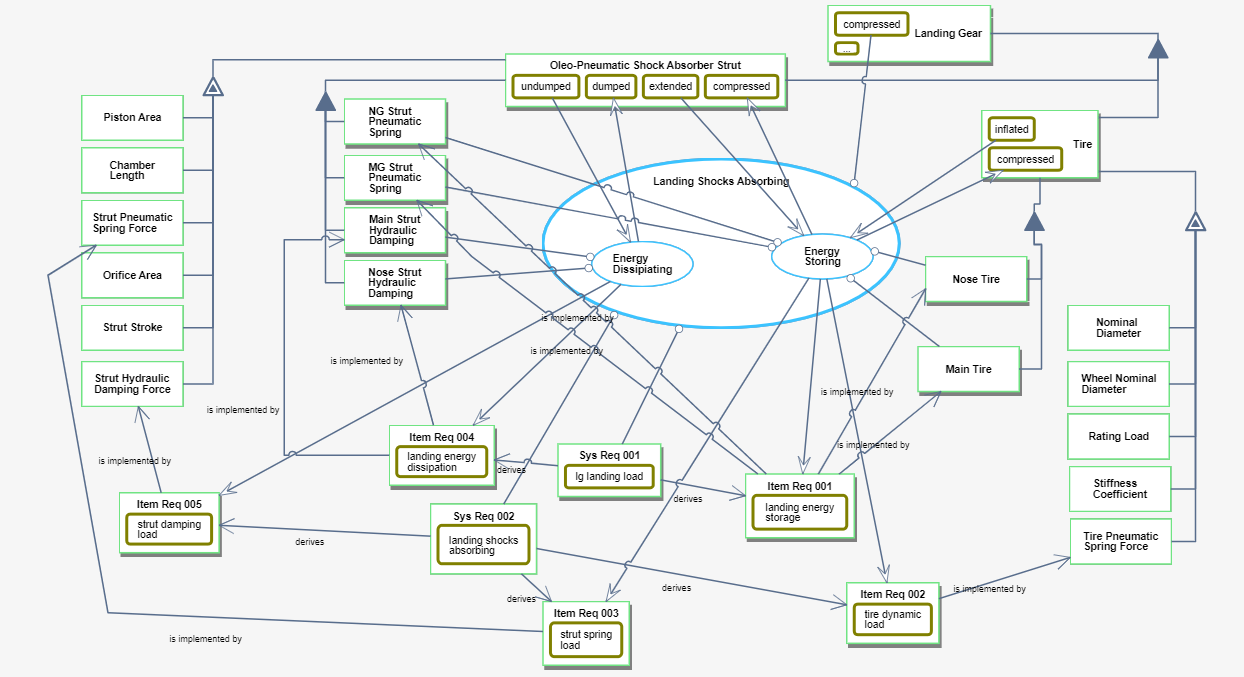


Figure 70. The **Landing Shocks Absorbing** process in-zoomed

## **Second Fidelity Hierarchy Level: An OPM Computational Model Extension**

In this section, we continue building our model by adding to it the computational part, including adding values to attributes and executable or computable functions as code to leaf processes. Based on [26] as a starting point, our initial goal was to integrate the computations which were made previously in Matlab into our qualitative model by expanding it with the computational and executable capabilities. These calculations aim to calculate the static load on the aircraft landing gear’s tire, considering the aircraft and its landing gear parameters in a single model. The calculated parameters are needed for selecting the most suitable tires. Having selected the tire type, the tires’ reaction forces can be calculated and used further to determine constraints on the struts. These, in turn, are needed to determine the maximal allowed cargo weight.

Since the formulas were already known and we modeled all the required parameters (see Figure 70), initially, this calculation task seemed to be relatively easy. However, when we started to assign values to computational objects and functions to computational processes in our model, we realized that the model was neither accurate nor sufficiently detailed, and we had to correct and refine it. Our experience has been that following the addition of computational capabilities, logical errors and inaccuracies in the conceptual model are discovered, because as numerical values or parameters that are used in equations are added, there is no room for vague or incomplete definition of model parameters. In what follows, we describe examples of errors that we found out and changes we had to make in the model in order to make it computable and executable. These unfolding revelations triggered the idea of the model fidelity hierarchy, which became the higher-level focus of this research.

## **Static load and load rating calculation**

Our first goal was to calculate the static load on the landing gears. To do this, we use two different equations [26]: one for the nose landing gear (Equation 1) and the other for the two main landing gears (Equation 2), where is the **Wheel** *Base*, is the center of gravity position, and is the **Weight** of the aircraft. For safety, the maximum weight before flight (with fuel) is taken as =71,000 kg.

Equation 1. The **static load** of the **nose landing gear**.

Equation 2. The **static load** of the **main landing gear**.

After understanding the required calculations, we realized that there is a need to compute two different static loads: one for the two (identical) main landing gear and another for the nose landing gear. As the static load is an attribute of **Landing Gear**, we needed to add two objects to represent the two kinds of landing gear: **Main Landing Gear** and **Nose Landing Gear**. As noted, in the other work [26] we performed the calculations in Matlab, separately from the conceptual model. Therefore, the absence of these two objects was not noticed. The integration of the computations into the conceptual model was not possible prior to conceiving how conceptual and computational modeling can be integrated into, and performed by, the same object-process modeling framework. The idea for doing this was to treat mathematical functions used for computations as informatical processes, and the input and output parameters of these functions as (informatical) objects, which are attributes of physical objects in the engineering problem. Further, this underlying integration idea needed to be implemented in the OPCloud software environment before we could use it in our Landing Gear problem as a case in point. We started refining the conceptual model part, presented in Figure 70, by adding these objects and connecting them to **Landing Gear** by a generalization-specialization link (white triangle), as shown in Figure 71. After adding the objects **Main Landing Gear** and **Nose Landing Gear**, we realized that the objects **Main Tire** and **Nose Tire** should be parts of the **Main Landing Gear** and the **Nose Landing Gear**, respectively, so in Figure 71 we added two aggregation participation links: one from **Main Landing Gear** to **Main Tire** and another one from **Nose Landing Gear** to **Nose Tire**. Finally, we changed the aggregation-participation links from **Tire** to **Main Tire** and **Nose Tire** to be generalization-specialization links, since the main and nose tires are kinds, not parts, of **Tire**.

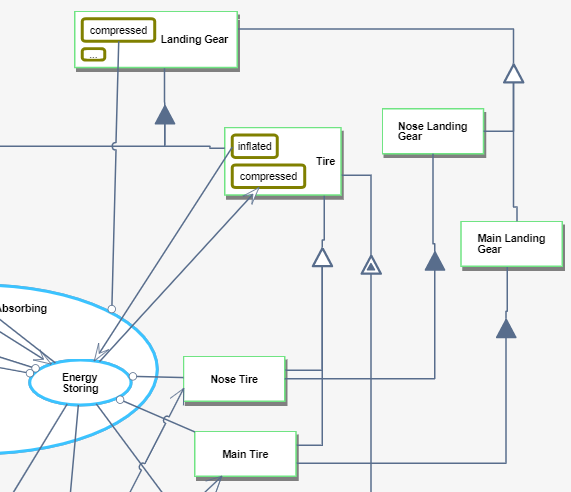


Figure 71. The fixed part of Figure 70, where we added **Main Landing Gear** and **Nose Landing Gear** as specilizations of **Landing Gear**,and refined the links

Having made these changes, we could create a refined OPD by zooming into the **Energy Storing** process and adding the first subprocess, **Landing Gear Parameters Defining**, which was supposed to calculate the static load. We then zoomed into that process and started to get the OPD in Figure 72. For the **Nose Static Force Calculating ()**, where the **()** symbol signifies a computational process, we used **Fwd CG Position** with the alias **A**, unit **m** (meter), and value **5**, **Wheel Base** with alias **D**, unit **m** and value **5**, and **Weight** of **Aircraft** measured in **kg**, with the alias **W** and value **71000**. The result of this process will be assigned as the value of the object **Nose Static Load** with units **kg** and alias **Fsn**. For the **Main Static Force Calculating ()**, we use **Aft CG Position** with the alias **A**, unit **m** and value **5.4**, **MG Strut Quantity** with alias **N,** which is unitless and has the value **2**, and with the same **Weight** of **Aircraft** and **Wheel Base** used for **Nose Static Force Calculating ().** The result of this process will be assigned as the value of the object **Main Static Load** with units **kg** and alias **Fsm**.

As the first subprocess, **Landing Gear Parameters Defining** is completely modeled and we have the results of **Nose Static Load** and **Main Static Load** (the main and nose static loads, respectively), we can now add the second subprocess, **Tires Selecting**. The purpose of this subprocess is to calculate the load rating of each tire, and based on the computed tire load rating results, the appropriate tires can be selected for the nose and main landing gears. As shown in Figure 73, the **Load Rating** is a function of the static load, and although it is calculated by using the same formula (Equation 3), the **Load Rating** of the **Main Tire** is different from that of the **Nose Tire**, because the static load input for the nose landing gear is different than that for the main landing gear.

Equation 3: The **load rating** formula. **Fs** is the **static load** (**nose** or **main**)

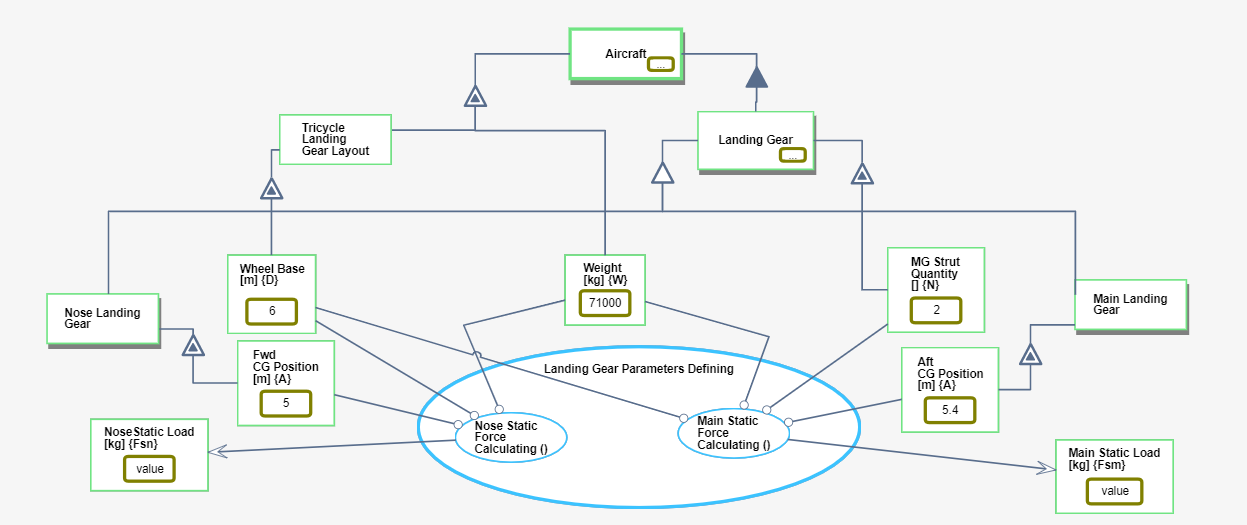


Figure 72. **Landing Gear Parameters Defining** in-zoom for calculating the **Static Load** for each landing gear

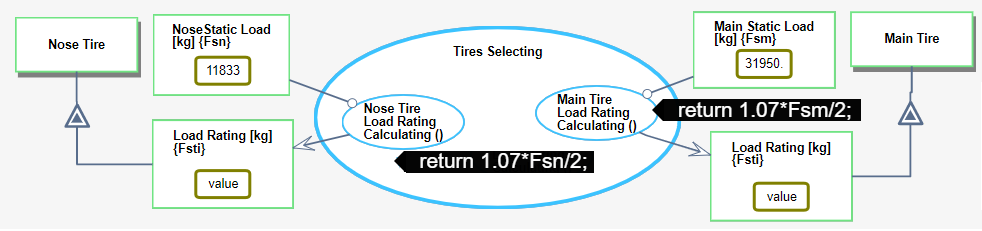


Figure 73. **Tires Selecting** in-zoom for calculating the required **Load Rating** for tires

## **Tire parameters defining**

The third and last process of the in-zoomed **Energy Storing** process is **Tire Parameters Defining**. A computational process is a leaf process; it cannot be further refined. As shown in Figure 74, **Tire Parameters Defining** is an informatical process, but not a computational one, as it is further refined into four computational (and hence leaf-level) subprocesses: **Nose Tire Stiffness Coefficient Calculating ()**, **Main Tire Stiffness Coefficient Calculating ()**, **Nose Reaction Force Calculating ()**, and **Main Reaction Force Calculating ()**.

Calculating the stiffness coefficients requires the wheel and tire nominal diameters and load ratings, which are different for the nose and main landing gears, as shown in Equation 4, whereis the **Load Rating** of the **Tire**, is **Wheel Nominal Diameter** and is **Tire Nominal Diameter**.

Equation 4: The formula for calculating the **Stiffness Coefficient**

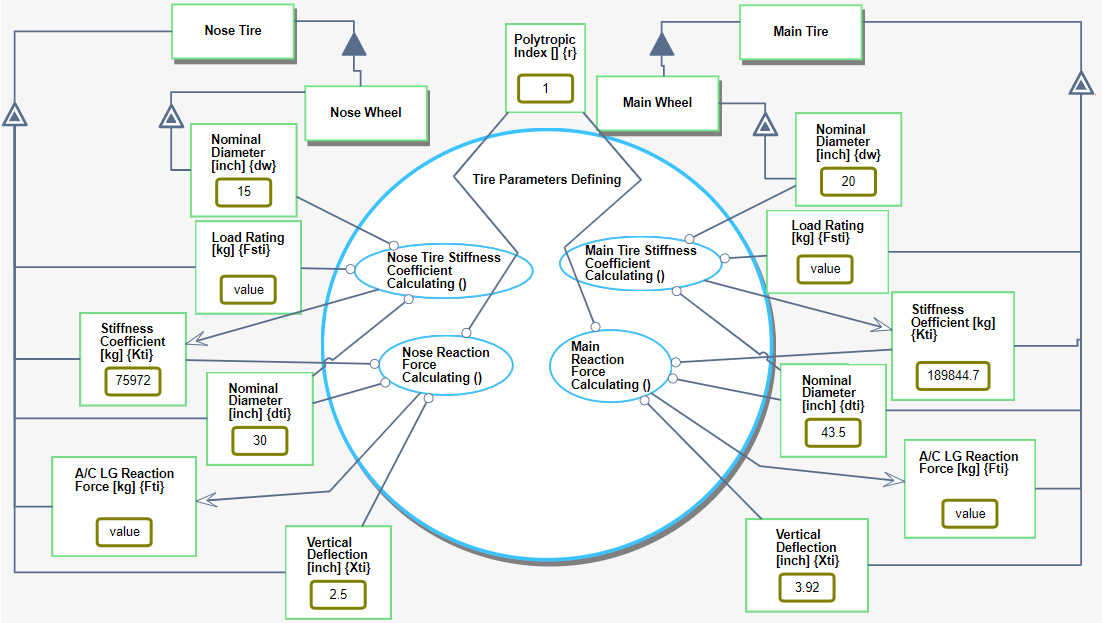


Figure 74. **Tire Parameters Defining** in-zoomed

Therefore, when we started to model the process **Stiffness Coefficient Calculating ()**, we realized that our conceptual model has a major flaw! According to Figure 70, all the five attributes of the tire—**Nominal Diameter**, **Wheel Nominal Diameter**, **Stiffness Coefficient**, **Load Rating**, and **Tire Pneumatic Spring Force**—must not be connected directly to the object **Tire**. Rather, these five **Main Tire** attributes are different than the **Nose Tire** ones, and therefore in the model they must be separated too! Following this new insight, we went back and fixed the conceptual model part by deleting these five tire attributes from the OPD in Figure 70 and adding them correctly twice, once for **Nose Tire** and once for **Main Tire**, as shown in Figure 74.

