Document title: The Arrowhead Tools Data Semantics Catalogue: An Initial Evaluation

Version	Status	Date
2.1	final	2020-12-03

Author	Contact
Géza Kulcsár	geza.kulcsar@incquerylabs.com

The Arrowhead Tools Data Semantics Catalogue: An Initial Evaluation

IQL: Géza Kulcsár
geza.kulcsar@incquerylabs.com

Abstract

This document contains an overview of the general concepts on how data semantics is interpreted in the Arrowhead Tools project, serving as a conceptual basis for interoperability notions. In particular, it provides a basic high-level definition of semantics itself, identifies three conceptual categories where semantics might be considered, and, finally, gives a guideline for identifying/placing a particular approach in this semantic framework.

ECSEL EU project 826452 - Arrowhead Tools
Project Coordinator: Professor Jerker Delsing | Luleå University of Technology

Table of contents

1. Introduction 3

2. General remarks: Semantics as toolchain interfaces 3

3. The Arrowhead Tools Interface Catalogue: An Initial Assessment 5

3.1 Systems Modeling 5

3.2 Ontology Modeling 7

3.3 Translation 10

4. Revision history 11

4.1 Contributing and reviewing partners 11

4.2 Amendments 12

4.3 Quality assurance 12

1. Introduction

The present document serves as a successor of documents **O4** and **O5** in the previous delivery of Work Package 4 (**D4.1**). This is a high-level summary of the intendedly loose collaboration taking place in **Task 4.2**.

The focus of the Arrowhead Tools semantic landscape is characterized by an increased understanding of the parallel co-existence of all the different approaches being of relevance to system-of-systems interoperability in the Arrowhead sense.

Therefore, the present document unifies general observations of semantics with actual examples, being an incremental improvement of the previous O4 and O5 documents, fulfilling the most crucial pieces of future work outlined earlier.

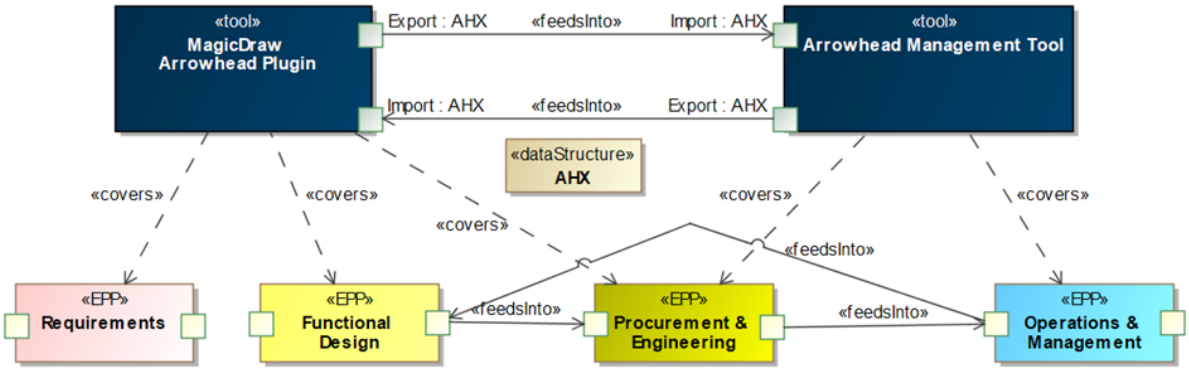
2. General remarks: Semantics as toolchain interfaces

Interoperability is a core concept of Arrowhead, addressing not only (actually, typically not) the communication between IoT devices within an SoS, but also the ways data are being exchanged between different tools within the scope of Arrowhead Tools.

For the general definitions of what characterizes an Arrowhead tool, refer to the document **O1** in the same delivery as this document (**D4.1**). We just recall here that interoperability is central to the concept of a *toolchain.*

A toolchain is a configurable, interconnected entity consisting of tools. The concept of a toolchain is versatile, allowing for being looked at from various perspectives, ranging from engineering processes to *toolchain interfaces*, the latter being the subject material of Task 4.2 and thus, this document.

