ARROWHEAD TOOLS

Document title: Definitions for the Arrowhead Tools design principles

Version	Status	Date
2.1	final	2020-12-03

Author
Marek Tatara, Federico Montori

Contact
marek.tatara@dac.digital,
federico.montori2@unibo.it

Definitions for the Arrowhead Tools design principles

Authored by Marek Tatara, Federico Montori and Géza Kulcsar. Definitions were contributed and agreed by the following partners:

AITIA:	Svetlin Tanyi szvetlin@aitia.ai
BnearIT:	Hans Forsberg Hans.Forsberg@bnearit.se
BME:	Pal Varga pvarga@tmit.bme.hu
DAC:	Mateusz Bonecki Mateusz.bonecki@dac.digital Anna Kwaśnik anna.kwasnik@dac.digital Krzysztof Radecki krzysztof.radecki@dac.digital Marek Tatara marek.tatara@dac.digital
EUROTECH:	Paolo Azzoni paolo.azzoni@eurotech.com
IUNET:	Federico Montori federico.montori2@unibo.it
IQL:	Géza Kulcsár geza.kulcsar@incquerylabs.com
LTU:	Ulf Bodin ulf.bodin@ltu.se Cristina Paniagua cristina.paniagua@ltu.se
POLITO:	Gianvito Urgese gianvito.urgese@polito.it

Abstract

This document contains the most essential definitions for the Arrowhead Tools design principles. The definitions of a tool, a toolchain and an engineering process unit are provided.

Table of contents

1. Initial definitions

This document contains a concise summary of the initial definitions for the Arrowhead Tools design principles, which address Arrowhead tools, toolchains, and their relation to the engineering process. Those design principles involve formal definitions as well as informal guidelines. The Arrowhead Tools reference architecture abstractly realizes these principles and it is supported by example tools and toolchain implementations. The definition has been expanded significantly from D4.1 with an exhaustive explanation that has been necessary to define tools in contexts that are different in scope.

1.1 Tools

In the Arrowhead ecosystem, there are functional systems that contribute to the main target of SoS and are always a part of a local cloud, and there are non-functional systems which support reaching this target and are not necessarily a part of any local cloud, these are commonly referred to as **Tools**. But what *is* a tool in this context? The key observation here is that an Arrowhead Tool is not necessarily something *technically attached* to actual deployments of the Arrowhead Framework. In contrast, in the context of a digital revolution and after decades of advancements in industrial computing, we can be sure that an Arrowhead Tool *is* or *has* a piece of software. It is instructive to quote the officially accepted definition of an Arrowhead Tool as proposed by the present authors, reflecting and elaborating on the above observations. Afterward, we summarize the key points and provide some examples. It is important to stress that henceforth we will use the word ``tool'' to identify ``Arrowhead Tools'' just as they are defined below, not by its English dictionary definition.

A tool is a software or a hardware (with adequate software on-board) entity/artifact that supports CP SoS and SoS engineering activities. The phases of the engineering process in principle can be managed without tools (*i.e.*, with a strong human component), but probably will use some.

1. **It could be a design-time or a run-time tool, depending on its place within the process.**
2. **It can be service provider, consumer, both or none; in short, it is compliant with the Eclipse Arrowhead (the first three cases) or not. We stress that it is not necessary for a tool to support Arrowhead by design (natively) to implement any services in the strict Eclipse Arrowhead sense; such a tool, which already exists and should be adapted to work with Arrowhead, is called *Arrowhead-enabled.* In contrast, a Framework-compliant tool (with embedded compliancy) is called an *Arrowhead Native Tool.***
3. **The output of a tool should be processable by other tools adopted in the other phases of the engineering process.**
4. **The output of a tool should be processable by other toolchains.**
5. **A tool is an atomic part of a toolchain, and cannot be broken down into subtools that can work autonomously.**

As the line between a tool and an application system, and also between tools and non-tools, might be unclear in some cases, we proceed with a number of examples in different categories.

Non-Tools
Let us first imagine an industrial scenario in which an automated machine prints silicon boards. We could say that silicon boards could be also hand-made with a significantly increased effort and, therefore, the machine is a tool that aids production. In fact, such a machine supports the

Operation & Management engineering phase; however, it seems so indispensable that we consider it as a *baseline*; theoretically, manual work is imaginable, but in an industry scenario not anymore.
A similar consideration applies for, *e.g.*, compilers, which could be seen as tools because they translate human-readable code into machine-readable code. However, using a compiler for software engineering is obviously a baseline now. Summarizing, a tool is not only a replacement for manual work in an industrial engineering process, but it also improves an already established industrial baseline (for now, our judgment of baselines is based on common sense and does not elaborate further on a proper definition).