## **Third Fidelity Hierarchy Level: OPM Computational Model Execution**

So far, we accounted for modeling errors that were detected thanks to the need to specify the precise calculations and integrate them into the conceptual model. These errors were found while adding to the purely conceptual model the computational things (objects and processes) needed for calculating the landing gear parameters. At this point, one might think that the model is complete, because it has everything it takes to compute the model parameters. Moreover, the model also seems to be correct, because it contains the proper equations provided by the expert aeronautics engineer. However, it turns out that even at this advanced stage, the model contained errors, albeit of a different nature, which escaped being detected. These errors could only be found and corrected during execution, as they stem from wrong values given initially to the parameters, as we describe next.

Having completed the model and prepared it for calculations of the various landing gear parameters, which are done during the model execution, we assigned the parameter values to all the input objects by typing them directly into the value slots in the OPM model using OPCloud, and clicked to execute the model in order to get the desired results. As the execution was running, we suddenly saw that while the **Main Static Load** is calculated, providing reasonable result, the **Nose Static Load** is zero (see bottom left in Figure 75), which makes no sense. We quickly found out that both **Wheel Base {D}** and **Fwd CG Position {A}** were assigned the value **5**. The formula for calculating the **Nose Static Load** is shown in Equation 1, and when the model subtracts **Fwd CG Position {A}** from **Wheel Base {D}**, the result is zero. The value of **Wheel Base {D}** had to be **6**. With the correct **Wheel Base** value, the execution ran smoothly, yielding the desired parameter results of the nose and main landing gears and tires, as shown in Figure 77.

Our takeaway lesson from this incident is that just as integrating computations into the conceptual model helps us find errors in the conceptual model, model execution helps us find errors in computations – in this case it was the initialization of the object **Wheel Base**.

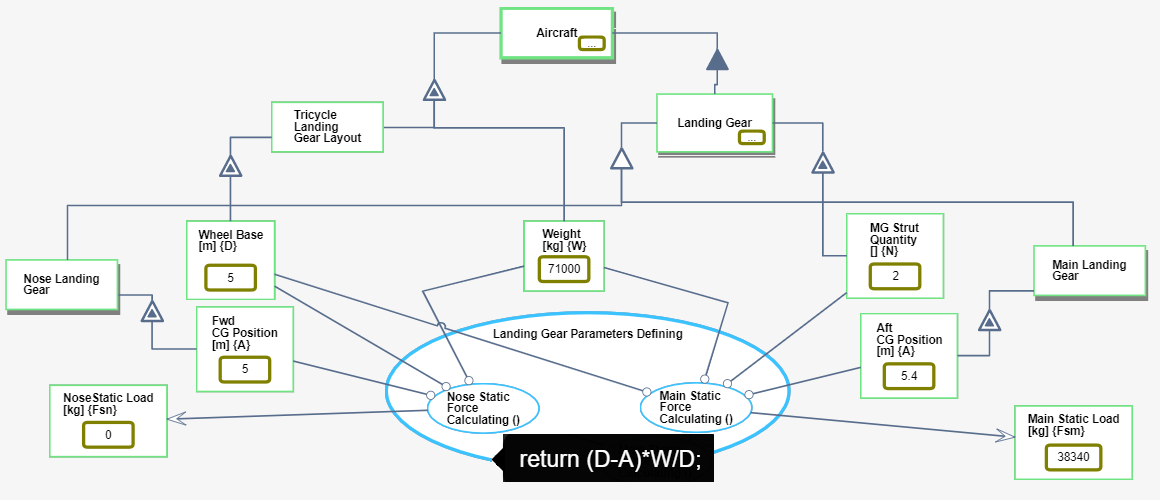


Figure 75. The value of **Main Static Load {Fsm}** is **38340** but the value of **Nose Static Load {Fsn}** is **0**

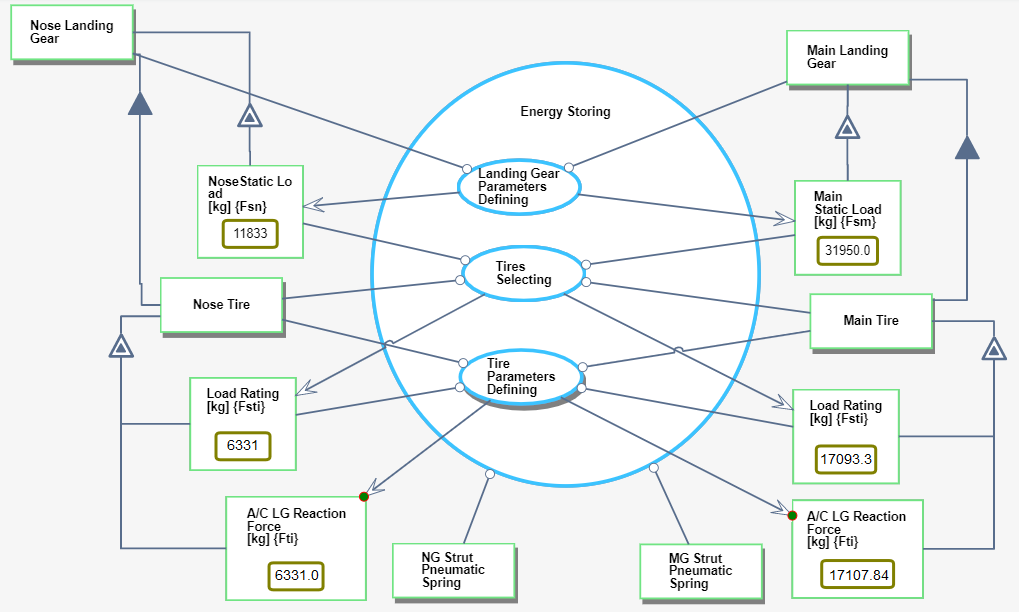


Figure 76. calculated parameters of the **Nose** and **Main Landing Gear** and **Tire**

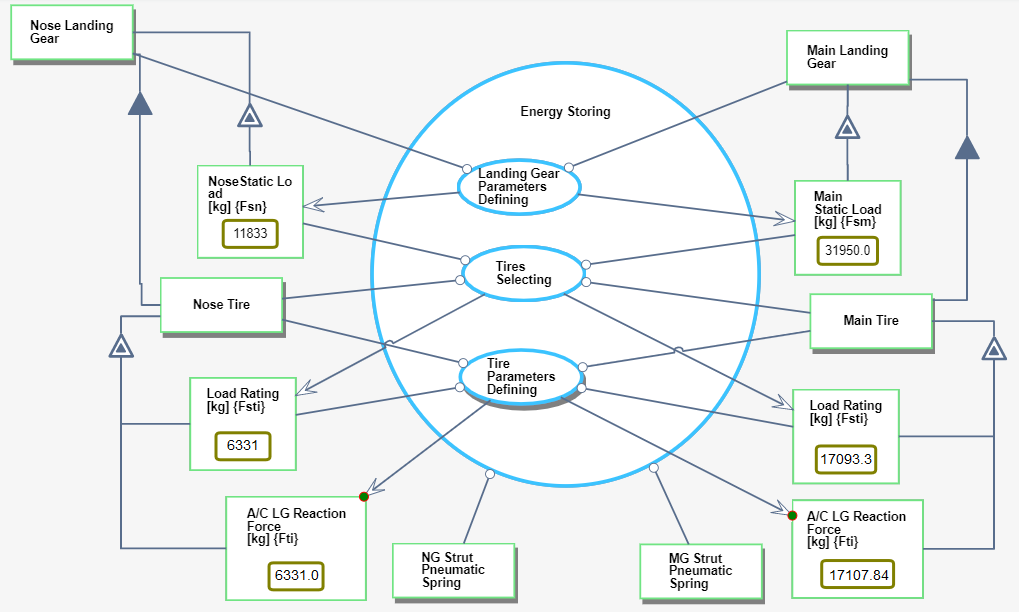


Figure 77. calculated parameters of the **Nose** and **Main Landing Gear** and **Tire**

# **Diagnostic Model – FTT Case Study**

For best diagnosis of FTT required data from two different periods: the perinatal period – the period of the pregnancy and immediately after the birth, and the postnatal period – the period from birth until the age of five years. These two periods are monitored by different medical staff - the first by a pregnancy escort nurse while the second by a pediatrician or by child development center. The data from these two periods is stored in different databases and not integrated at all. Moreover, no one of these staff has an FTT knowledge and can't diagnose FTT, or suspect one, until it is in a very progressive stage.

In FTTell, our integrated diagnostic tool, we use data from both periods, insert computations using MAXIM for calculating parametric data and execute the model, using MAXIM as well, for getting a diagnosis.

We start constructing our model by defining the main process, **Failure To Thrive (FTT) Diagnosing & Treating**,which is physical, as denoted in Figure 78 by the shading of the ellipse representing this process.

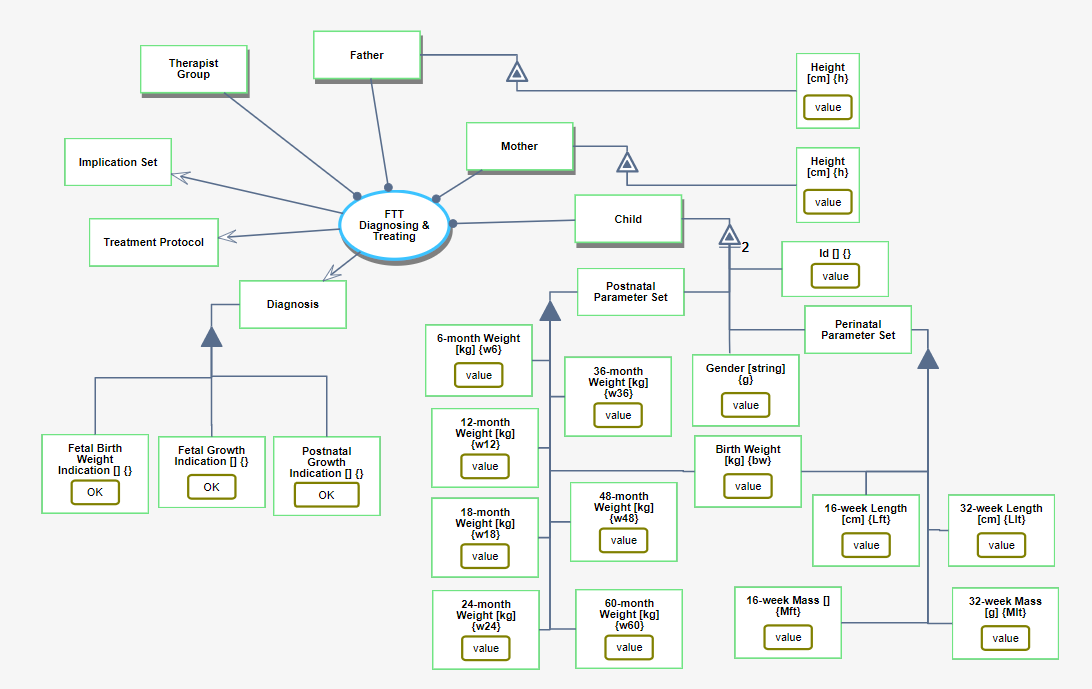


Figure 78: SD of the FTTell system, showing the main process which is called **Failure To Thrive (FTT) Diagnosing & Treating**, the involved object set that serve as agents, and resulting objects. One of the three outputs of this process is the object **Diagnosis** (at the center left), which consist of three parts, each for a different developmental period.

The **Failure To Thrive (FTT) Diagnosing & Treating** process involves the objects **Child**, **Mother**, **Father**,and **Therapist Group.** Each one of the is a physical object, connected to the main process by an agent link—a line with a black lollipop on its process end. OPM agents are humans involved in the process. The **Failure To Thrive (FTT) Diagnosing & Treating** process outputs three results: **Diagnosis**, **Treatment Protocol**,and **Implication Set**. Each result is represented by an object connected to the main process by a result link, as shown in Figure 78.

An OPM attribute is an object that describes the exhibitor—the exhibiting object. The height of the child’s parents can affect the FTT potential decision, so the **Mother** and **Father** have each a **Height** attribute. As Figure 78 shows, **Height** is a computational object with **cm** units and the alias **h**. The mother’s and father’s **Height** attributeobjects are respectively connected to the **Mother** and **Father** by an exhibition-characterization link, serving as input values.

The **Child** has three attributes: **Perinatal Parameter Set**, **Postnatal Parameter Set**, and an **Id**. The object **Perinatal Parameter Set** consists of the following five computational attributes: **Mass** [g] and **Length** [cm] at **16-weeks** and **32-weeks** pregnancy, as well as **Birth Weight** [kg]. The object **Postnatal Parameter Set** consists of the following seven computational attributes: **Weight** at 6-month, 12-month, 18-month, 24-month, 36-month, 48-monthand 60-month. All values that are part of the **Perinatal Parameter Set**, the **Postnatal Parameter Set** and the **Id** should be filled by the pediatrician.

The objects **Diagnosis**, **Treatment Protocol**, and **Implication Set** are connected to the process **FTT Diagnosing & Treating** with result links, which means that they are the output – the result of the process. The **Diagnosis** object consists of three parts: **Fetal Birth Weight Indication**, **Fetal Growth Indication**, and **Postnatal Growth Indication**. This is expressed in Figure 78 by aggregation-participation links—the lines with the black triangle in the middle. For the **Fetal Birth Weight Indication** object there are two possible result values: 'OK' and 'low'. For the **Fetal Growth Indication** object there are four possible values: 'OK', 'mother-dependent', 'mother- and child-dependent' and 'child dependent'. A result with value 'ok' means that there were no growth issues during the pregnancy. However, if the result is one of the other three options, then there were some problems, and an indication of possible reasons is then provided. The value of **Postnatal Growth Indication** is a text that indicates one of two options: (1) 'OK', which means that there is no FTT, or (2) an indication of FTT with the number of crossed percentiles and a score of the child’s FTT severity level.

We refine the main process by zooming into it, exposing three subprocesses (Figure 79 left): **Diagnosing**, **Treatment Defining**, and **Implication Defining**. As in this work we focus on the **Diagnosing** process, we move on to refining **Diagnosing** by zooming into it (Figure 79 right) and exposing three new, lower-level subprocesses: **Perinatal Growth Examining**, **Postnatal Growth Examining**, and **FTT Determining**. For diagnosing FTT, both perinatal and postnatal data should be considered.

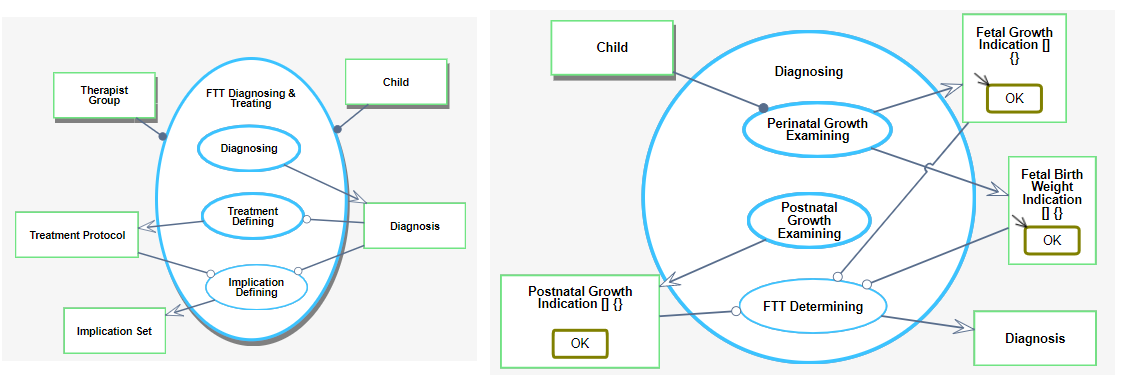


Figure 79: Main process **FTT Diagnosing & Treating** in-zoomed(left) and its **Diagnosing** subprocess in-zoomed further (right)

We start with **Perinatal Growth Examining**, whichuses data from the **Child** and provides two results: **Fetal Growth Indication** and **Fetal Birth Weight Indication**. These objects have the default value 'OK', indicating that by default the growth and the birth weight of the fetus were normal.

In Figure 80, we zoom into the **Perinatal Growth Examining** process to determine the FTT potential. To this end, we calculate the ponderal index (PI) at 16 and 32 weeks, using the corresponding fetal **Mass** and **Length** at these time points.