As an example of a possible (realistic) toolchain representation, consider the following SysML-based model, an excerpt from the *Arrowhead Design Suite* toolchain:



Here, in Task 4.2 and the present document, we focus on the <<dataStructure>>box called AHX in the middle. This is an example of a ***toolchain interface***.

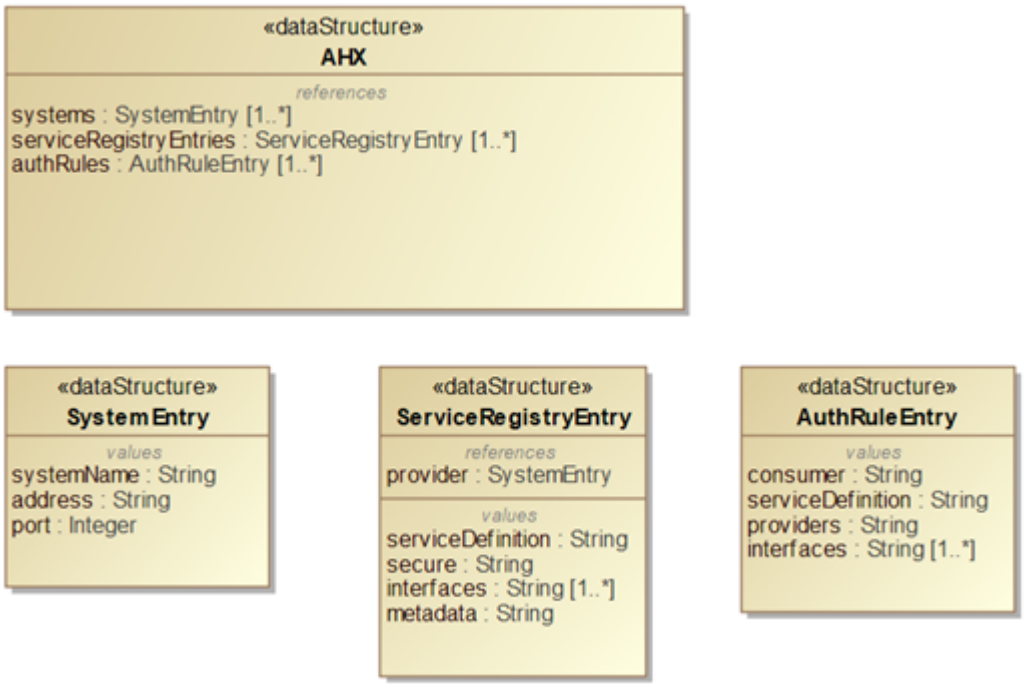
In a broader context, the definition of semantics as a toolchain interface is deliberately loose and can be summarized as follows:

A *semantics* is a collection of notions for describing an interface between different tools, engineering phases or other concepts, optionally along with (a set of) practical scenarios where those notions are applied.

- A semantics can be *formal,* being a mathematically rigorous data structure description as typical for, e.g., verification scenarios, *informal,* being a natural-language explanation of the involved notions, or it might involve both of the above flavors.
- A semantics can be *static,* describing or referring to snapshots of an SoS without addressing change or evolution, *dynamic,* describing or referring to the evolution or environmental interaction of systems or SoSs, or might comprise both.

It is crucial to add that in any of the above cases, a semantics contains a *data model* or *data structure* (we use those words interchangeably in this context).

In the above example, the AHX data model is provided in the same language, SysML:



In the following, intended to serve as an intuitive guideline for interoperability-focused Arrowhead development and engineering activities, we describe the major categories in which toolchain interfaces might play a role. *Note that this is different from interaction in the strict sense of Arrowhead systems:* here, the tools involved might be on different abstraction levels, might provide design-time modeling, validation or verification support. Thus, interoperability is addressed here from a purpose-oriented perspective, as it is reflected in the broad toolchain interpretation of AHT.

- Design and Modeling:

Here, the purpose of semantics is typically to define a data format, serving as an abstraction or mapping models for relating design-time artifacts to their (actual or future) run-time counterparts, possibly not in a 1-to-1 fashion, but rather in the form of a custom-tailored view on the system (or SoS). A typical example of a design-time artifact could be a diagram; in turn,

a formal semantics of a diagram might be a meta-model consisting of the concepts appearing on the diagram, while an informal semantics might simply state the meaning of concepts in natural language.

