Eclipse Arrowhead

Concepts Reference

Framework Description (FD)

Abstract

This document provides authoritative definitions for the most fundamental concepts of relevance to *Eclipse Arrowhead*, a framework designed to facilitate the effective creation of highly dynamic automation systems. It is meant to serve as foundation for other documents with relevance to the framework, providing a precise vocabulary untied to any specific practices or technologies. While presented in the form of a model, the document does not in and of itself build upon or endorse any particular modeling language.

Contents

1 Introduction

We expect the automation systems of today to keep becoming more and more computerized, digitized and interconnected. By this we mean that more aspects of and surrounding automation machines will be handled by computers, more information will be made available to those computers and, finally, comparatively more such computers will be given the opportunity to collect, communicate and act on that information. Manufacturing, transportation, energy distribution, medicine, recycling, as well as all other industrial sectors concerned with automation will be affected by this development. It will lead to increased automation efficiency and flexibility, as machines become able to perform more of the work traditionally assigned to humans. However, it will also lead to new magnitudes of complexity, not the least because of the renewed incentive to use more and more of these highly communicative machines.

The *Arrowhead framework* is designed to address this explosion of complexity. It provides a foundation for *service-oriented communication* [1] between automation systems and other computers, such that interoperability, security, safety, performance, and other major concerns can be addressed efficiently and effectively. It notably allows for system capabilities to be described, shared and exploited dynamically by communicating devices.

In this document, we, the Eclipse Arrowhead project, present an authoritative set of concept definitions, meant to serve as the fundamental language for describing Arrowhead-based system designs. It exist to help mitigate compatibility and consistency issues in software, tooling, models, documentation and all other things of relevance to the Arrowhead framework.

1.1 Primary Audiences

This document is being written and maintained for all who need precise and rigorous definitions of important Arrowhead concepts, which we understand to likely include the following groups:

- Advanced users of Arrowhead systems.
- Architects contributing to or extending the Arrowhead framework.
- Developers of Arrowhead systems, or of devices that are expected to host Arrowhead systems.
- Operators of Arrowhead systems.
- Researchers concerned with analyzing or refining the Arrowhead framework or Arrowhead systems.

1.2 Scope

This document is intended to clearly define all technical concepts of fundamental importance to the Arrowhead framework. It does not specify how Arrowhead-based automation systems ought to be designed. This makes its purpose analogous to that of a dictionary. Dictionaries define words. They may give examples of how certain words may be used, but they do not require that those words be used for any particular purposes. This document provides an Arrowhead vocabulary other documents or models may use to express software-centric automation system designs. It does not recommend any particular methodologies or technologies.

The concepts presented here are meant to be useful as a resource for advanced Arrowhead framework learners, as well as to serve as foundation for other documentation and modeling efforts. This document does *not* define an Arrowhead profile for SysML [2], or any other modeling language. For those interested in using this document for software-architectural purposes, a description of how it can be used as a metamodel in the context of an ISO/IEC/IEEE 42010 model kind is provided in Section 4.1.

1.3 Notational Conventions

This document adheres to the notational conventions presented in the below subsections.

1.3.1 Graph Diagrams

In a graph diagram, a box with a solid border and a name inside it denotes a named model entity, representing an artifact or stakeholder. Model entities can be associated with attributes by describing those attributes in text in relation to the diagrams in which they occur. A named arrow from a source box to a target box denotes the relationship implied by the name. Relationship names are defined either here, in the glossary of Section 5, or in relation to the figures they are used in. The following relationship names are defined here only:

1. *conforms to*, implying that the target *has* a set of constraints satisfied by the source;
2. *extends*, meaning that the source *conforms to* and inherits all relationships and attributes of the target;
3. *is*, meaning that the source *extends* the target and belongs to a set named after it;
4. *uses*, meaning that the source depends on the target to fulfill its purpose;
5. *has*, meaning that the target is *used by* and must cease to exist without the source; and

Quantifiers If an arrow has an associated positive integer or range, which we refer to as a *quantifier*, the relationship is to be considered as extending to the number of distinct entities indicated by that quantifier. No quantifier being associated with a certain relationship implies that it has a quantity of 1. A range is denoted by $x..y$, where x and y are integers and $0 \leq x < y$. If y is substituted by $*$, the range is to be understood to extend infinitely from x (e.g. "$1..*$").