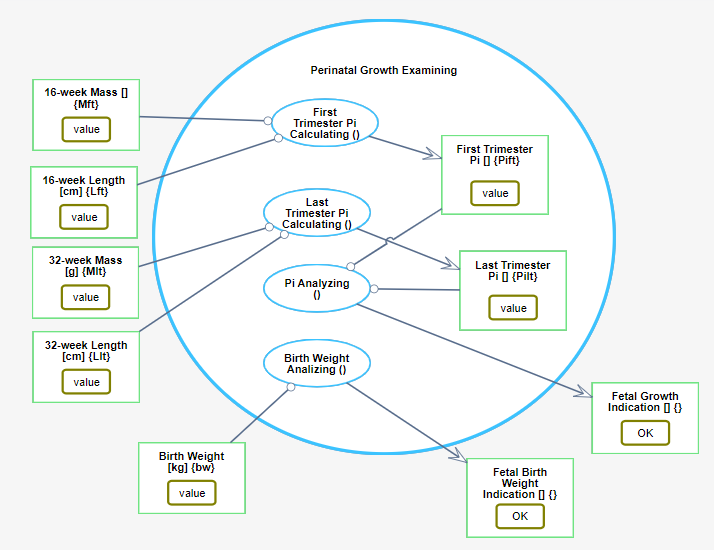


Figure 80: SD1.1.1 - **Perinatal Growth Examining** in-zoomed

To perform these calculations, we wrote software code in OPCloud using the Typescript programming language.

The mathematical formula of PI is:

The first subprocess, **First Trimester PI Calculating ()**, gets as input two values: **16-week Mass** {Mft} and **16-week Length** {Lft}. The result is written into the output object **First Trimester PI** {PIft}. The Typescript code is:

return (Mft/Math.pow(Lft, 3))\*100;

**Last Trimester PI Calculating ()** has a similar structure, using as inputs **32-week Mass** {Mlt}and **32-week Length** {Llt}, so the code for calculating **Last Trimester PI** {PIlt} is:

return (Mlt/Math.pow(Llt, 3))\*100;

To assess the FTT potential, **PI Analyzing ()** gets as input **PIft** and **PIlt**, and the result can be one of the following: (1) 'OK', which means no FTT, (2) 'mother-dependent', which can result from causes such as the mother’s stress or bad nutrition during pregnancy, (3) 'mother- and child-dependent', i.e., FTT is caused by both the mother and some problem with fetal development, and (4) 'child-dependent', implying a fetal development problem, such as lack of proteins. The program code of this process is:

let ft = true, lt = true;

let result = 'ok';

if (PIft<2.32 || PIft>2.85) {

ft = false;

}

if (PIlt<2.32 || PIlt>2.85) {

lt = false;

}

if (!ft) {

result = 'mother-dependent';

if (!lt) {

result = 'mother- and child-dependent';

}

} else if (!lt) {

result = 'child-dependent';

}

return result;

Finally, the **Birth Weight Analyzing ()** process uses the input **Birth Weight** {bw}, and the output is 'low' if the birth weight is less than 2.5 kg:

if (bw < 2.500) {

return 'low';

} else {

return 'OK';

}

Going back to Figure 79, we now zoom into the **Postnatal Growth Examining** process, where we calculate child’s percentile for each one of the eight periods (birth, 6 months, 12, 18, 24, 36, 48, and 60 months) based on actual data according to the standards set by the WHO[79] and the CDC[80], as shown in Figure 81.

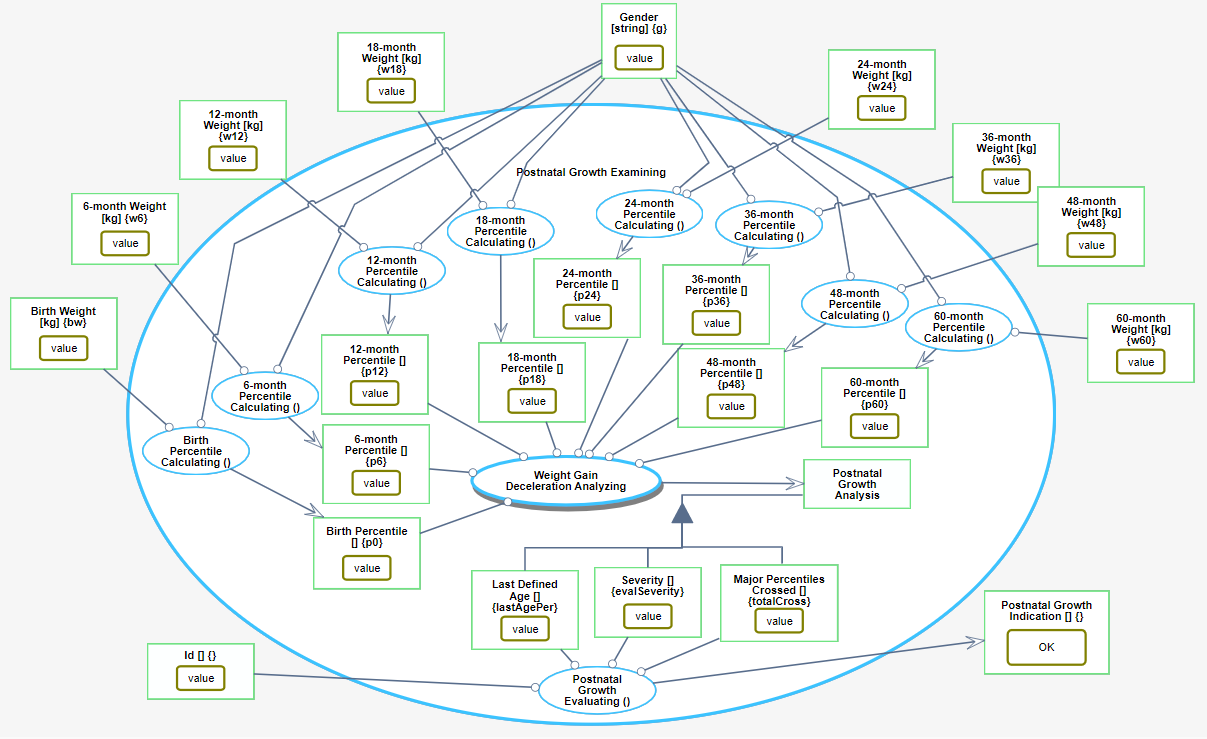


Figure 81: **Postnatal Growth Examining** in-zoomed

The percentage calculating for each period is done by translating the tables in the standards to TypeScript code, an example of which for calculating the percentile of a birth-weight (bw) follows.

if(g==='boy') {

if(bw<2.6) return '0-0';

if (bw>=2.6 && bw<2.8) return '1-5';

if (bw>=2.8 && bw<3) return '2-10';

if (bw>=3 && bw<3.3) return '3-25';

if (bw>=3.3 && bw<3.7) return '4-50';

if (bw>=3.7 && bw<4) return '5-75';

if (bw>=4 && bw<4.2) return '6-90';

if (bw>=4.2) return '7-95';

}

if(g==='girl') {

if(bw<2.5) return '0-0';

if (bw>=2.5 && bw<2.7) return '1-5';

if (bw>=2.7 && bw<2.9) return '2-10';

if (bw>=2.9 && bw<3.2) return '3-25';

if (bw>=3.2 && bw<3.6) return '4-50';

if (bw>=3.6 && bw<3.9) return '5-75';

if (bw>=3.9 && bw<4) return '6-90';

if (bw>=4) return '7-95';

}

return '9-99';

The pairs of numbers, such as '2-10' in line 4 of the code, express the percentile’s ordinal value and percent number.

Example in Figure 82: a 6-month old boy with a weight of 7 kg is in the second percentile where the 10% of the lower weight boys at that age are in, so in the TypeScript code below it is written as '2-10'. A 60-month (five years) old boy whose weight is 15 kg is in the first percentile where the 5% of the lower weight boys at that age are written in the code as '1-5'.

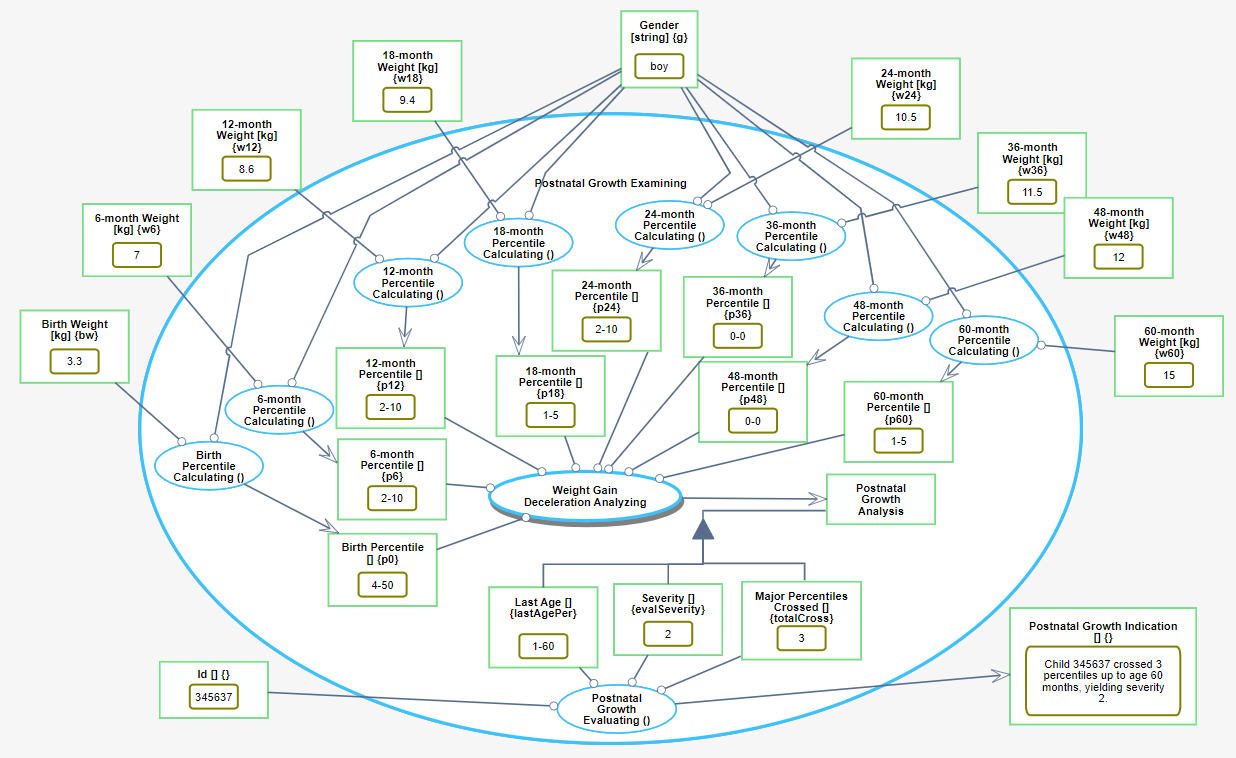


Figure 82: **Postnatal Growth Examining** in-zoomed – with real values

The input for calculating the percentile at each age is the corresponding child’s weight at that age. The process that follows the eight percentile calculating processes is **Weight Gain Deceleration Analyzing**, whose inputs are the values of the eight percentiles and whose output is the compound object **Postnatal Growth Analysis**.

Figure 83 presents the final output, calculated by the subprocess **Weight Gain Deceleration Analyzing**. This is the object **Postnatal Growth Analysis,** whichconsists of computational unitless parts: **Last Defined Age**, **Severity** and **Major Percentiles Crossed.** The value of the object **Major Percentiles Crossed** is the total number of percentiles the child crossed from birth until the age for which data is available. The value of the second object, **Severity**, represents the severity of the FTT, which is affected by **Major Percentiles Crossed** and bythe age at which the crossing occurred. The value of the third object, **Last Age**, is the most recent age in which the child was measured. The last process in the OPD in Figure 81, **Postnatal Growth Evaluating (),** combines the three results into a single sentence, which is the value of the result object. This is the only outcome of the model that the pediatrician should examine in the postnatal period.

It is phrased as a sentence “**Child XXX crossed p percentiles up to age YYY, yielding severity s.**”

There is one case, when a child crossed the third percentile, in which the sentence is different: "**Attention!!! Child XXX is under third percentile at age YYY months**". The reason is that in this case, the child is prone to develop FTT even if she or he did not cross two percentiles.

In the next step we zoom into the process **Weight Gain Deceleration Analyzing ().**

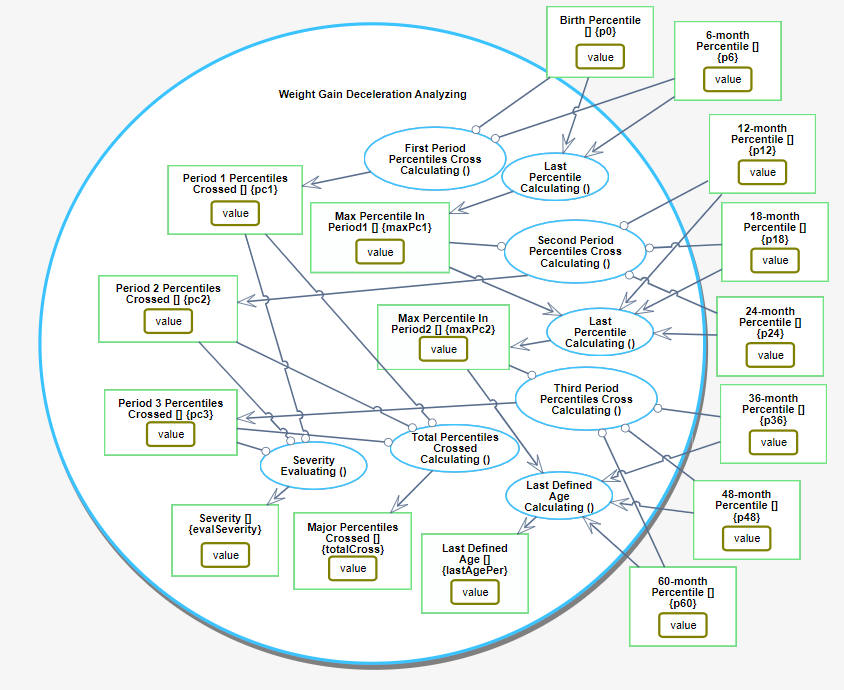


Figure 83: **Weight Gain Deceleration Analyzing** process in-zoomed

The major percentiles are defined as 5, 10, 25, 50, 75, 90, 95. Since the probability of reducing two percentiles changes over time, we divide the age of the child into three periods: 0-6, 7-24, and 25-60 months (see Table 3).

FTT is suspected when a child crosses two or more major percentiles downwards. The probability that a child will cross two percentiles at different periods decreases with age: The severity of such crossing at an early period is less than that in an older age period. The severity score is in the opposite direction to the crossing probability. Thus, from birth to six months, the probability is 39%, so the severity score for this period is 0.61, while at age 25-60 months the probability is much closer to 0, causing the severity score to be close to 1 (see Table 3).