It is only the presence of semantics which makes design and modeling tasks interoperable: otherwise, they solely serve representational (visualization) purposes. The (mathematical or language) structure of that semantics might take different shapes, but the overall idea is always to integrate design artifacts into the engineering process and potentially even run-time engineering phases.

- Data Interpretation:

Such an interface allows for creating different views, understandable by other tools or human users, over any collection of system or SoS data. E.g., a query language with a well-defined semantics can be used for extracting features or patterns and to perform structural validation; a reporting and visualization interface might abstract away from existing data for representational purposes; while for a verification approach, we might want to convert the data into a formal structure (e.g., transition systems for model checking) allowing for the inference of properties (reachability, safeness, liveness, etc.)

- Data Storage:

Such an interface summarizes, condensates or extracts data from other system(s) in order to persist it in some other system or tool, for a well-defined and established share access by other systems and tools. This principle can be manifested not only by different database paradigms and their underlying semantics, but also by additional concerns such as versioning (i.e., providing a means for tracking the history of system states) and branching (i.e., dividing the data into different domains or scopes of interest).

3. The Arrowhead Tools Interface Catalogue: An Initial Assessment

3.1 Systems Modeling

IncQuery Labs (IQL)

Part of the *Arrowhead Design Suite* toolchain

The core of the systems engineering aspect of the AHT semantics landscape revolves around the SysML language and its extensions for diverse SoS engineering tasks. We frequently elaborate on these elsewhere, just as in the above example, as well as various research papers and tool demonstrations (see also **O3** in the present delivery).

Here, we just quickly recapitulate the main interfaces utilized by the SysML-based *Arrowhead Design Suite:*

- AHX is a custom data structure for propagating local cloud designs towards the Management Tool, the central operation interface of the Arrowhead Framework.

- ADX (*Arrowhead Deployment eXchange*) is a SysML-compliant adaptation of the Eclipse Vorto modeling language, which we utilize for integrating device and deployment data with our functional local cloud plans.
- ACX (*Arrowhead Choreography eXchange*) is a simple exchange format for communicating the choreographies (e.g., process and event models) created in external tools to the Management Tool for actual execution.

TU Eindhoven (TUE)

Use-Case: Efficient engineering processes for diagnostic imaging (UC2)

Part of the Model ecosystem toolchain

Model-based systems design involves various modeling languages and activities across different product versions and variants, and additionally throughout the entire product lifecycle. In order to support the design, management and evolution of model-based systems, we integrate the following items into the framework:
- Model repository: a model repository allows controlled CRUD operations on modeling artifacts, with additional features such as versioning and provenance.
- Model analytics functionality: services for performing complicated analysis on modeling artifacts, such as detecting duplication and other problems, monitoring the size and evolution of models.
- Model management functionality: a dashboard-like system that runs model analytics services on the model repository for the purpose of monitoring the model-based system, its quality and evolution.
- Model consistency: in typical model-based systems engineering, multiple models (e.g., covering different parts of a system, or coming from different domains) are used. They need to stay consistent when changes are made to one or more of them. As a first step, impact analysis and flagging of potentially inconsistent models and model parts is needed; ideally, automatic co-evolution is possible.

This toolchain is a generic model storage and management setup, with the following tools and interfaces:

Model repository <-> model management and analysis: Internal exchange via binary files

There should be an interchange of model files (single model files or multiple files as a zip file) between the model management and analytics system, and the model repository system. Model files may include formats such as SysML, UML, and BPMN.

Model management and analysis -> Dashboard: Custom data object

There should be a flow of model analytics data from the model management and analytics system to the dashboard system. A custom basic data object is designed for this purpose. It is expected that the object structure will get more complex to cover more advanced analytics metrics.

Expleo, BME (FTSRG) and TU Brno (BUT)

Use-Case: Automated formal verification (UC1)

There is no fixed Tool Chain for UC1 yet, nonetheless, we are working on the interoperability of the possible tools involved.

Considered tools and interfaces:

Expleo considers the usage of their tool Modica, with a custom XML as input.
Modica uses usage models to generate (executable) test cases. These usage models are in an XML-format.