General-purpose industrial tools.
Here, we extend the examples above to become actual tools in our conceptual framework. Frist, getting back to the silicon board printer, we could imagine to plug it into a software system that is able to determine, based on historical sales data, what is the optimal amount of boards to produce every day without wasting resources, how much time the machine should be up and running in order to achieve this number and, based on the curve of the daily price of electricity, when it is optimal to turn it on in order to have the expenses reduced. Now, this software system significantly improves the industrial baseline by cutting its costs without altering its main purpose; therefore it is a tool, specifically devoted to the Operation & Management phase.
For a software-oriented example, let us revisit compilers: Integrated Development Environments (IDE) used for programming usually have an integrated pre-compiler that suggests potential bugs at design time: those are tools that help speeding up the Procurement & Engineering phase of producing a software artifact.

Multi-purpose (abstract) Arrowhead tools.
To illustrate the above tool concept in its originating Arrowhead context, we first provide some common examples of tools encompassing multiple EPPs, planned to be fully implemented in the future, mature state of the Arrowhead Tools platform. The list is far from being exhaustive but aims at showcasing the diverse scope range of thinkable Arrowhead Tools from early software validation to actual production.

- **Test Tool** A general software solution to test basic validity requirements for any Arrowhead local cloud before its deployment; *e.g.*, every local cloud should have a running Service Registry, systems which are known to be communicating from the beginning should have data interfaces with compatible encodings, etc. This belongs to Procurement & Engineering as well as Deployment & Commissioning phases.
- **Deployment Tool** Software running on a central computer, overseeing the deployment procedure of an Arrowhead SoS design, *i.e.* installing software systems to their dedicated hardware and establishing communication (Deployment \& Commissioning phase).
- **Local Cloud On-boarding Tool** A piece of software that can be executed on a device with basic wireless and Arrowhead capacities. It can be used for detecting local clouds in the environment that the device is entitled to join, and possibly even manage basic negotiations.
- **Component Presence Detector** A tool that takes as input camera stream, and produces a binary output denoting whether a physical component (*e.g.*, a piece to assembly) is present at the desired position with the correct orientation. This is an example of the Operation & Management tool (that could be either Arrowhead-enabled or not, depending on its implementation).

- **Arrowhead-enabled Reconfigurable Sensor Data Provider** This is a run-time tool consisting of a hardware component (an STMicroelectronics microcontroller) with bare-metal firmware enabling communication with the Arrowhead Framework over the HTTP protocol, built on LWIP's RAW TCP/IP API stack, along with the implemented support for the developed on-boarding process of new measurement nodes. This is, also being an example of a hardware-software combination, a run-time tool, an (Arrowhead-enabled) service provider, and cannot be subdivided into tools (as expected). The input of the tool is the information about the interface, on which the sensed data should be read. The output of the tool is the data measured on the configured interface. The tool falls under Deployment & Commissioning and Operation & Management phases.

1.2 Toolchains

The characteristics of an Arrowhead tool as detailed in the previous section (interoperability and atomicity in particular) make this notion appropriate as a basic constituent of well-founded toolchain descriptions: (i) the requirements on tool interoperability ensure their integrability into tool sequences and (ii) their atomicity facilitates the clarity of the resulting toolchain architecture.

Just as we did for Arrowhead tools, let us first revisit the accepted definition of an Arrowhead toolchain:

A **toolchain** is a collection of tools and of the definitions of the corresponding interfaces potentially organized in chain-based or parallel structures. Tools in a toolchain can be substituted/replaced with other tools with the same input/output interfaces.

2. It can be design-time, run-time or both.
3. It aims for a certain level of automation in information processing/transfer throughout the engineering process.
4. It can allow iterative use of its parts (tools and toolchains).
5. It can cover only some (not necessarily consecutive) parts of the engineering process, or the whole product lifecycle (typical for general infrastructural tools), even iteratively until the end-of-life phase.