Table 3: Probability of reducing two percentiles (PRTP) and calculated severity score for the three periods

|  |  |  |
| --- | --- | --- |
| Period (age in months) | Probability of Reducing Two Percentiles | Severity Score (range: 0 to 1) |
| 0-6 | 0.39 | 0.61 |
| 7-24 | 0.06-0.15 | 0.895 |
| 25-60 | 0.01-0.05 | 0.97 |

The process described in Figure 83 has several subprocesses. Three of them calculate the percentiles crossed during each period - **First Period Percentiles Cross Calculating ()**, **Second Period Percentiles Cross Calculating ()**, and **Third Period Percentiles Cross Calculating ()**. The output of each such process is assigned to a corresponding object: **Period 1 Percentiles Crossed**, **Period 2 Percentiles Crossed**, and **Period 3 Percentiles Crossed**. The code inserted to each of these three processes is show in Function 2, Function 3, and Function 4. The next three processes calculate the last age in which the child was measured during that period. The output of those processes is assigned to a corresponding object: **Max Percentile In Period1**, **Max Percentile In Period2, Last Defined Age**. The seventh process, **Total Percentiles Crossed Calculating (),** summarizes the first three values and assigns the result to the object **Major Percentiles Crossed**. The last process, **Severity Evaluating (),** calculates the severity of the percentiles crossed using a weighted sum. The weight is taken from the column *Severity Score*in Table 3. The result is stored in an object named **Severity.** For example, a child who crossed two major percentiles in the first period, one major percentile in the second period, and one in the third period, will have a **Major Percentiles Crossed** value of , with **Severity** of . A child who crossed one major percentile in the first period, one major percentile in the second period, and two major percentiles in the third period will have a **Major Percentiles Crossed** value of 4 as well, but in this case, the **Severity** will be higher: . The reason is that it is more common for a baby under six months old to cross down two percentiles, but the situation is more severe for a child older than two years.

|  |
| --- |
| Function 2. First Period Percentiles Cross Calculating (). Pi means percentiles at I months |
| let level0 =+p0.substr(0,1);  let level6 =+p6.substr(0,1);  let crossPeriod1 = 0;  if ((level0!==9) && (level6!==9)) {  crossPeriod1 = level0 – level6;  } ­­  return crossPeriod1; |

|  |
| --- |
| Function 3. Second Period Percentiles Cross Calculating (). Pi means percentiles at I months |
| let prevLevel = +maxPc1.substr(0,1);  let level12=+p12.substr(0,1);  let level18=+p18.substr(0,1);  let level24=+p24.substr(0,1);  let crossPeriod2 = 0;  if (prevLevel!==9) {  if(level24!==9) {  crossPeriod2 = prevLevel – level24;  } else if (level18!==9) {  crossPeriod2 = prevLevel – level18;  } else if (level12!==9) {  crossPeriod2 = prevLevel – level12;  }  } else if (level12!==9) {  if(level24!==9) {  crossPeriod2 = level12 – level24;  } else if (level18!==9) {  crossPeriod2 = level12 – level18;  }  } else if (level18!==9) {  if(level24!==9) {  crossPeriod2 = level18 – level24;  }  }  return crossPeriod2; |

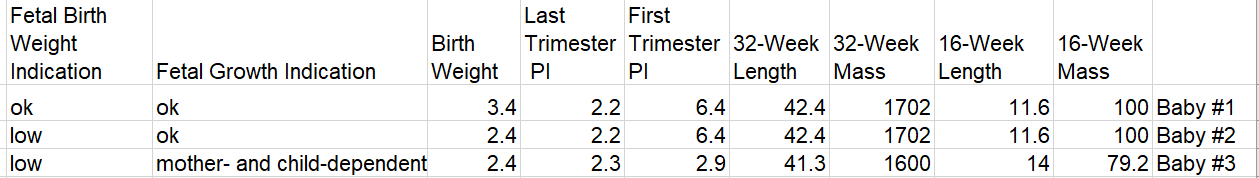
|  |
| --- |
| Function 4. Third Period Percentiles Cross Calculating (). Pi means percentiles at I months |
| let prevLevel=+maxPc2.substr(0,1);  let level36=+p36.substr(0,1);  let level48=+p48.substr(0,1);  let level60=+p60.substr(0,1);  let crossPeriod3 = 0;  if (prevLevel!==9) {  if(level60!==9) {  crossPeriod3 = prevLevel – level60;  } else if (level48!==9) {  crossPeriod3 = prevLevel – level48;  } else if (level36!==9) {  crossPeriod3 = prevLevel – level36;  }  } else if (level36!==9) {  if (level60!==9) {  crossPeriod3 = level36 – level60;  } else if (level48!==9) {  crossPeriod3 = level36 – level48;  }  } else if (level48!==9) {  if (level60!==9) {  crossPeriod3 = level48 – level60;  }  }  return crossPeriod3; |

## **FTTell Perinatal Stage Evaluation and Results**

Having developed and validated the perinatal stage in FTTell model, we inserted into it the required data of each specific child, one at a time. The data is simulated and was generated to test the efficacy of the FTTell system. Clicking on the execution button, we can follow tokens running visually through the different OPDs in a depth-first manner. Finally, the result values are updated in the model and can be exported to an Excel file.

Table 4 presents the results of particular data sets of three babies after executing the model. These are saved in an Excel file for documentation and can further serve for statistical analysis.

Table 4: Model execution results as saved in an Excel file



The first baby, Baby #1, was born with a weigh of 3.4 kg. It had a weight of 100 g and a length of 11.6 cm as a 16-weeks fetus, and a weight of 1702 g and a length of 42.4 cm as a 32-weeks fetus. This baby is defined as normal, with no FTT suspected. Baby #2 has the same pregnancy values but a low birth weight of 2.4 kg, so it is classified by the system as 'ok' in the **Fetal Growth Indication** parameter, but as 'low' in the **Fetal Birth Weight Indication** parameter. Therefore, it may have some FTT suspicion and should be under follow-up. However, there is a chance that this baby’s weight will catch up with the normal one in the next growth interval. Baby #3 has low length and mass values during the different perinatal stages, and is classified by the system with **Fetal Growth Indication** value ‘mother- and child-dependent’. This result is determined by calculating PI values and analyzing them. Bad PI values at the 16-week pregnancy time point are usually due to mother-related reasons, such as bad nutrition and stress. Bad PI values at the 32-week pregnancy time point are caused usually by child-related reasons, which can be genetic causes, malnutrition, or lack of proteins. In addition, the third baby has a low birth weight. For all these reasons, it has high chances to suffer from FTT and has to be checked by an expert.

## **FTTell Postnatal Stage Evaluation and Results**

To evaluate the validity of the diagnosis produced by FTTell, the system we developed, we compared the system's diagnosis results on a sample of 100 children to those of pediatricians who are experts in FTT. To this end, we requested and got permission number 0665-19-RMB from the Institutional Helsinki committee. Some of the children in this sample do not have FTT at all, some of them have light FTT and the others—severe FTT. Table 5 contains the weight data of all the 100 children, distributed over the three periods.

Table 5: data for 100 childern used for evaluating FTTell. A blank cell indicates a missing value, as the child was not weighed at the age of that cell

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Gender | Birth Weight | 6-month Weight | 12-month Weight | 18-month Weight | 24-month Weight | 36-month Weight | 48-month Weight | 60-month Weight |
| Child #1 | girl | 2.8 |  | 7.5 | 8.5 | 9.6 | 11 |  |  |
| Child #2 | boy | 4.36 | 6.42 |  |  | 11 |  |  | 17 |
| Child #3 | girl | 2.74 | 5.83 | 7.6 | 9 |  | 12 |  |  |
| Child #4 | boy | 3.18 | 6.86 | 9 | 9.27 | 10.7 | 12 | 13.2 | 15 |
| Child #5 | girl | 2.58 | 5.4 | 7.3 | 8.24 | 9.5 | 10.7 |  | 14.5 |
| Child #6 | girl | 3.9 | 7.8 | 8.3 | 9.8 |  |  | 15 |  |
| Child #7 | girl | 3.42 | 5.6 | 7.1 |  | 9.5 |  | 12 |  |
| Child #8 | boy | 3.76 | 6.9 |  | 10.4 |  |  | 15 | 16.5 |
| Child #9 | girl | 2.65 |  |  | 7.94 |  | 11 |  | 12.7 |
| Child #10 | boy | 2.8 | 7 | 7.85 | 8.55 |  | 10.2 |  | 13.2 |
| Child #11 | boy | 3.425 | 5.8 | 8 | 10.5 |  |  | 15.5 | 18.5 |
| Child #12 | boy | 3.55 | 7.15 | 8.5 |  | 10.6 | 12.4 | 14 | 15.4 |
| Child #13 | boy | 3.15 | 6.3 | 7.4 | 8 | 8.75 | 10.54 |  | 13.8 |
| Child #14 | boy | 3.25 | 6.19 | 7.85 | 8.8 | 10 | 11.5 | 12.2 | 13.8 |
| Child #15 | girl | 2.9 |  | 7 |  | 9.9 |  |  | 12.5 |
| Child #16 | girl | 2.85 | 6.1 | 7 | 8.36 | 9 |  | 12.2 | 13.6 |
| Child #17 | boy | 2.95 | 7.18 | 7.8 | 9.2 | 11 | 12 | 12.5 | 16.3 |
| Child #18 | girl | 3.4 | 7.18 | 8.7 | 10.6 |  |  |  | 17 |
| Child #19 | girl | 3.35 |  | 8.2 | 8.67 | 9.5 | 10.4 |  |  |
| Child #20 | girl | 3.7 | 7 |  | 9.79 | 12 | 14 |  | 17.3 |
| Child #21 | girl | 3 |  | 6.58 | 7.6 | 8.5 | 11 | 11.9 |  |
| Child #22 | boy | 3.25 | 7 | 8.4 | 11.5 |  | 12 |  |  |
| Child #23 | boy | 3.065 | 6.4 | 7.8 | 9.2 | 10.3 |  |  | 15.9 |
| Child #24 | boy | 2.9 | 7.42 | 8.9 | 9.89 | 11 | 13 | 14.5 | 17.1 |
| Child #25 | boy | 2.81 | 7.3 | 8.75 |  | 11.2 | 14 | 15.6 |  |
| Child #26 | girl | 2.8 | 5.15 | 6.6 | 8.1 |  | 11.2 | 12.7 | 15.1 |
| Child #27 | girl | 3.5 | 6.6 | 8 | 9.3 | 11 | 12 | 14 | 15.5 |
| Child #28 | girl | 3.5 | 6.6 | 7.8 |  | 9.6 |  | 14 | 15.8 |
| Child #29 | girl | 2.9 | 6.35 | 8.1 | 9 | 10 |  | 14.8 | 15.7 |
| Child #30 | boy | 3.85 | 6.9 | 8.1 |  |  | 12.6 | 15 |  |
| Child #31 | girl | 2.85 |  | 8.1 | 9.3 | 10.5 | 11 | 12.4 |  |
| Child #32 | boy | 3.5 | 6.9 | 8.38 | 9.2 |  | 12 | 12.5 | 14 |
| Child #33 | girl | 2.8 | 5.7 | 7.2 | 8.4 | 10 |  |  |  |
| Child #34 | boy | 3.12 | 6.3 | 7.89 |  | 11 | 11.7 | 13.8 | 15.2 |
| Child #35 | girl | 3.08 | 6.4 | 8.5 |  |  |  |  | 19 |
| Child #36 | girl | 2.91 |  | 7.7 |  |  | 12 |  |  |
| Child #37 | boy | 3.2 |  | 8.3 | 9.3 | 10.7 | 12 | 14.1 | 15.5 |
| Child #38 | boy | 3.3 | 6.5 | 8.35 | 9.2 |  |  |  | 19.5 |
| Child #39 | boy | 2.6 | 6.5 | 8.5 |  | 10 | 13 | 14.5 | 20 |
| Child #40 | girl | 3.4 | 7.42 | 8.2 |  | 11.3 | 15 |  |  |
| Child #41 | boy | 3.05 |  | 8 | 9 |  | 11.9 | 12.8 |  |
| Child #42 | girl | 2.5 | 6.72 | 8.15 |  |  |  |  | 14.4 |
| Child #43 | girl | 3.2 |  | 8.45 |  | 10 |  | 14.5 |  |
| Child #44 | girl | 2.65 | 5.9 | 7.5 |  |  | 10.4 |  |  |
| Child #45 | girl | 3.7 | 6 |  |  |  | 11.4 |  |  |
| Child #46 | boy | 2.8 | 6.3 | 8.2 | 9.4 | 10.2 | 12 |  |  |
| Child #47 | boy | 3.6 | 7.7 | 8.9 |  |  | 13 | 14.1 |  |
| Child #48 | girl | 3.5 | 6.3 | 8 |  | 11.2 |  |  | 17.5 |
| Child #49 | girl | 2.6 | 5.9 | 7 |  | 9.5 | 10 |  |  |
| Child #50 | girl | 2.6 |  |  | 6.9 | 9 | 10.3 |  |  |
| Child #51 | boy | 3 | 6.1 | 7.2 |  | 9.3 | 10 | 12.5 |  |
| Child #52 | girl | 3 |  |  |  |  | 11 | 12.7 |  |
| Child #53 | boy | 2.95 | 6.4 | 7.9 | 10 | 10.7 | 11.8 | 13.5 | 15.5 |
| Child #54 | boy | 3 | 6.7 | 8.6 |  | 10.7 | 13.4 | 16 |  |
| Child #55 | girl | 3.16 |  | 7.8 | 9.5 | 12.5 | 14.5 |  |  |
| Child #56 | boy | 2.74 | 6.3 | 6.5 | 7.9 | 8.8 | 10 |  |  |
| Child #57 | boy | 2.71 |  | 8.5 | 10 | 11.2 | 12 | 12.9 | 14 |
| Child #58 | boy | 2.8 | 5.9 | 7.3 | 8.4 | 9.2 | 11 | 11.5 | 12.8 |
| Child #59 | boy | 3.5 | 6.7 | 8.3 | 9.15 | 10 | 11.2 | 12.6 |  |
| Child #60 | girl | 3.4 | 6.3 | 7.8 |  | 11 |  |  |  |
| Child #61 | boy | 2.9 | 5.72 | 7.62 | 8.7 | 9.4 | 10.7 | 13.7 |  |
| Child #62 | girl | 3.27 |  | 8 | 9.2 | 10.1 | 11 | 12.8 | 14.3 |
| Child #63 | boy | 3.36 | 7.3 | 8 | 8.5 | 9.6 | 11.6 |  |  |
| Child #64 | girl | 3.6 | 6.25 | 7.18 |  | 9 |  |  |  |
| Child #65 | girl | 2.95 |  | 7.32 |  | 10 |  |  |  |
| Child #66 | girl | 3.175 | 6.8 |  |  | 10 | 13 |  |  |
| Child #67 | boy | 3.14 | 7 | 8.4 | 9.5 |  |  | 15 |  |
| Child #68 | boy | 3 | 7.4 | 7.5 |  | 10.5 | 12.4 | 13.8 | 14.8 |
| Child #69 | boy | 3.6 |  | 9.2 | 9.3 |  |  | 14 |  |
| Child #70 | girl | 3.16 | 6.6 | 7.8 | 10 |  |  | 14 | 15.8 |
| Child #71 | boy | 3 | 7 | 8.15 | 9.2 | 10.5 |  |  |  |
| Child #72 | boy | 2.8 | 7.6 | 8.9 |  |  | 12.3 | 13.6 |  |
| Child #73 | boy | 2.87 | 6.8 | 8.6 | 10 |  | 12.5 | 13.5 | 14.5 |
| Child #74 | boy | 3.4 | 6.5 | 8 | 9.3 |  | 12 | 13.5 | 15.5 |
| Child #75 | girl | 2.85 | 6.5 | 8 | 8.7 | 10.3 | 11.4 | 12.8 | 15 |
| Child #76 | girl | 2.714 | 5.2 | 6.7 | 7.9 | 8.5 | 10.5 |  |  |
| Child #77 | boy | 3.3 | 7 | 8.6 | 9.4 | 10.5 | 11.5 | 12 | 15 |
| Child #78 | girl | 3.075 | 7 | 8.5 | 9.9 | 10.7 |  | 16 |  |
| Child #79 | boy | 3.26 |  | 7.6 | 8.3 | 10 | 12 | 13.8 |  |
| Child #80 | girl | 3.038 | 6.3 |  | 9 | 9.7 | 12 | 13 | 14.4 |
| Child #81 | boy | 3 | 6.5 | 7.7 | 9.25 |  | 12.5 | 14.3 |  |
| Child #82 | girl | 2.73 | 5.8 | 7.2 | 10 | 10 | 11.4 | 13.2 | 15 |
| Child #83 | boy | 3.9 | 8.5 | 10.28 | 11 | 11.6 | 13.5 | 15.8 | 17 |
| Child #84 | girl | 2.65 | 5.5 | 7.2 | 8.3 | 9.5 | 11.4 | 12 |  |
| Child #85 | girl | 3.18 | 6 | 7.95 | 9 | 9.7 | 11.2 |  | 14.3 |
| Child #86 | girl | 2.7 | 6.1 | 7.5 | 9 | 10.5 | 13 | 15 |  |
| Child #87 | boy | 2.67 | 7 | 8.35 | 9.3 | 10.4 |  | 13.8 |  |
| Child #88 | boy | 2.77 | 5.48 | 6.8 | 8 | 9.2 | 11 |  | 13.1 |
| Child #89 | boy | 3 | 5.8 | 7.6 | 8.6 | 10.4 | 11.6 | 13.2 | 14.7 |
| Child #90 | boy | 2.64 | 6.2 | 7.25 | 8.95 | 9.6 | 10.3 | 11 | 12.9 |
| Child #91 | boy | 2.52 | 5.6 | 7.18 |  |  | 11.4 | 13 | 14.5 |
| Child #92 | boy | 2.89 | 6.5 | 7.4 | 8.35 | 9.9 | 11 | 12 | 14.2 |
| Child #93 | girl | 3.01 | 7.5 | 9 | 10.6 | 12.7 | 15.4 | 16.5 |  |
| Child #94 | boy | 3.66 | 8.8 | 10.5 |  | 12 | 14 | 15.8 | 17.4 |
| Child #95 | boy | 3.55 | 7.2 | 9 |  | 12 | 15.8 | 16.5 |  |
| Child #96 | girl | 3.15 | 7.4 | 9 | 10.9 |  | 14 |  | 18 |
| Child #97 | boy | 3.3 | 7 | 8.3 | 8.8 | 9.2 | 10.8 | 12 | 13 |
| Child #98 | girl | 3.2 | 6.6 | 7.9 | 9.3 | 10.7 | 12.6 | 14.5 | 16.7 |
| Child #99 | boy | 3.6 | 6.5 | 7.7 | 9.2 | 10 | 11.9 | 13 | 14.2 |
| Child #100 | boy | 2.66 | 6.2 | 7.1 | 8 | 9.5 |  | 13.4 | 14.5 |

Executing in batch mode the OPM model at the heart of FTTell with the 100 children’s data has produced an Excel file with a table containing the summarized results. Table 6 shows the intermediate results, which are part of the excel file – the percentiles in each age.