Grouped Relationships To save space or improve clarity, arrows are sometimes grouped such that either their target or source ends are shared, as in Figure 1. If such a group of arrows has a relationship name closest to its shared part, it must be understood to apply to each arrow of the group, as if they were not grouped at all. Relationship quantifiers are always closest to the non-shared parts. Grouped arrows can always be replaced with non-grouped arrows without loss of information.

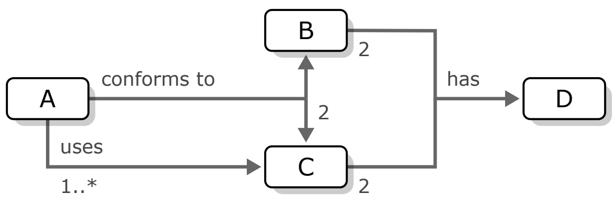

Figure 1: An example graph diagram. **A** *conforms to* 1 **B**, as no quantifier is associated with the arrow from **A** to **B** and 1 is the default quantity. **A** also *conforms to* 2 **C**s, as well as *uses* 1 or more **C**s. Both **B** and **C** *has* 2 **D**s.

1.3.2 References

Square brackets around integers (e.g. [3]) are references to the reference list in Section 6. The integer within the brackets of any given reference corresponds to the entry with the same integer in the reference list.

References within this document are hyperlinked, which means that those reading it electronically can click the references and immediately be taken to their targets. Special treatment is given to references targeting Section 5, the Glossary. These are displayed as regular text rendered with blue color.

1.3.3 Requirements

Use of the terms **must**, **must not**, **should**, **should not** and **may** are to be interpreted as follows when used in this document: **must** and **must not** denote absolute requirements and prohibitions, respectively; **should** and **should not** denote recommendations that should be deviated from only if special circumstances make it relevant; and, finally, **may** denotes something being truly optional.

1.4 Relationships to Other Documents

This document reuses or builds upon the concepts presented in the following works:

1. **IoT Automation: Arrowhead Framework** (IoTA:AF) [3], which significantly includes an overview of the *local automation cloud* concept in its second chapter, as well as the *Arrowhead framework architecture* in its third chapter. The book most significantly represents the state of the Arrowhead framework up until it was written. Even though the framework has evolved since then, it still represents the most comprehensive description of the framework. While the strictly architectural aspects of IoTA:AF are outside the scope of this document, the two mentioned chapters contain several definition with a high degree of relevance here.

2. **ISO/IEC/IEEE 42010 Systems and software engineering — Architecture description** (ISO42010) [4], which outlines a standardized approach to structuring architectural documents and models. The standard is adhered to in the sense that the definitions of this document are meant to be useful as a metamodel part of a so-called *model kind*, as defined by the standard. No claim of conformance to the standard is made for this document on its own. Please refer to Section 4.1 for more details.

3. **Reference Model for Service Oriented Architecture** (SOA-RM) [1], which provides a standardized definition of Service-Oriented Architecture (SOA). Communications between Arrowhead systems are expected to adhere to this paradigm, which is what makes the standard relevant here.

4. **Reference Architecture Model Industrie 4.0** (RAMI4.0) [5], which outlines an ontological and architectural description of *Industry 4.0*. The document may be seen as a predecessor to, or major influence on, the conceptual aspects of the Arrowhead framework. In particular, the document describes how to model and design communicating industrial systems such that key Industry 4.0 characteristics can be facilitated, such as high degrees of dynamicity and interoperability. However, as RAMI4.0 is a reference *architecture* rather than a reference *model*, we have only been concerned with what concepts it defines and what problems it frames. This delimitation excludes its “architectural layers”, “life-cycle & value-stream” phases and “hierarchical levels”, as well as the abstract design of its “asset administrative shell”. These excluded aspects are neither condemned nor endorsed by this document. They are simply outside its scope.

Only conformity with IoTA:AF and ISO42010 is observed strictly, which means that concept definitions presented here may diverge from those of the other two works. All significant terminology differences are noted in the glossary of Section 5, which provides a brief definition of each concept of relevance to this document.

1.5 Section Overview

The remaining sections of this document are organized as follows:

Section 1 This section.

Section 2 A brief and formal overview of Arrowhead, describing how its core concepts relate to each other. The section also serves to provide a workable summary of the framework and to prepare readers for better understanding Section 3.

Section 3 A formal description of the most significant concepts of Arrowhead. Each of its subsections is concerned with one primary concept, ranging from entities to systems-of-local-clouds.