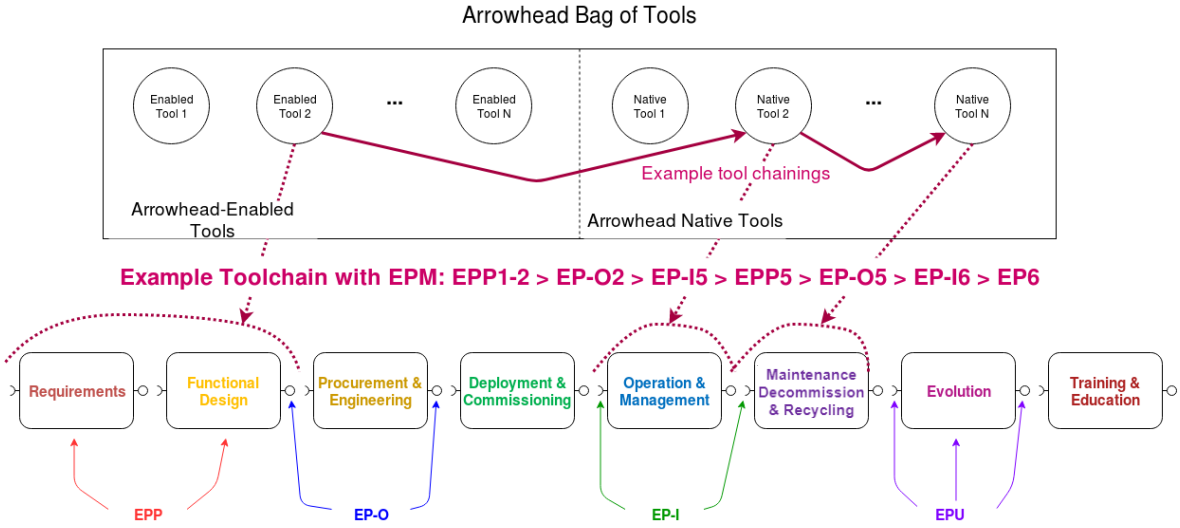
We also recall that an **Engineering Process Unit** (EPU) is either an **Engineering Process Phase** (EPP), based on the AHT-EP (Arrowhead Tools Engineering Process) or an **Engineering Process Interface**, which can be in turn either an input interface (EP-I) or an Output interface (EP-O) between them. An **Engineering Process Mapping** (EPM) relates a tool to one or more EPU it covers.

Thus, the toolchain concept proposed here naturally integrates with AHT-EP and its flexible phase ordering principle. In turn, specifying a toolchain with an engineering process mapping (EPM) combines this flexibility with an adequate amount of requirements to fulfill: whenever two EPPs are attached to each other via their corresponding interfaces, we pose a requirement on the toolchain to realize data exchange in the corresponding direction between those (or generally, some of those) tools addressing the respective EPPs.

2. Tools and Engineering Process Mapping

The following figure shows an abstract overview of the Arrowhead Tools landscape: The upper box represents the so-called “Bag” (*i.e.* loose collection) of Arrowhead Tools, where the division into Arrowhead-enabled and Native tools is also indicated. The bottom row of blocks with connectors is a concise representation of the engineering process (see details below the figure). The central part, in purple, is an abstract example of what an EPM for a fictive toolchain might be, and how this could be graphically represented.

The solid arrows between three (fictive, non-concrete) tools represent their chaining, while the single tools are also mapped via dashed EPM arrows to the engineering process phases and their input/outputs.



The upper part of the figure above shows an abstract representation of the overall model resulting from the combination of AHT-EP with our toolchain concept. Circles in the upper row represent an initially unordered collection of existing or even potential (Arrowhead) tools, divided into Arrowhead-enabled and Arrowhead Native (*i.e.*, Framework-capable) tools, emphasizing that those categories are easily combined within a single toolchain. The actual toolchain is then specified by selecting its constituting tools and defining an EPM such that the data exchange links (*i.e.*, the chainings) between the tools are reflected in an EPP configuration via interfaces. Moreover, a correct toolchain specification should be synchronized on both sides: while every contained EPP and interface has to be covered by at least one tool, also, every tool chaining has to correspond to at least one EPP interface pairing (*i.e.*, an output interface connected to an input interface). The figure also hints at a noticeable corner case: sometimes, a single tool might cover multiple EPPs and also their connecting interface.

Thus, we have summarized the principles of a comprehensive high-level engineering workflow model for Arrowhead Tools. In order to align it with AHT-EP, we propose the name Arrowhead Toolchain Model, or AHT-TC for short.

While AHT-EP could formally be considered as a part of AHT-TC, we stress that their actual usage highlights their different origins and both models, while sharing a common baseline, are utilized with different focal points and in slightly different contexts. The usage of AHT-EP is elaborated in detail in D2.1.