Table 6: Calculated percentiles by FTTell for each age of each child

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Birth Percentile | 6-month Percentile | 12-month Percentile | 18-month Percentile | 24-month Percentile | 36-month Percentile | 48-month Percentile | 60-month Percentile |
| Child #1 | 2-10 | ------- | 1-5 | 1-5 | 1-5 | 0-0 | ------ | ------ |
| Child #2 | 7-95 | 0-0 | ------ | ----- | 2-10 | ----- | ------ | 3-25 |
| Child #3 | 2-10 | 0-0 | 1-5 | 2-10 | ------ | 2-10 | ------ | ------ |
| Child #4 | 3-25 | 1-5 | 3-25 | 1-5 | 2-10 | 1-5 | 0-0 | 1-5 |
| Child #5 | 1-5 | 0-0 | 1-5 | 0-0 | 1-5 | 0-0 | ------ | 1-5 |
| Child #6 | 6-90 | 4-50 | 3-25 | 3-25 | ------ | ------ | 3-25 | ------ |
| Child #7 | 4-50 | 0-0 | 0-0 | ------ | 1-5 | ------ | 0-0 | ------ |
| Child #8 | 5-75 | 2-10 | ------ | 3-25 | ------ | ------ | 3-25 | 2-10 |
| Child #9 | 1-5 | ------ | ------ | 0-0 | ------ | 0-0 | ------ | 0-0 |
| Child #10 | 2-10 | 2-10 | 0-0 | 0-0 | ------ | 0-0 | ------ | 0-0 |
| Child #11 | 4-50 | 0-0 | 0-0 | 3-25 | ------ | ------ | 3-25 | 4-50 |
| Child #12 | 4-50 | 2-10 | 2-10 | ------ | 2-10 | 2-10 | 2-10 | 1-5 |
| Child #13 | 3-25 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | ------ | 0-0 |
| Child #14 | 3-25 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 |
| Child #15 | 3-25 | ------ | 0-0 | ------ | 2-10 | ------ | ------ | 0-0 |
| Child #16 | 2-10 | 1-5 | 0-0 | 0-0 | 0-0 | ------ | 0-0 | 0-0 |
| Child #17 | 2-10 | 2-10 | 0-0 | 1-5 | 2-10 | 1-5 | 0-0 | 2-10 |
| Child #18 | 4-50 | 3-25 | 3-25 | 4-50 | ------ | ------ | ------ | 3-25 |
| Child #19 | 4-50 | ------ | 3-25 | 1-5 | 1-5 | 0-0 | ------ | ------ |
| Child #20 | 5-75 | 3-25 | ------ | 3-25 | 4-50 | 4-50 | ------ | 3-25 |
| Child #21 | 3-25 | ------ | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | ------ |
| Child #22 | 3-25 | 2-10 | 2-10 | 4-50 | ------ | 1-5 | ------ | ------ |
| Child #23 | 3-25 | 0-0 | 0-0 | 1-5 | 1-5 | ------ | ------ | 2-10 |
| Child #24 | 2-10 | 3-25 | 2-10 | 2-10 | 2-10 | 2-10 | 2-10 | 3-25 |
| Child #25 | 2-10 | 2-10 | 2-10 | ------ | 2-10 | 3-25 | 3-25 | ------ |
| Child #26 | 2-10 | 0-0 | 0-0 | 0-0 | ------ | 0-0 | 0-0 | 1-5 |
| Child #27 | 4-50 | 2-10 | 2-10 | 2-10 | 3-25 | 2-10 | 2-10 | 2-10 |
| Child #28 | 4-50 | 2-10 | 2-10 | ------ | 1-5 | ------ | 2-10 | 2-10 |
| Child #29 | 3-25 | 2-10 | 2-10 | 2-10 | 2-10 | ------ | 3-25 | 2-10 |
| Child #30 | 5-75 | 2-10 | ------ | ------ | ------ | 2-10 | 3-25 | ------ |
| Child #31 | 2-10 | ------ | ------ | 2-10 | 2-10 | 0-0 | 0-0 | ------ |
| Child #32 | 4-50 | 2-10 | 1-5 | 1-5 | ------ | 1-5 | 0-0 | 0-0 |
| Child #33 | 2-10 | 0-0 | 0-0 | 1-5 | 2-10 | ------ | ------ | ------ |
| Child #34 | 3-25 | 0-0 | 0-0 | ------ | 2-10 | 0-0 | 1-5 | 1-5 |
| Child #35 | 3-25 | 2-10 | 3-25 | ------ | ------ | ------ | ------ | 4-50 |
| Child #36 | 3-25 | ------ | 2-10 | ------ | ------ | 2-10 | ------ | ------ |
| Child #37 | 3-25 | ------ | 1-5 | 1-5 | 2-10 | 1-5 | 2-10 | 2-10 |
| Child #38 | 4-50 | 0-0 | ------ | 1-5 | ------ | ------ | ------ | 4-50 |
| Child #39 | 1-5 | 0-0 | 2-10 | ------ | 0-0 | 2-10 | 2-10 | 4-50 |
| Child #40 | 4-50 | 4-50 | 3-25 | ------ | 3-25 | 4-50 | ------ | ------ |
| Child #41 | 3-25 | ------ | 0-0 | 0-0 | ------ | 1-5 | 0-0 | ------ |
| Child #42 | 1-5 | 3-25 | 2-10 | ------ | ------ | ------ | ------ | 1-5 |
| Child #43 | 4-50 | ------ | 3-25 | ------ | 2-10 | ------ | 2-10 | ------ |
| Child #44 | 1-5 | 0-0 | 1-5 | ------ | ------ | 0-0 | ------ | ------ |
| Child #45 | 5-75 | 1-5 | ------ | ------ | ------ | 1-5 | ------ | ------ |
| Child #46 | 2-10 | 0-0 | 1-5 | 1-5 | 1-5 | 1-5 | ------ | ------ |
| Child #47 | 4-50 | 3-25 | 2-10 | ------ | ------ | 2-10 | 2-10 | ------ |
| Child #48 | 4-50 | 2-10 | 2-10 | ------ | 3-25 | ------ | ------ | 3-25 |
| Child #49 | 1-5 | 0-0 | 0-0 | ------ | 1-5 | 0-0 | ------ | ------ |
| Child #50 | 1-5 | ------ | ------ | 0-0 | 0-0 | 0-0 | ------ | ------ |
| Child #51 | 3-25 | 0-0 | 0-0 | ------ | 0-0 | 0-0 | 0-0 | ------ |
| Child #52 | 3-25 | ------ | ------ | ------ | ------ | 0-0 | 0-0 | ------ |
| Child #53 | 2-10 | 0-0 | 0-0 | 2-10 | 2-10 | 1-5 | 1-5 | 2-10 |
| Child #54 | 3-25 | 1-5 | 2-10 | ------ | 2-10 | 3-25 | 3-25 | ------ |
| Child #55 | 3-25 | ------ | 2-10 | 3-25 | 5-75 | 4-50 | ------ | ------ |
| Child #56 | 1-5 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | ------ | ------ |
| Child #57 | 1-5 | ------ | 2-10 | 2-10 | 2-10 | 1-5 | 0-0 | 0-0 |
| Child #58 | 2-10 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 |
| Child #59 | 4-50 | 1-5 | 1-5 | 1-5 | 0-0 | 0-0 | 0-0 | ------ |
| Child #60 | 4-50 | 2-10 | 2-10 | ------ | 3-25 | ------ | ------ | ------ |
| Child #61 | 2-10 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | 1-5 | ------ |
| Child #62 | 4-50 | ------ | 2-10 | 2-10 | 2-10 | 0-0 | 0-0 | 0-0 |
| Child #63 | 4-50 | 2-10 | 0-0 | 0-0 | 0-0 | 0-0 | ------ | ------ |
| Child #64 | 5-75 | 2-10 | 0-0 | ------ | 0-0 | ------ | ------ | ------ |
| Child #65 | 3-25 | ------ | 1-5 | ------ | 2-10 | ------ | ------ | ------ |
| Child #66 | 3-25 | 3-25 | ------ | ------ | 2-10 | 3-25 | ------ | ------ |
| Child #67 | 3-25 | 2-10 | 2-10 | 2-10 | ------ | ------ | 3-25 | ------ |
| Child #68 | 3-25 | 3-25 | 0-0 | ------ | 2-10 | 2-10 | 1-5 | 1-5 |
| Child #69 | 4-50 | ------ | 3-25 | 1-5 | ------ | ------ | 2-10 | ------ |
| Child #70 | 3-25 | 2-10 | 2-10 | 3-25 | ------ | ------ | 2-10 | 2-10 |
| Child #71 | 3-25 | 2-10 | 1-5 | 1-5 | 2-10 | ------ | ------ | ------ |
| Child #72 | 2-10 | 3-25 | 2-10 | ------ | 2-10 | 2-10 | 1-5 | ------ |
| Child #73 | 2-10 | 1-5 | 2-10 | 2-10 | ------ | 2-10 | 1-5 | 0-0 |
| Child #74 | 4-50 | 0-0 | 0-0 | 1-5 | ------ | 1-5 | 1-5 | 2-10 |
| Child #75 | 2-10 | 2-10 | 2-10 | 1-5 | 2-10 | 1-5 | 0-0 | 1-5 |
| Child #76 | 2-10 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | ------ | ------ |
| Child #77 | 4-50 | 2-10 | 2-10 | 1-5 | 2-10 | 0-0 | 0-0 | 1-5 |
| Child #78 | 3-25 | 3-25 | 3-25 | 3-25 | 3-25 | ------ | 3-25 | ------ |
| Child #79 | 3-25 | ------ | 0-0 | 0-0 | 0-0 | 1-5 | 1-5 | ------ |
| Child #80 | 3-25 | 2-10 | ------ | 2-10 | 1-5 | 2-10 | 1-5 | 1-5 |
| Child #81 | 3-25 | 0-0 | 0-0 | 1-5 | ------ | 2-10 | 2-10 | ------ |
| Child #82 | 2-10 | 0-0 | 0-0 | 3-25 | 2-10 | 1-5 | 1-5 | 1-5 |
| Child #83 | 5-75 | 5-75 | 4-50 | 4-50 | 3-25 | 3-25 | 3-25 | 3-25 |
| Child #84 | 1-5 | 0-0 | 0-0 | 0-0 | 1-5 | 1-5 | 0-0 | ------ |
| Child #85 | 3-25 | 1-5 | 2-10 | 2-10 | 1-5 | 0-0 | ------ | 0-0 |
| Child #86 | 2-10 | 1-5 | 1-5 | 2-10 | 2-10 | 3-25 | 3-25 | ------ |
| Child #87 | 1-5 | 2-10 | 1-5 | 1-5 | 1-5 | ------ | 1-5 | ------ |
| Child #88 | 1-5 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | ------ | 0-0 |
| Child #89 | 3-25 | 0-0 | 0-0 | 0-0 | 1-5 | 0-0 | 0-0 | 1-5 |
| Child #90 | 1-5 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 |
| Child #91 | 0-0 | 0-0 | 0-0 | ------ | ------ | 0-0 | 0-0 | 0-0 |
| Child #92 | 2-10 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 |
| Child #93 | 3-25 | 4-50 | 4-50 | 4-50 | 5-75 | 5-75 | 4-50 | ------ |
| Child #94 | 4-50 | 5-75 | 5-75 | ------ | 3-25 | 3-25 | 3-25 | 3-25 |
| Child #95 | 4-50 | 2-10 | 3-25 | ------ | 3-25 | 5-75 | 4-50 | ------ |
| Child #96 | 3-25 | 4-50 | 4-50 | 4-50 | ------ | 4-50 | ------ | 3-25 |
| Child #97 | 4-50 | 2-10 | 1-5 | 0-0 | 0-0 | 0-0 | 0-0 | 0-0 |
| Child #98 | 4-50 | 2-10 | 2-10 | 2-10 | 3-25 | 2-10 | 2-10 | 3-25 |
| Child #99 | 4-50 | 0-0 | 0-0 | 1-5 | 0-0 | 1-5 | 0-0 | 0-0 |
| Child #100 | 1-5 | 0-0 | 0-0 | 0-0 | 0-0 | ------ | 1-5 | 0-0 |

For each child, a pediatric gastrologist resident wrote her diagnosis based on the child’s data without being exposed to the analysis produced by FTTell. The expert gave a score of 1 for no FTT or a light FTT, 2 for a medium FTT, and 3 for a severe FTT. We then compared these scores to the FTTell result. Cases for which there was a difference between the two scores were carefully reassessed by a highly experienced pediatric gastrologist expert. The result demonstrated in Table 7.