BME FTSRG pursues a modelling and analysis (sub-)toolchain. We receive engineering models in some high-level languages and formats and we do formal verification, extra-functional analysis and test generation based on the SysML, EMF and XML standards.

BUT is considering OSLC-compliant representations (in XML and RDF format, conforming to the OSLC Automation Specification Version 2.0.

One of the main goals of the use case is to automate verification tasks. As the tasks can be realized by different tools (with different interfaces), OSLC unifies all of these interfaces into one OSLC-compliant interface thought which any of these tool can be invoked in a standardized way. The interface is web-based and provides a REST API. Therefore, even tools that can only be invoked locally, can now be invoked remotely by any REST client. This enables the tools to be offloaded to designated servers instead of requiring them to be installed locally. Moreover, the interface supports data persistence, meaning that all OSLC Automation Specification requests and responses processes by the interface are stored in a SPARQL database and are readily available to any OSLC-compliant client at any time (any phase of the development) in a standardized format.

3.2 Ontology Modeling

Mondragon University (MGEP)

A General-Purpose Ontology Design Methodology

For a large number of details on this important, general-purpose contribution, refer to the accompanying document **O5** and its **O5_annex**.

TU Kaiserslautern (TUK)

Use-case: Smart diagnosis

The aim of this semantic activity is to define a data model for digital twins with the capability for runtime verification.

The data model strives to join data from both from run-time and development-time with
a) Numerical analysis techniques, such as verification, optimization. This data includes:

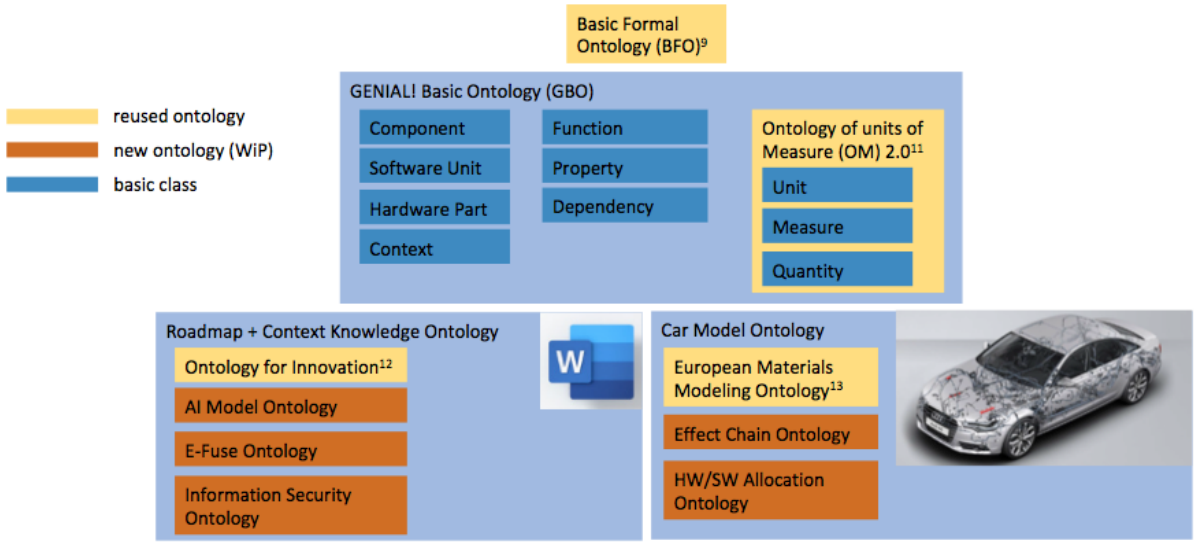
- Reachability and sensitivity data of selected quantities that act as a basis for a sound and system-wide verification and monitoring approach as described in (*Zivkovic, 2019)*.
- Samples and data obtained from operation.

b) Semantic technologies that semantify and disambiguate data throughout the entire product life cycle and across the value chain, in particular, of the automotive industry (tier-2, tier-1, OEM), assigning the meaning to the data passed between subsequent subsystems.
The link between the numerical techniques and the semantic technologies is created by

- Hierarchical decomposition of the system in parts and functions, that defines the context of the data, and the associated
- Properties (and constraints thereof) of parts and functions, and the
- Dependencies of parts and functions from other properties.