For the sake of intuitiveness and logical consequentiality, we represent graphically the EPPs in a sequence. However, this does not force the use case designer to follow such phases in this succession; rather, a phase could be executed iteratively, phases could be skipped and connected to “previous” ones and so on. Note that EP-I and EP-O outline inputs and outputs that are meant to be produced/consumed by other tools in the toolchain. A generic input or output of the tool is therefore not necessarily represented this way.

For a better clarity, we give EPPs alternative names as follows:

- EPP1: Requirements
- EPP2: Functional Design
- EPP3: Procurement & Engineering
- EPP4: Deployment & Commissioning
- EPP5: Operation & Management
- EPP6: Maintenance, Decommissioning & Recycling
- EPP7: Evolution
- EPP8: Training & Education

Similarly, EP-I and EP-O are numbered as follows:

- EP-I1: Input for Requirements
- EP-I2: Input for Functional Design
- EP-I3: Input for Procurement & Engineering
- EP-I4: Input for Deployment & Commissioning
- EP-I5: Input for Operation & Management
- EP-I6: Input for Maintenance, Decommissioning & Recycling
- EP-I7: Input for Evolution
- EP-I8: Input for Training & Education

- EP-O1: Output of Requirements
- EP-O2: Output of Functional Design
- EP-O3: Output of Procurement & Engineering
- EP-O4: Output of Deployment & Commissioning
- EP-O5: Output of Operation & Management
- EP-O6: Output of Maintenance, Decommissioning & Recycling
- EP-O7: Output of Evolution
- EP-O8: Output of Training & Education

This means that, for instance, EP-O4 does not necessarily feed only EP-I5, but it can serve as an input of any other phase.

To conclude, in order to clarify better, there may be different conceptual kinds of tools and toolchain: existing tools in the toolchains that are currently used to develop the use cases, potentially covering all the engineering phases (*e.g.,* gcc, Eclipse IDE, Synopsys SW, CAD, *e.g,* for operation Eclipse Kura, NILM algorithms, …); tools that we will develop in the project that could be Arrowhead Framework compliant or not, because of lack of functionalities. Furthermore, there are some artifacts produced by the toolchain in the engineering process (typically in EPP1-4) that are not tools in such phases, but become tools in the following phases (e.g. an IoT framework used for manage a fleet of devices, is a software produced by EPP1-4 and in EPP1-4 it is not a tool, but in EPP5-6 it becomes a tool).

3. List of abbreviations

Abbreviation	Meaning
SoS	System of Systems
CP SoS	Cyber-Physical System of Systems
EPU	Engineering Process Unit
EPP	Engineering Process Phase
EAEM	Extended Automation Engineering Model
EP-I	Engineering Process Input Interface
EP-O	Engineering Process Output Interface
EPM	Engineering Process Mapping

4. Revision history

4.1 Contributing and reviewing partners

Contributions	Reviews	Participants	Representing partner
X	X	Marek Tatara	DAC
X	X	Federico Montori	IUNET
X	X	Géza Kulcsár	IQL
	X	Svetlin Tanyi	AITIA
	X	Hans Forsberg	BnearIT
	X	Pal Varga	BME
	X	Mateusz Bonecki	DAC
	X	Anna Kwaśnik	DAC
	X	Krzysztof Radecki	DAC
	X	Ulf Bodin	LTU
	X	Cristina Paniagua	LTU
X	X	Paolo Azzoni	Eurotech
X	X	Gianvito Urgese	Polito
	X	Partick Moder	IFAG

	X	Laurentiu Barna	Wapice

4.2 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2019-09-24	0.1	First Draft	Marek Tatara
2	2019-09-30	0.2	Interface I/O Mapping	Federico Montori
3	2019-10-04	0.3	Refinement of the definitions	Marek Tatara, Paolo Azzoni
4	2019-10-16	0.4	Refinement of the definitions, definition of the mapping	Géza Kulcsár
5	2019-10-25	1.0	Final Version	Marek Tatara, Federico Montori
6	2019-12-11	1.1	Final Version for D4.1 with conclusions	Federico Montori, Marek Tatara
7	2020-09-24	1.2	Second Draft, added all the new content for D4.2, to be revised	Federico Montori
8	2020-11-02	2.0	Final Version for second year	Federico Montori, Marek Tatara
9	2020-11-25	2.0	Internal Review	Patrick Moder
10	2020-12-02	2.0	Internal Review	Laurentiu Barna
11	2020-12-03	2.1	Addressing comments of the internal reviewers	Federico Montori, Marek Tatara

4.3 Quality assurance

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