Table 8: Percentiles crossed in three age periods, Total Percentiles Cross, Evaluated severity and Last Measured Age according to FTTell and Domain Expert Severity Evaluation to each child

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Period 1 Percentiles Crossed {pc1}  (0-6 months) | Period 2 Percentiles Crossed {pc2}  (6-24 months) | Period 3 Percentiles Crossed {pc3}  (24-60 months) | Total Percentiles Crossed  (pc1+pc2+pc3) | Evaluated Severity  (0.61\*pc1+ 0.895\*pc2+ 0.97\*pc3) | Last Month of Weighing | Domain Expert Severity Evaluation |
| Child #1 | 0 | 1 | 1 | 2 | 2 | 60 | 3 |
| Child #2 | 7 | -2 | -1 | 4 | 2 | 60 | 2 |
| Child #3 | 2 | -2 | 0 | 0 | -1 | 36 | 1 |
| Child #4 | 2 | -1 | 1 | 2 | 1 | 60 | 3 |
| Child #5 | 1 | -1 | 0 | 0 | 0 | 60 | 2 |
| Child #6 | 2 | 1 | 0 | 3 | 2 | 48 | 1 |
| Child #7 | 4 | -1 | 1 | 4 | 3 | 48 | 3 |
| Child #8 | 3 | -1 | 1 | 3 | 2 | 60 | 1 |
| Child #9 | 0 | 1 | 0 | 1 | 1 | 60 | 2 |
| Child #10 | 0 | 2 | 0 | 2 | 2 | 60 | 3 |
| Child #11 | 4 | -3 | -1 | 0 | -1 | 60 | 3 |
| Child #12 | 2 | 0 | 1 | 3 | 2 | 60 | 3 |
| Child #13 | 3 | 0 | 0 | 3 | 2 | 60 | 3 |
| Child #14 | 3 | 0 | 0 | 3 | 2 | 60 | 3 |
| Child #15 | 0 | 1 | 2 | 3 | 3 | 60 | 2 |
| Child #16 | 1 | 1 | 0 | 2 | 2 | 60 | 3 |
| Child #17 | 0 | 0 | 0 | 0 | 0 | 60 | 3 |
| Child #18 | 1 | -1 | 1 | 1 | 1 | 60 | 1 |
| Child #19 | 0 | 3 | 1 | 4 | 4 | 36 | 2 |
| Child #20 | 2 | -1 | 1 | 2 | 1 | 60 | 1 |
| Child #21 | 0 | 3 | 0 | 3 | 3 | 48 | 3 |
| Child #22 | 1 | -2 | 3 | 2 | 2 | 36 | 2 |
| Child #23 | 3 | -1 | -1 | 1 | 0 | 60 | 2 |
| Child #24 | -1 | 1 | -1 | 0 | 0 | 60 | 2 |
| Child #25 | 0 | 0 | -1 | -1 | -1 | 48 | 2 |
| Child #26 | 2 | 0 | -1 | 1 | 0 | 60 | 2 |
| Child #27 | 2 | -1 | 1 | 2 | 1 | 60 | 2 |
| Child #28 | 2 | 1 | -1 | 2 | 1 | 60 | 2 |
| Child #29 | 1 | 0 | 0 | 1 | 1 | 60 | 1 |
| Child #30 | 3 | 0 | -1 | 2 | 1 | 48 | 2 |
| Child #31 | 0 | 0 | 2 | 2 | 2 | 48 | 2 |
| Child #32 | 2 | 1 | 1 | 4 | 3 | 60 | 3 |
| Child #33 | 2 | -2 | 0 | 0 | -1 | 60 | 3 |
| Child #34 | 3 | -2 | 1 | 2 | 1 | 60 | 2 |
| Child #35 | 1 | -1 | -1 | -1 | -1 | 60 | 1 |
| Child #36 | 0 | 1 | 0 | 1 | 1 | 36 | 1 |
| Child #37 | 0 | 1 | 0 | 1 | 1 | 60 | 2 |
| Child #38 | 4 | -1 | -3 | 0 | -1 | 60 | 1 |
| Child #39 | 1 | 0 | -4 | -3 | -3 | 60 | 1 |
| Child #40 | 0 | 1 | -1 | 0 | 0 | 36 | 1 |
| Child #41 | 0 | 3 | 0 | 3 | 3 | 48 | 3 |
| Child #42 | -2 | 1 | 1 | 2 | 2 | 60 | 1 |
| Child #43 | 0 | 2 | 0 | 2 | 2 | 48 | 2 |
| Child #44 | 1 | -1 | 1 | 1 | 1 | 36 | 2 |
| Child #45 | 4 | 0 | 0 | 4 | 2 | 36 | 2 |
| Child #46 | 2 | -1 | 0 | 1 | 0 | 36 | 3 |
| Child #47 | 1 | 1 | 0 | 2 | 2 | 48 | 2 |
| Child #48 | 2 | -1 | 0 | 1 | 0 | 60 | 2 |
| Child #49 | 1 | -1 | 1 | 1 | 1 | 36 | 2 |
| Child #50 | 0 | 1 | 0 | 1 | 1 | 36 | 2 |
| Child #51 | 3 | 0 | 0 | 3 | 2 | 48 | 3 |
| Child #52 | 0 | 0 | 3 | 3 | 3 | 48 | 3 |
| Child #53 | 2 | -2 | 0 | 0 | -1 | 60 | 3 |
| Child #54 | 2 | -1 | -1 | 0 | -1 | 48 | 3 |
| Child #55 | 0 | -2 | 1 | -1 | -1 | 36 | 1 |
| Child #56 | 1 | 0 | 0 | 1 | 1 | 36 | 3 |
| Child #57 | 0 | -1 | 2 | 1 | 1 | 60 | 2 |
| Child #58 | 2 | 0 | 0 | 2 | 1 | 60 | 3 |
| Child #59 | 3 | 1 | 0 | 4 | 3 | 48 | 3 |
| Child #60 | 2 | -1 | 0 | 1 | 0 | 48 | 2 |
| Child #61 | 2 | 0 | -1 | 1 | 0 | 48 | 3 |
| Child #62 | 0 | 2 | 2 | 4 | 4 | 60 | 3 |
| Child #63 | 2 | 2 | 0 | 4 | 3 | 36 | 3 |
| Child #64 | 3 | 2 | 0 | 5 | 4 | 36 | 2 |
| Child #65 | 0 | 1 | 0 | 1 | 1 | 36 | 1 |
| Child #66 | 0 | 1 | -1 | 0 | 0 | 36 | 1 |
| Child #67 | 1 | 0 | -1 | 0 | 0 | 48 | 1 |
| Child #68 | 0 | 1 | 1 | 2 | 2 | 60 | 3 |
| Child #69 | 0 | 3 | -1 | 2 | 2 | 48 | 1 |
| Child #70 | 1 | -1 | 1 | 1 | 1 | 60 | 1 |
| Child #71 | 1 | 0 | 0 | 1 | 1 | 60 | 3 |
| Child #72 | -1 | 1 | 1 | 2 | 2 | 48 | 2 |
| Child #73 | 1 | -1 | 2 | 2 | 2 | 60 | 1 |
| Child #74 | 4 | -1 | -1 | 2 | 1 | 60 | 2 |
| Child #75 | 0 | 0 | 1 | 1 | 1 | 60 | 3 |
| Child #76 | 2 | 0 | 0 | 2 | 1 | 36 | 3 |
| Child #77 | 2 | 0 | 1 | 3 | 2 | 60 | 3 |
| Child #78 | 0 | 0 | 0 | 0 | 0 | 48 | 1 |
| Child #79 | 0 | 3 | -1 | 2 | 2 | 48 | 3 |
| Child #80 | 1 | 1 | 0 | 2 | 2 | 60 | 1 |
| Child #81 | 3 | -1 | -1 | 1 | 0 | 48 | 3 |
| Child #82 | 2 | -2 | 1 | 1 | 0 | 60 | 1 |
| Child #83 | 0 | 2 | 0 | 2 | 2 | 60 | 1 |
| Child #84 | 1 | -1 | 1 | 1 | 1 | 48 | 3 |
| Child #85 | 2 | 0 | 1 | 3 | 2 | 60 | 2 |
| Child #86 | 1 | -1 | -1 | -1 | -1 | 48 | 1 |
| Child #87 | -1 | 1 | 0 | 1 | 1 | 48 | 2 |
| Child #88 | 1 | 0 | 0 | 1 | 1 | 60 | 3 |
| Child #89 | 3 | -1 | 0 | 2 | 1 | 60 | 3 |
| Child #90 | 1 | 0 | 0 | 1 | 1 | 60 | 3 |
| Child #91 | 0 | 0 | 0 | 0 | 0 | 60 | 2 |
| Child #92 | 2 | 0 | 0 | 2 | 1 | 60 | 2 |
| Child #93 | -1 | -1 | 1 | 0 | 0 | 48 | 1 |
| Child #94 | -1 | 2 | 0 | 2 | 2 | 60 | 1 |
| Child #95 | 2 | -1 | -1 | 0 | -1 | 48 | 1 |
| Child #96 | -1 | 0 | 1 | 1 | 1 | 60 | 1 |
| Child #97 | 2 | 2 | 0 | 4 | 3 | 60 | 3 |
| Child #98 | 2 | -1 | 0 | 1 | 0 | 60 | 1 |
| Child #99 | 4 | 0 | 0 | 4 | 2 | 60 | 3 |
| Child #100 | 1 | 0 | 0 | 1 | 1 | 60 | 2 |

In 82 of the 100 cases, the evaluation of FTTell was similar to the evaluation of the pediatric gastrologist resident. In the remaining 18 case, in 5 cases the FTTell was in agreement with the clinical assessment.

In 9 of the 18 cases, FTTell determined that there is no FTT since the child did not cross percentiles and at the last measurement the child’s weight was at the fifth percentile or higher. Nevertheless, the expert assessment was that the child is at risk to develop FTT and needs follow up. since at some point the child's weight was below the third percentile.

In four cases FTTell determined that there is FTT because the child crossed more than two percentiles, while the expert determined that there is no FTT, because the first percentiles were crossed right after birth or at 6 months, after which the child’s weight stabilized.

# **SUMMARY**

Conceptual modeling supports the development of the qualitative aspect—what is the benefit of the systems to its beneficiaries and other stakeholders, what is the system’s main function—the main process and the operand(s) which that process transforms to provide the benefit, what are the subprocesses of the main process and the objects that they require or transform, what are the environmental objects and processes and how they interact with the system, etc.

We have presented a proof-of-concept for a first step towards fusing systems engineering and software engineering. We did so by developing a Methodical Approach to Executable Integrated Modeling—MAXIM, which extends OPM ISO 19450:2015 by incorporating executable computational objects and processes into representations of physical objects and processes.

Traditionally, physical things are modeled using a conceptual modeling language, while the calculations use a different tool, such as MATLAB or Modelica. The modeling language that comes closest to what we propose here is SysML, where one can express computational aspects using the dedicated static parametric diagram. Yet, to the best of our knowledge, even with SysML, it is not possible to show how system values change over time during a simulated execution of the system as we have presented here. Even if such execution exists or shall be developed, it must involve several kinds of SysML diagrams, cognitively burdening the human inspector, who has to view, follow, and comprehend the simulation as it dangles from one diagram kind to another. Synchronization and consistency among all the different SysML diagram kinds with each change in the model is an issue that would require a lot of attention.

While parts of MAXIM models may be modeled by different SysML diagrams, the benefit of modeling the entire system in a single kind of diagram is lost, as the modeler must cognitively unify the various system aspects, each represented in a different set of notations. One can use swim lanes in a SysML activity diagram, possibly allowing software and hardware engineers to model their respective aspects in parallel, but activity diagram is only one of nine kinds of SysML diagrams, each representing a different aspect of the system. This leaves all the parties involved in the modeling with the model multiplicity problem [81] — the problem of cognitively fusing all the diagram kinds to obtain a coherent view of the system’s structural and procedural aspects, let alone executing it conceptually and computationally as we propose here. Providing all the additional aspects, in the same model does not lead to readability issues, because one of OPM principles is that a single diagram should not be overloaded. If more model facts need to be represented, we refine in a new diagram (by zooming into or unfolding) the process or object that needs elaboration. This way, no single OPM diagram is too overloaded with details, and the model can grow indefinitely in the number of interrelated diagrams and detail (nesting) levels.

The common model goes beyond traditional conceptual modeling in that (1) it enables seamless combination of conceptual modeling with computational aspects, which currently are considered only downstream, and (2) it is executable both computationally and visually, so while execution is underway, the modeler can watch and debug both the operational logic and the values as they are being updated.

The ability of the system designers to visually inspect the animated execution of the model, see how objects are generated and consumed, and how specific values of parameters change over time, is a major benefit of our approach. It enables testing the system “in silico”, i.e., using computing power, before cutting any metal and building any physical part of the system, saving precious resources and providing a holistic view on the entire system. This enables engineers from disparate domains to jointly inspect the execution as it runs and analyze the results to spot both qualitative and quantitative problems. These problems can then be resolved early on, before they propagate downstream and become too costly, potentially posing real threats to the project success.

During our research we came across the realization that there is a model fidelity hierarchy, which progresses from spoken to written text, and from there to conceptual and computational models, culminating in the execution of an executable model. With each transition from one level in this model fidelity hierarchy to the next, the accuracy requirements increase, causing a new set of errors to be detected and corrected.

Our study has demonstrated that there is important synergy between system-level conceptual qualitative modeling and component- and discipline-level design, where more concrete and quantitative considerations come to play. Adding the quantitative layer on top of the qualitative one helps in discovering design errors that would otherwise be either detected at a later stage in the development process of the system, incurring significant costs and project delays, or worse yet, escape detection and remain uncorrected in the system, causing even worse, sometimes detrimental or fatal damage during the system’s use and service. This approach is information lossless because we keep building and adding details on top of the current evolving model. The execution layer on top of both the qualitative and quantitative modeling helps in further identifying problems in the system. Specifically, we have described two transitions in the modeling process that not only make it more grounded and concrete, but, not less importantly, are most instrumental in revealing errors in the system’s conceptual model. The first transition is from conceptual, qualitative to computational, quantitative modeling, during which we seamlessly incorporate into the conceptual model executable code of mathematical equations that express the underlying physics and engineering considerations. The code, embedded into leaf-level processes in a model, can be of any complexity level, and include not just mathematical formulae, but also any control flows such as conditions and loops. This first kind of transition, from conceptual to computational model, revealed conceptual modeling errors and triggered changes in the model. The second transition is moving from the combined qualitative-quantitative model to running the execution of that model with the objective of calculating system parameter values. When actual values are assigned to input objects, the computation is performed by executing the model, yielding sought parameter values. Errors found at this stage can be not just wrong inputs, but also flaws in the formulae or even errors that were caused earlier on, during the conceptual modeling stage. The ability to spot and correct errors as early as possible in the system’s lifecycle is of paramount importance, since the cost of correcting errors grows exponentially with the system lifecycle stage [82].

In this research we used MAXIM to model an Airbus aircraft braking system as a case in point. By doing it we demonstrated how conceptual modeling and computations are seamlessly combined in a straightforward, natural way, opening the way to the construction of a new, end-to-end systems engineering environment that does not mandate abandoning the conceptual model when transitioning to detailed design, in which computations are necessary and prevalent.

We have shown the viability and potential value of MAXIM as an extension of OPM by using it to model a simplified version of a real-life complex braking system of a large commercial aircraft designed and manufactured by Airbus. Consider a modern digital control system, like the ones found in today's civil airplanes, such as air conditioning, fuel supply, flight control, or braking and steering. These are CPSs that typically consist of a fabric comprising physical parts, a digital platform, and a communication network. While the individual physical parts are specific to the system under consideration, the digital platform and the communication network are shared resources used by many systems with different dynamics. Understanding the interactions between the different parts of a complex networked system is one of the most important goals of defining the architecture.

This understanding is facilitated by the capabilities of MAXIM. Both physical (analog) and informatical (digital) parts can be modeled together, and their dynamics can be explored using the combined simulation and computation capabilities. As MAXIM allows to easily traverse up and down the different levels of details in a complex architecture, from the overall system level through the various subsystem levels down to the equipment level, communication between system architects, safety and security specialists, human factors engineers, and equipment designers improves significantly.

Two papers that we wrote during this doctoral work have shown the effectiveness of MAXIM in detecting inaccuracies in models. In the first paper [26] we developed a model that captured the qualitative and quantitative aspects of the system under consideration, where these two aspects were modeled and kept separately using different languages and tools. In that work, MAXIM has not been developed or considered at all.

In a follow-up paper [83] we developed a model for the same system and captured both the qualitative and quantitative aspects using MAXIM. MAXIM integrates the system’s conceptual aspect with its computational one. Together, these two aspects provide the basis for execution and for the model fidelity hierarchy. The result of this seamless aspects integration is a comprehensive model that is conducive to model verification and error detection: Using the model fidelity hierarchy, we were able to find conceptual errors when the computational part was added, and computational errors when the execution of the system was performed. Importantly, the errors detected in [26] were not detected without MAXIM, because MAXIM provides the scaffolding for the model fidelity hierarchy introduced in [83].

The two papers, [26] and [83], have provided a favorable evaluation of the effectiveness of MAXIM: Developing the same system with and without MAXIM has clearly demonstrated the benefit of the qualitative-quantitative modeling approach in the context of system verification and error detection.

Further, we came to the understanding that conceptual models have been used mostly for descriptive or prescriptive purposes, serving for understanding and specifying phenomena and systems, respectively. In this research, we propose using a diagnostic model – the application of conceptual modeling for the purpose of medical diagnosis. Currently, high-quality diagnosis of FTT potential depends on the experience and expertise of the pediatrician. Hence, as a proof-of-concept to our diagnostic modeling approach, we have developed, implemented, and tested FTTell – a model-based diagnosis system for FTT potential assessment and diagnosis using MAXIM. Using this cloud-based modeling environment has enabled us to transition back-and-forth between qualitative and quantitative modeling in a seamless and effortless fashion. The importance of this research is that it provides pediatricians with an accessible, easy-to-use decision- making tool that can help them determine objectively if a child they treat suffers from FTT. The objectivity of the method stems from the fact that it takes as input quantitative measures of weight at specific time points along the child’s early development. Using this tool, the pediatrician does not need to be FTT expert, since the FTTell diagnosis algorithm embeds the formal FTT definitions of recognized healthcare bodied, such as WHO and CDC, as well as all the state-of-the-art knowledge, documented in the medical literature. The input needed for the computation is easy to insert, and the execution is done by a click of a button in a friendly user interface.