Properties, constraints thereof, and dependencies permit us to, during runtime, check the consistency of the samples obtained from operation with the models from development (represented by reachability analysis results). This is permitted by the dependencies that are just ASCII representations of constraints and equations that must hold universally.
In particular, in Arrowhead Tools, properties are checked by runtime-verification against streams of samples from a deployed plant.
Focus of semantic interoperability is on the extension of the ontologies of the GENIAL! Project, that is a German BMBF-Project, into the Development-Operation continuum. However, this ontology is just defining a top-level ontology into which other ontologies can easily be integrated. The focus on the use case smart testing is to extend the symbolic model checking approach from development time to run-time while providing a sound, scalable , and effective data-model across the product lifecycle. The GENIAL! Ontology that we use is based resp. is compatible with:

- The Basic Formal Ontology (http://purl.obolibrary.org/obo/bfo/2.0/) gives the highest-level structure.
- The GENIAL! Basic Ontology defines the means to represent configurations, specializations, and hierarchical decomposition of a system, its components, parts, dependencies, properties and functions. It complies to the ISO26262 standard. These terms are well defined and broadly used, e.g. in safety. It for example emphasizes a clear has-parts hierarchy for means of definition and correctness.
- The module suite is structured in layers and components, so that it is easily extendable and modularizable. It is roughly divided in roadmap/context knowledge, that entail domains such as innovation, artificial intelligence, e-mobility, demographic development, legislation and more. Those parts are part of roadmaps that lead key developments for whole industries and are used for e.g. planning. The other division is the car model, which comprises all that is related to the car and its hardware, like mechanics, hardware/software, sensors, distributed in-vehicle networking and more. The following figure gives an outline:



For Arrowhead Tools, we added means to import and manage input streams to the GENIAL! graph database, and to link the input streams to the GENIAL! ontology. Furthermore, we check compatibility of the sensor data (unit, dimension) via units of measure ontology.

Jotne and HIOF

Use-case: Offshore Crane (the Norwegian use-case)

Part of the ISO10303 toolchain

Here, the comprehensive and established industrial standard ISO10303 is used via various data formats defined there, to form a comprehensive toolchain for operating large-scale Arrowhead IoT installments.

In particular, ISO 10303-239, ISO 10303-242, ISO 10303-209 will be used for Digital Twins deployed for CAD, CAE, PLM and IoT connections using the Arrowhead Framework.

BnearIT

Use-case: Smart Kitting to Manage High Diversity

Still under definition.

BEIA

Part of the MQTT Sensor Management Toolchain

We use an acquisition system that includes Grafana for display results, INFLUX DB(SQL database), Mosquitto MQTT broker and sensor codes written in Python, C or C++, JSON. These tools work together when receiving sensor data.

Data between Grafana server and sensor devices is sent through MQTT.

MQTT answers issues related to IoT requirements like constrained devices, low-bandwidth and high-latency connections. It is based on a publish/subscribe communication but it does not have an authentication mechanism of its own. Mosquitto Go Auth is an authentication and authorization plugin for the Mosquitto MQTT broker, launched in 2020. A device sends messages through the MQTT protocol by publishing it to an MQTT Broker.

Infineon (IFAG)

Use-Case: Quick and reliable decision making in the semiconductor industry (UC5)

Part of the *Digital Reference* Toolchain

We are working on the tools Digital Reference and the UC5-specific ontology that form a toolchain with the other tools within our UC5, namely WHF and TePex.

The interface specifications are as follows:

Interface between ontologies: OWL-custom;
Interfaces of ontologies to WHF/TePex: tbd, most probably XML-custom.

Interface between ontologies: transfer conceptual information between ontologies (+ mapping, matching, merging) in machine interpretable and inferencable manner.
Interfaces of ontologies to WHF/TePex: still to be defined, transfer data points for instantiation of ontologies.