# **CONCLUSION**

Adopting this integrated evolutionary approach, as the software parts improve by eliminating errors, modeled hardware parts are gradually replaced by their physical counterparts, until a complete working prototype is achieved. This is a significant step forward, ahead of the current practices of even the most advanced agile MBSE approaches that large and smaller corporations are currently struggling to adopt and practice.

Adopting OPM with MAXIM, corporations will get added value as they will be able to fuse systems engineering with software engineering, overcoming the “Grand Canyon” that currently separates these two sister disciplines. System engineering and software engineering are based on the same systems science principles; they need and complement each other. While a modeler may be able to express values in fUML or formulas in SysML, or execute the model using an external tool, doing it all in a single model using a single tool, is a major added value of MAXIM.

The proof-of-concept framework we are proposing in this research is a first step towards accomplishing this unified framework goal. We have shown the feasibility of integrating conceptual and computational model elements in a concrete, real-life example of conceptually modeling an aircraft braking system and computing its critical parameters, such as Braking Force and Speed.

By developing the model fidelity hierarchy of systems, we provide two important implications:

(1) The model fidelity hierarchy of systems has four levels: (a) Ideation and informal idea communicating – At the most abstract and fuzzy level there are the processes of ideation, verbal, and written text specification. (b) Conceptual-qualitative modeling – The model of the system resulting from this process expresses the specification formally. In our case, this model was created using OPCloud, which applies OPM ISO 19450. (c) Computational augmenting and enhancing – The conceptual-qualitative model is augmented and enhanced with quantitative computational elements. In our case, these were integrated seamlessly into the conceptual model using MAXIM on the same OPCloud platform. (d) Model executing – executing the model to ascertain that the conceptual-qualitative-and-quantitative model runs as expected. Different kinds of mistakes are discovered and corrected while transitioning from one level to the next, which were not discovered at the earlier level despite intensive checks and best efforts, because they are so subtle that only the fidelity required at the next level exposes them. Figure 84 is an OPD of the Model Fidelity Hierarchy, which graphically summarizes our main findings.

(2) Modeling a complex system, including both its conceptual-qualitative and computational-quantitative parts, using the same modeling paradigm, has the advantage of a streamlined, lossless process. The usage of OPM has made it possible to reveal the model fidelity hierarchy and enabled level transitions to be information lossless, providing the most value while requiring the minimal effort. The value added by the additional modeling layers with their computation and simulation capabilities is that they enable the creation of a seamless continuum, spanning the spectrum that starts with the abstract level of business requirements and goes all the way to specialized disciplinary engineering considerations.

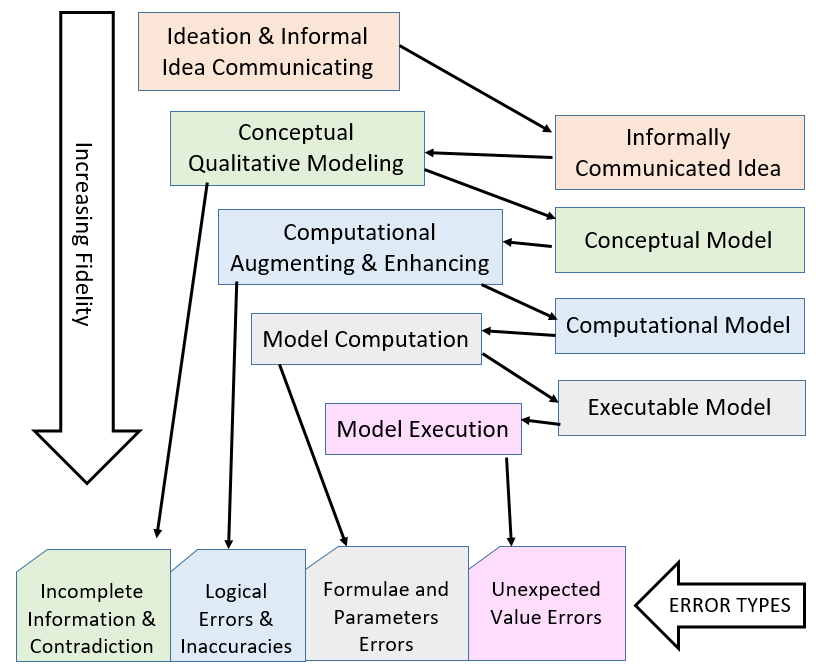


Figure 84. Model Fidelity Hierarchy diagram

Using MAXIM and conclusions made from the developed model fidelity hierarchy, we developed FTTell, which can easily and accurately diagnose FTT in children until they are five years old, basing on perinatal and postnatal periods. Since FTTell is web-based, its model and algorithms can be continuously and transparently updated with recent published data, and the diagnosis system can be operated simply and intuitively by pediatricians or nurses on their teams, knowing that they always use the most informed and up-to-date system with no need to upgrade or download new versions.

# **FUTURE WORK**

Future research should scale up this activity by considering other effectiveness measures, such as correctness and performance, to show that the differences between not using the MAXIM approach and using it are significant. The MAXIM approach can be further evaluated by involving two groups of engineers, one of which will be tasked with performing a combined conceptual-computational assignment, such as the one presented in this work - the braking system model, using MAXIM, while the other will use the traditional approach of separating the conceptual modeling from the computational one. The research will compare the performance of the two groups using both objective metrics, such as accuracy, loss of information, and the time to perform the assignment, and these engineers’ subjective evaluations of the level of accessibility and required intellectual effort.

Other future research and development directions should include the following:

1. Complete OPCloud’s execution capabilities by accounting for the complete OPM operational semantics, including events, conditions, execution paths, probabilities, and process invocations.
2. Support recursion. Currently, as OPCloud is implemented to support MAXIM, recursion may be expressed by a loop using invocation link. More research should be done to tackle The Halting Problem and proving or refuting the assumption that MAXIM is Turing-complete.

(3) Improve OPCloud’s model execution visualization, providing for stepwise and reverse execution with breakpoints to aid modelers in “debugging” the system.

(4) Enabling external engineering software packages that specialize in specific domains, such as computational fluid dynamics (CFD), structural strength, and heat transfer analysis, in order to provide for an extended simulation that accounts for these aspects, adopting the subcontracting approach [18].

(5) A complete reflective metamodel [84] of OPM [85] is presented in Appendix C of ISO 19450 [3]. The MAXIM extension of OPM is scheduled to be incorporated into the OPM reflective metamodel in the new version of ISO 19450 planned for 2021-2022. The updated metamodel will include an additional attribute of an OPM Informatical Thing (Object or Process) called Computability, with two possible values – computable and non-computable. These enable the specialization of Informatical Thing into a Non-Computable Thing and a Computable Thing, where the Essence attribute of the latter can be only informatical.

In context to the model fidelity hierarchy, our plans are extending the hierarchy to include a fifth level – Hardware in the Loop [86], where we combine OPM and its MAXIM extension to execute models to which hardware, including robots, is connected, getting input from sensors and commanding actuators to respond in real time. In addition, we plan to further characterize the various errors discovered at each fidelity level and apply the findings in OPCloud to improve the modeling process. As for extending OPM, we suggest two actions: (1) Implementing the generalization-specialization and the inheritance it induces, as specified in [1][2]. This will eliminate the need to model separate sets of attributes for specialized objects, such as the nose and main landing gears in our case study, while still being able to input and get separate parameter values for each set. Enabling this option might involve also the OPM path mechanism. (2) Enable automated batch execution, in which a massive amount of object input values will be fed from synthetic input values, generated randomly from a probability distribution function that mimics the behavior of the real system. By automatically executing the model multiple times with diverse sets of values, we are much more likely to reveal additional wrongly modeled cases or errors in formulas defined as functions in computational process. Examples of such errors include division by zero and a software function having if-else conditions with input values that fit neither the if nor the else part of the condition, yielding an 'undefined' result value that leads to revealing the error.

Future research in FTTell development can take several directions: (1) Extend FTTell to include adding input reasons that would yield mother-related FTT indications, and model an individually adapted treatment protocol that accounts for each child’s data and indications. (2) Modeling the implications of FTT on the patient’s stature in adulthood. (3) The relatively high 87% model correspondence with the diagnosis of an authoritative FTT expert can be further improved by updating the model to include in the diagnosis algorithm additional considerations based on “picking up the brains” of the expert. (4) Modify the model-based diagnostic approach to additional medical conditions and diseases that similarly require a stepwise procedure and computations using data supplied by the medical expert. (5) Apply a similar approach to the diagnosis of failure causes of technological products and systems, and how to repair or treat them.

# **Bibliography**

[1] D. Dori, *Object-process methodology: a holistic systems paradigm*, 1st ed. Verlag Berlin Heidelberg: Springer, 2002.

[2] D. Dori, *Model-based systems engineering with OPM and SysML*. 2016.

[3] “ISO/PAS 19450 Automation systems and integration -- Object-Process Methodology,” 2015. [Online]. Available: https://www.iso.org/standard/62274.html.

[4] R. E. Mayer, “The promise of multimedia learning: using the same instructional design methods across different media,” *Learn. Instr.*, vol. 13, no. 2, pp. 125–139, 2003.

[5] R. E. Mayer and R. Moreno, “Nine ways to reduce cognitive load in multimedia learning,” *Educ. Psychol.*, vol. 38, no. 1, pp. 43–52, 2003.

[6] R. Rettie and C. Brewer, “The verbal and visual components of package design,” *J. Prod. Brand Manag.*, vol. 9, no. 1, pp. 56–70, 2000.

[7] B. Selic, “The pragmatics of model-driven development,” *IEEE Softw.*, vol. 20, no. 5, pp. 19–25, 2003.

[8] A. M. Christie, “Simulation : An Enabling Technology in Software Engineering,” *CrossTalk*, 1999.

[9] L. L. Gardner, M. E. Grant, and L. J. Rolston, “Using Simulation To Benchmark Traditional Vs. Activity-Based Costing In Product Mix Decisions,” *Proc. 1994 Winter Simul. Conf.*, pp. 1050–1057, 1994.

[10] W. Böhm, S. Henkler, F. Houdek, A. Vogelsang, and T. Weyer, “Bridging the Gap between Systems and Software Engineering by Using the SPES Modeling Framework as a General Systems Engineering Philosophy,” *Procedia Comput. Sci.*, vol. 28, pp. 187–194, 2014.

[11] A. Johanson and W. Hasselbring, “Software Engineering for Computational Science: Past, Present, Future,” *Comput. Sci. Eng.*, vol. PP, no. 99, pp. 90–109, 2018.

[12] D. Dori and M. Goodman, “On bridging the analysis-design and structure-behavior grand canyons with object paradigms,” *Rep. Object Anal. Des.*, vol. 2, no. 5, pp. 25–35, 1996.

[13] A. Pyster *et al.*, “Exploring the Relationship between Systems Engineering and Software Engineering,” *Procedia - Procedia Comput. Sci.*, vol. 44, pp. 708–717, 2015.

[14] R. Turner, A. Pyster, and M. Pennotti, “Developing and validating a framework for integrating systems and software engineering,” in *2009 3rd Annual IEEE Systems Conference*, 2009, pp. 407–412.

[15] S. Bolshchikov, A. Renick, S. Mazor, J. Somekh, and D. Dori, “OPM Model-Driven Animated Simulation with Computational Interface to Matlab,” in *2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2011, pp. 193–198.

[16] “OPCloud.” [Online]. Available: https://www.opcloud.tech/.

[17] D. Dori, A. Jbara, N. Levi, and N. Wengrowicz, “Object-Process Methodology, OPM ISO 19450--OPCloud and the Evolution of OPM Modeling Tools,” *Syst. Eng. Newsl. (PPI SyEN)*, vol. 61, pp. 6–17, 2018.

[18] D. Dori, A. Renick, and N. Wengrowicz, “When quantitative meets qualitative: enhancing OPM conceptual systems modeling with MATLAB computational capabilities,” *Res. Eng. Des.*, vol. 27, no. 2, pp. 141–164, 2016.

[19] Y. Mordecai, O. Orhof, and D. Dori, “Model-Based Interoperability Engineering in Systems-of-Systems and Civil Aviation,” *IEEE Trans. Syst. Man. Cybern.*, no. 1 September, 2016.

[20] J. Somekh, G. Haimovich, A. Guterman, D. Dori, and M. Choder, “Conceptual Modeling of mRNA Decay Provokes New Hypotheses,” *PLoS One*, vol. 9, no. 9, pp. 1–14, 2014.

[21] D. Durman, “Rappid.” .

[22] R. E. Mayer, “Techniques that reduce extraneous cognitive load and manage intrinsic cognitive load during multimedia learning.,” in *Cognitive load theory.*, R. Moreno, Ed. New York, NY, US: Cambridge University Press, 2010, pp. 131–152.

[23] D. Firesmith, “Modern Requirements Specification,” *J. Object Technol.*, vol. 2, pp. 53–64, 2003.

[24] A. Blekhman, J. P. Wachs, D. Dori, and S. Member, “Model-Based System Specification With Tesperanto : Readable Text From Formal Graphics,” pp. 1–11, 2015.

[25] J. Otero and W. Kintsch, “Failures to Detect Contradictions in a Text: What Readers Believe versus What They Read,” *Psychol. Sci.*, vol. 3, no. 4, pp. 229–235, 1992.

[26] L. Li, N. Levi-Soskin, A. Jbara, M. Karpel, and D. Dori, “Model-Based Systems Engineering for Aircraft Design with Dynamic Landing Constraints Using Object-Process Methodology,” *IEEE Access*, vol. 7, pp. 1–1, 2019.

[27] I. D. Schwartz, “Failure To Thrive: An Old Nemesis in the New Millennium,” *Pediatr. Rev.*, vol. 21, no. 8, pp. 257–264, 2007.

[28] Z. Mei, L. M. Grummer-Strawn, D. Thompson, and W. H. Dietz, “Shifts in Percentiles of Growth During Early Childhood: Analysis of Longitudinal Data From the California Child Health and Development Study,” *Pediatrics*, vol. 113, no. 6, pp. e617–e627, 2004.

[29] P. Dobrev, O. Kalaydjiev, and G. Angelova, “From conceptual structures to semantic interoperability of content,” in *International Conference on Conceptual Structures*, 2007, pp. 192–205.

[30] J. A. Muguira, “The Levels of Conceptual Interoperability Model,” no. September 2003.

[31] B. Kruse and M. Blackburn, “Collaborating with OpenMBEE as an Authoritative Source Truth Environment,” *Procedia Comput. Sci.*, vol. 153, pp. 277–284, 2019.

[32] M. Tiller, *Introduction to Physical Modeling with Modelica*, 1st ed., vol. 615. Springer US, 2001.

[33] R. Klee, H., Allen, *Simulation of Dynamic Systems with MATLAB and Simulink*, 3rd ed. 2018.

[34] The Object Management Group, “Semantics of a Foundational Subset for Executable UML Models (fUML),” *Object Manag. Gr.*, no. October, p. 441, 2012.

[35] OMG group, *OMG, Systems Modeling Language (SYSML) Specification*, 1.3. 2012.

[36] “Design patterns for open tool integration,” *Softw Syst Model*, vol. 4, pp. 157–170, 2005.

[37] B. Akesson, A. Molnos, A. Hansson, J. Ambrose Angelo, and K. Goossens, “Composability and Predictability for Independent Application Development, Verification, and Execution,” in *Multiprocessor System-on-Chip*, M. Hübner and J. Becker, Eds. Springer Verlag, 2010, pp. 25–56.

[38] G. Kiczales *et al.*, “Aspect-oriented programming,” in *ECOOP’97 --- Object-Oriented Programming*, 1997, pp. 220–242.

[39] I. Akkaya, P. Derler, S. Emoto, and E. A. Lee, “Systems Engineering for Industrial Cyber-Physical Systems Using Aspects,” *Proc. IEEE*, vol. 104, no. 5, pp. 997–1012, 2016.

[40] OMG, “Object Management Group.” [Online]. Available: https://www.omg.org/.

[41] The Object Management Group, “Systems Modeling Language (SysML®) v2 Request For Proposal (RFP),” 2018. [Online]. Available: https://www.omgsysml.org/SysML-2.htm.

[42] S. J. Mellor and M. J. Balcer, “Executable and Translatable UML,” *Embed. Syst. Program.*, vol. 16, no. 2, pp. 25--30, 2003.

[43] C. L. L. and I. L. and B. P. and S. M. and I. G. Czibula, “Using a fUML Action Language to Construct UML Models,” in *2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 93–101.

[44] E. Seidewitz, “UML with Meaning: Executable Modeling in Foundational UML and the Alf Action Language,” in *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, New York: ACM, 2014, pp. 61--68.

[45] Specification, “Action Language for Foundational UML ( Alf ) Concrete Syntax for a UML Action Language,” 2011.

[46] O. M. G. OMG, “Unified Modeling Language (OMG UML), infrastructure.” version 2.4. 1. Tech. rep., Object Management Group, 2011.

[47] O. M. G. OMG, “Unified Modeling Language (OMG UML), superstructure.” version 2.4. 1. Tech. rep., Object Management Group, 2011.

[48] D. Thomas, “MDA: Revenge of the Modelers or UML Utopia?,” *IEEE Softw.*, vol. 21, no. 3, pp. 15–17, 2004.

[49] R. Gelbard, D. Teeni, and M. Sade, “Object-oriented analysis: Is it just theory?,” *IEEE Softw.*, vol. 27, no. 1, pp. 64–71, 2009.

[50] B. Dobing and J. Parsons, “How UML is used,” *Commun. ACM*, vol. 49, no. 5, pp. 109–113, 2006.

[51] R. France and B. Rumpe, “Does model driven engineering tame complexity?” Springer-Verlag, 2007.

[52] R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg, “Model-driven development using UML 2.0: promises and pitfalls,” *Computer (Long. Beach. Calif).*, vol. 39, no. 2, pp. 59–66, 2006.

[53] D. Dori, “Why significant UML change is unlikely,” *Commun. ACM*, vol. 45, no. 11, pp. 82–85, 2002.

[54] A. Nugroho and M. R. V Chaudron, “A survey into the rigor of UML use and its perceived impact on quality and productivity,” in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, 2008, pp. 90–99.

[55] “ISO / IEC JTC1 / SC7 N2683 PDTR Ballot PDTR 19760 Systems Engineering – Guide for ISO / IEC 15288 ( System Life Cycle Processes ),” 2002.

[56] D. Gianni, A. D’Ambrogio, and A. Tolk, *Modeling and Simulation-Based Systems Engineering Handbook*. .

[57] M. L. Federici, S. Redaelli, and G. Vizzari, “Models, Abstractions and Phases in Multi-Agent Based Simulation.,” in *WOA*, 2006.

[58] K. Forsberg and H. Mooz, “The relationship of system engineering to the project cycle,” in *INCOSE International Symposium*, 1991, vol. 1, no. 1, pp. 57–65.

[59] A.-P. Bröhl, *Das V-Modell: Der Standard für die Softwareentwicklung mit Praxisleitfaden*. Oldenbourg, 1993.

[60] Wikipedia, “Systems modeling language.” [Online]. Available: https://en.wikipedia.org/wiki/Systems\_Modeling\_Language. [Accessed: 09-Feb-2015].

[61] J. Cabot and M. Gogolla, “Object constraint language (OCL): a definitive guide,” in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, 2012, pp. 58–90.

[62] F. Jouault, J. Bézivin, and M. Barbero, “Towards an advanced model-driven engineering toolbox,” *Innov. Syst. Softw. Eng.*, vol. 5, no. 1, pp. 5–12, 2009.

[63] D. Alonso, C. Vicente-Chicote, J. A. Pastor, and B. Álvarez, “Stateml+: From graphical state machine models to thread-safe ada code,” in *International Conference on Reliable Software Technologies*, 2008, pp. 158–170.

[64] W. Schäfer and H. Wehrheim, “Model-driven development with mechatronic uml,” in *Graph transformations and model-driven engineering*, Springer, 2010, pp. 533–554.

[65] Microsoft, “Architectural patterns and style.” [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658117(v=pandp.10). [Accessed: 29-Aug-2018].

[66] Wikipedia, “Lifecycle modeling language.” [Online]. Available: https://en.wikipedia.org/wiki/Lifecycle\_Modeling\_Language. [Accessed: 30-Aug-2018].

[67] Y. M. and D. Dori, “Minding the Cyber-Physical Gap: Model-Based Analysis and Mitigation of Systemic Perception-Induced Failure,” *Sensors*, vol. 17(1644), 2017.

[68] Y. Mordecai and D. Dori, “6.5. 1 I5: A Model-Based Framework for Architecting System-of-Systems Interoperability, Interconnectivity, Interfacing, Integration, and Interaction,” in *INCOSE international symposium*, 2013, vol. 23, no. 1, pp. 1234–1255.

[69] R. A. Greenes, S. Tu, A. A. Boxwala, M. Peleg, and E. H. Shortliffe, “Toward a shared representation of clinical trial protocols: Application of the GLIF guideline modeling framework,” in *Cancer Informatics*, Springer, 2002, pp. 212–228.

[70] T. A. Pryor and G. Hripcsak, “The arden syntax for medical logic modules,” *Int. J. Clin. Monit. Comput.*, vol. 10, no. 4, pp. 215–224, 1993.

[71] J. Fox, N. Johns, and A. Rahmanzadeh, “Disseminating medical knowledge : the PRO forma approach.”

[72] S. W. Tu and M. A. Musen, “Modeling data and knowledge in the EON guideline architecture Modeling Data and Knowledge in the EON Guideline Architecture,” no. May 2014, 2001.

[73] P. Terenziani, S. Montani, A. Bottrighi, M. Torchio, G. Molino, and G. Correndo, “The GLARE approach to clinical guidelines: Main features,” *Stud. Health Technol. Inform.*, vol. 101, no. May 2014, pp. 162–166, 2004.

[74] C. Guo, S. Ren, Y. Jiang, P. L. Wu, L. Sha, and R. B. Berlin, “Transforming Medical Best Practice Guidelines to Executable and Verifiable Statechart Models,” *2016 ACM/IEEE 7th Int. Conf. Cyber-Physical Syst. ICCPS 2016 - Proc.*, 2016.

[75] “yakindu statechart tools.” [Online]. Available: https://www.itemis.com/en/yakindu/state-machine/.

[76] “Y2U,” 2009. [Online]. Available: http://www.cs.iit.edu/~code/software/Y2U/.

[77] B. Goldsmith and W. W. Norton, “The International System of Units (SI),” *Chem. Int.*, 2006.

[78] I. Graessler, J. Hentze, and T. Bruckmann, “V-models for interdisciplinary systems engineering,” *Proc. Int. Des. Conf. Des.*, vol. 2, pp. 747–756, 2018.

[79] “weight\_for\_age.” [Online]. Available: https://www.who.int/childgrowth/standards/weight\_for\_age/en/.

[80] R. J. Kuczmarski, *CDC growth charts: United States*, no. 314. US Department of Health and Human Services, Centers for Disease Control and~…, 2000.

[81] M. Peleg and D. Dori, “The model multiplicity problem: experimenting with real-time specification methods,” *IEEE Trans. Softw. Eng.*, vol. 26, no. 8, pp. 742–759, 2000.

[82] A. Nasa and J. Space, “Error Cost Escalation Through the Project Life Cycle,” 2019.

[83] N. Levi-Soskin, A. Jbara, and D. Dori, “The Model Fidelity Hierarchy: From Text to Conceptual, Computational, and Executable Model,” *IEEE Syst. J.*, 2020.

[84] OMG group, “Meta-Modeling and the OMG Meta Object Facility ( MOF ),” pp. 1–7, 2017.

[85] I. Reinhartz-Berger and D. Dori, “A Reflective Meta-Model of Object-Process Methodology: The System Modeling Building Blocks,” in *Business systems analysis with ontologies*, IGI Global, 2005, pp. 130–173.

[86] H. Kohen and D. Dori, “Incorporating Hardware-in-the-Loop Simulation into Object-Process Methodology,” in *14th Annual IEEE International Systems Conference (SysCon2020)*, 2020.

צמצום הפער בין הנדסת מערכות להנדסת תוכנה על ידי שילוב חישובים במתודולוגיית עצמים-תהליך.

חיבור על מחקר

לשם מילוי חלקי של דרישות לקבלת התואר

דוקטור לפילוסופיה

נטלי לוי

הוגש לסנט הטכניון – מכון טכנולוגי לישראל

|  |  |  |
| --- | --- | --- |
| שבט תשפ"א | חיפה | ינואר 2021 |

מחקר זה נערך בפקולטה להנדסת תעשיה וניהול בהנחיית הפרופסור דב דורי וד"ר אחמד ג'בארה

מחקר זה התאפשר בסיוע תמיכתם הכספית הנדיבה של הגופים הבאים:

הטכניון – מכון טכנולוגי לישראל

מרכז ברנרד מ. גורדון להנדסת מערכות בטכניון – מכון טכנולוגי לישראל

תקציר

המחסור בשפת מידול הניתנת להרצה המשלבת הנדסת מערכות, הנדסת תוכנה ואף תחומי הנדסה נוספים מהווה סיבה עיקרית לבעיות וכשלים בתהליך פיתוח מערכת. בנוסף, הגישה הקיימת כיום להנדסת מערכות מבוססות מודלים מסתמכת על מגוון סוגי מודלים כאשר לכל אחד יש רמת נכונות ורמת דיוק משלו. המודלים השונים מופרדים לחלוטין או משולבים חלקית במקרה הטוב. הפער הנוצר בין שלב מידול המערכת לשלב פיתוח התוכנה עבורה גורם לפגיעה באמינות המערכת, היות ופעמים רבות התוכנה המפותחת אינה תואמת למודל של המערכת בה התוכנה אמורה לתמוך.

במסגרת עבודת מחקר זו אנו מנסים להתגבר על הפער הנוצר בין הנדסת תוכנה להנדסת מערכות על ידי שילוב יכולות חישוביות ויכולת הרצת סימולציה לתוך מתודולוגיית עצמים-תהליכים, OPM, המוכרת כתקן ISO-19450 וכמתודולוגיה מובילה הן למידול רעיוני והן להנדסת מערכת מבוססת מודלים. כתוצאה מכך אנו מקבלים שפת מידול אחידה, בעלת יכולות רעיוניות וחישוביות וניתנת להרצה.

אנו מגשרים על הפער הנוצר בין הנדסת מערכת להנדסת תוכנה באמצעות הרחבת OPM על ידי גישה מתודולוגית למידול משולב, הניתן להרצה – MAXIM. אנחנו מדגימים את השימוש ב MAXIM על ידי בניית מודל חישובי, הניתן להרצה, למערכת בלמים של כני נסע במטוס נוסעים שפותח על ידי חברת איירבוס. כפי שניתן לראות בדוגמא זו, באמצעות MAXIM, מהנדסים מתחומים שונים יכולים לשתף פעולה כבר בשלבים ההתחלתיים של פיתוח המערכת וביחד לבנות מודל הוליסטי המשלב היבטים איכותניים וכמותיים מהתחומים השונים.

תוך כדי פיתוח המתודולוגיה של MAXIM הגענו להבנה בנוגע לקיום היררכיית מהימנות המודל. בתחתית ההיררכיה נמצאת השפה המדוברת, בצורה מופשטת ולא פורמלית. לאחר השפה המדוברת נמצא טקסט הכתוב במסמך, לאחר מכן מודל קונספטואלי, בהמשך ההרחבה שלו עם יכולות חישוביות ולבסוף מודל הניתן להרצה. אנו מדגימים את קיום ההיררכיה במקרה בוחן המתאר כן נסע של מטוס. ככל שמתקדמים בתהליך המידול ברמות ההיררכיה השונות, מתגלות שגיאות מסוגים שונים בכל רמה. היכולת של גישת המידול הרציפה ש OPM וההרחבה של MAXIM לגילוי שגיאות בדיוק מתגבר בין הרמות השונות הינה עוצמתית ומשמעותית ביותר; היות ותיקון שגיאות אשר מתגלות בשלבים מוקדמים יותר הינו זול יותר, בכמה סדרי גודל, לעומת תיקון שגיאות המתגלות בשלבי מידול ופיתוח מתקדמים.

העקרונות של MAXIM והפעולות הניתנות לביצוע ממומשים ב OPCloud – אתר שיתופי למידול מודלים קונספטואליים בשפת המידול OPM. מקרה הבוחן של כו הנסע ממחיש את מה שייתכן והינו צעד חשוב לקראת הצורך הקריטי הגדל לשילוב הנדסת מערכות והנדסת תוכנה בצורה המאפשרת מעבר חלק משלב הארכיטקטורה ברמה העליונה והמופשטת לאיפיון מפורט לכל תחום בנפרד.

דבר נוסף אותו מאפשרת OPM ההרחבה של MAXIM הוא סוג חדש של מודלים. כיום מודלים בדרך כלל מתארים או מערכת שמתכננים לבנות, כולל המבנה, הפונקציונאליות והתנהגות או מערכת קיימת שרוצים להבין אין היא עובדת ולנתח אותה. OPM עם ההרחבה של MAXIM מציג מתודולוגיה קונספטואלית-חישובית חדשה המאפשרת לפתח ולעסוק בסוג שלישי, חדש של מודלים – מודלים דיאגנוסטיים. לצורך הצגת הסוג החדש בנינו מודל להערכת פוטנציאל של חוסר שגשוג בילדים (FTT) בעת שלב ההריון והלידה ושלב הילדות. למרות ש FTT הינו שכיח בילדים קטנים ונחקר רבות, הגורמים לתופעה לעיתים קרובות אינם ברורים. רופאי ילדים עלולים לפספס פעוטות וילדים עם חוסר שגשוג ובעיקר את המקרים הגבוליים שבהם. הפתרון המושלם לכך הינו מערכת בה רופא הילדים יכול להזין את נתוני המטופל בכלי יחיד ולקבל הערכה לגבי הפוטנציאל של הילד לחוות חוסר שגשוג. כחלק מעבודת מחקר זו פיתחנו את FTTell – כלי מבוסס מודל המקבץ את הידע הקיים ומבצע דיאגנוזה. הכלי ניתן להרצה, ובו ההיבטים האיכותניים והפרמטרים החישוביים ממודלים יחד באמצעות MAXIM, מתמקדים גם בתקופת ההריון והלידה וגם בתקופת הילדות, עד גיל חמש. תוצאות הדיאגנוזה של הכלי שפותח ניתנות להשוואה ולחוות דעת של מומחה בתחום.

היעילות של הכלי בדיאגנוזה של ילדים בעת תקופת הילדות מודגמת על ידי מידע הנאסף בנוגע ל 100 ילדים. לכל ילד הכלי חישב ציון המורכב משני חלקים: הקיום של FTT והחומרה שלו במידה והוא קיים. השווינו את תוצאות המודל לדיאגנוזה שבוצעה על ידי מומחה בתחום. ב 82 מתוך 100 מקרים, האבחון של המודל ושל המומחה היה זהה. המקרים בהם התוצאה של המודל לא תאמה את אבחון המומחה, נבחנו על ידי ראש תחום ונמצאו עוד 5 מקרים בהם המודל צדק. FTTell יכול לשמש בצורה יעילה ככלי אבחון ל FTT המגביר את דיוק הדיאגנוזה של רופאי הילדים ומאפשר דיאגנוזה מדוייקת.

המודל הדיאגנוסטי שלנו יכול להתעדכן כל הזמו בממצאים ומחקרים חדשים, בצורה שקופה למשתמש. רופאי ילדים יכולים להשתמש במודל שפיתחנו כדי שפר את דיאגנוזת ה FTT שלהם מה שיעזור להתערב בזמן ובכך למזער את הנזק העלול להיגרם לילד.

תוך כדי העבודה על המודל, השתמשנו בהיררכיה הנאמנות שלמדנו בכדי לשפר את המודל ואת התוכנה המבוססת על MAXIM ומשולבת בתוך המודל.

הפיתוח של MAXIM מהווה הרחבה גדולה ומשמעותית ביותר ל OPM המעלה אותה משפת מידול קונספטואלי למתודולוגיה המשלב מידול קונספטואלי וחישובי.

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. <https://www.opcloud.tech/> [↑](#footnote-ref-1)