Eurotech

Use-Case: UC8.3 and 8.4

Part of the toolchain for designing, developing and operating the IoT solutions used in UC8.3 and UC8.4

The interfaces between tools don't use files (but data streams), except for tools configuration that is backup-restored using XML/Json files. We are planning a new tool to automate documentation editing and it will probably use html and XML files. The tools that we will use to set up the integration platform (Eclipse Kura and Kapua) in phases 4, 5 and 6 will exchange information using MQTT and AWS/Azure specific protocols. These tools will exchange mainly telemetry and command/control messages. For the rest of the phases of the EP, the toolchain is a standard Java development toolchain that is already integrated in the Eclipse IDE. Regarding the documentation automation, we currently adopt ReadMe solution in our documentation process. These solutions have a low degree of automation and our goal is to tackle this gap and enable ReadMe to keep the documentation up-to-date by adding the description of issues found in the source code. The Documentation Editing Tool is intended for this purpose and will be a tool natively integrated with the Arrowhead Framework. It will automate the connection between phase 3 and 8 of the engineering process, providing automatic functionalities for documentation editing.

3.3 Translation

Technical University of Lulea (LTU)
Machine learning based semantics/message translator
Highly dynamic and heterogeneous systems of systems create a heterogeneous semantic environment where a variety of data formats and models, legacy systems and semantic spaces

defined by different ontologies, engineers, and standards. Ontologies at the system level details the contexts, models, and interactions, which creates a complex semantic space that is expected to be non-static due to reconfigurations, updates, technology migration, maintenance etc [INDIN18].
The machine learning based semantics translator is a concept under development that aims to use machine learning methods to translate messages from one system to another, aiming to reduce the engineering effort required to create and maintain scalable interoperability in the Arrowhead framework. The approach is to optimize a translator using message data, metadata, and engineering utility to enable dynamic and operational interoperability and is outlined in [INDIN19]. The approach is partially implemented and evaluated in [INES20], achieving a translation accuracy of 75% using an encoder-decoder type unsupervised deep learning model.
The concept is under further development to implement and test the additional aspects outlined in [INDIN19], with particular focus on further incorporation of metadata and more sure-friendly utility and policy descriptions.
References
[INDIN18] J Nilsson and F Sandin, *Semantic Interoperability in Industry 4.0: Survey of Recent Developments and Outlook,* 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), pp. 127-132.
[INDIN19] J Nilsson, F Sandin, J Delsing, *Interoperability and machine-to-machine translation model with mappings to machine learning tasks*, 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), pp. 284-289.
[INES20] J Nilsson, J Delsing and F Sandin, *Autoencoder Alignment Approach to Run-Time Interoperability for System of Systems Engineering*, 2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES), pp. 139-144.

4. Revision history

4.1 Contributing and reviewing partners

Contributions	Reviews	Participants	Representing partner
X		Géza Kulcsár	IQL
	X	Federico Montori	IUNET
	X	Marek Tatara	DAC
	X	Andor Péter Umann	IQL
	X	Szabolcs Fábián	IQL
x		Christoph Grimm	TUK
x		Mahdi Saeedi Nikoo	TUE
x		Barbara Jung	EXPLEO
x		András Vörös	BME

x		Jan Fiedor	BUT
x		Kjell Bengtsson	JOTNE
x		An Lam	HIOF
x		George Suciu	BEIA
x		Patrick Moder	IFAG
x		Paolo Azzoni	EUROTECH
x		Jacob Nilsson	LTU
x		Fredrik Sandin	LTU
	x	Jurasky Wiking	IFAG
	x	Laurentiu Barna	Wapice

4.2 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2020-10-09	0.1	First draft	Géza Kulcsár
2	2019-12-10	1.0	Final Sanity Check	Federico Montori, Marek Tatara
3	2020-10-29	1.1	Year 2 integration	Géza Kulcsár
4	2020.11.02	2.0	Final Sanity Check	Federico Montori, Marek Tatara
5	2020-11-25	2.0	Internal Review	Patrick Moder
6	2020-12-02	2.0	Internal Review	Laurentiu Barna
7	2020-12-03	2.1	Addressing comments of the internal reviewers	Marek Tatara

4.3 Quality assurance

No	Date	Version	Approved by
1	2020-12-03	2.1	Jerker Delsing