Section 4 A list of requirements, meant to help determine if a document or model referring to the concepts of this document can be considered conformant. A special subsection on ISO/IEC/IEEE 42010 conformance is also provided.

Section 5 Lists all significant terms and abbreviations presented in this document in alphabetical order.

Section 6 Lists references to publications referred to in this document.

Section 7 Records the history of officially ratified changes made to this document.

2 Overview

The Arrowhead framework can be divided into a framework of ideas and a framework of software, as shown in Figure 2. The former division concerns the assumptions, concepts, values and practices that frame the problem domain of *coordinating dynamic automation systems*. The latter division concerns the software specifications and implementations meant to address that problem domain. In this section, we provide an overview of the primary *concepts* of the Arrowhead framework. While *assumptions* and *values* may be possible derive from this overview, no other framework aspects are considered here or in the rest of this document.

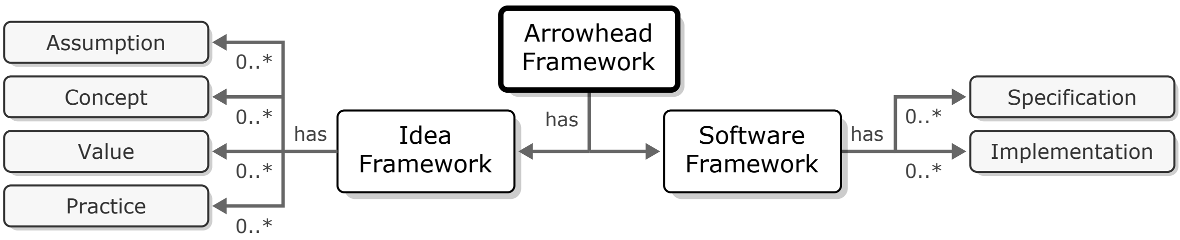

Figure 2: The two subframeworks of the Arrowhead framework, concerned with ideas and software.

2.1 Stakeholders and Artifacts

There are two kinds of members of the world of Arrowhead, (1) stakeholders and (2) artifacts, as depicted in Figure 3. The former denotes a person or organization engaged in an Arrowhead enterprise, while the latter is any thing or object, tangible or intangible, that could be relevant to consider as part of such an enterprise. Stakeholders own, supply, develop, operate, and use artifacts, among many other possible activities. It is their business needs and ambitions that govern what and how Arrowhead artifacts are employed.

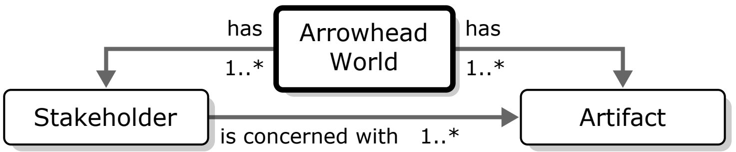

Figure 3: The two kinds of members of the Arrowhead world: stakeholders and artifacts.

2.2 Devices, Systems and Services

The most essential types of artifacts in the world of Arrowhead are (1) hardware devices, (2) software systems and (3) services, all shown in Figure 4. *Hardware devices*, or just *devices*, are physical machines, such as servers, robots or tools, able to maintain, or *host*, *software systems*. A software system, or just *system*, is a communicating software instance that provides *services*. Every service represents a set of tasks a system can perform for other systems. Those tasks are concretely represented by a set of operations that system can execute as requested by other systems. A service may be concerned with manufacturing, repairs, analysis, or any other physical or virtual activity. Each of its operations is dedicated to one aspect of its concern. A service providing control over a door could, for example, have one operation for checking if the door is open and another for opening and closing it. Service operations can be executed, or *consumed*, by other systems or persons.

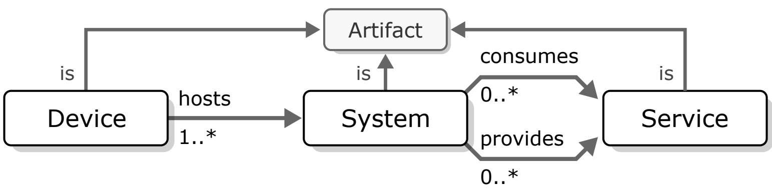

Figure 4: Hardware devices *host* software systems, which *consume* and/or *provide* services.

2.3 Service Provision and Consumption

Communication between systems is formulated in terms of the provision and consumption of services. Systems may *provide* services, which other systems can *consume* by sending messages, as depicted in Figure 5.

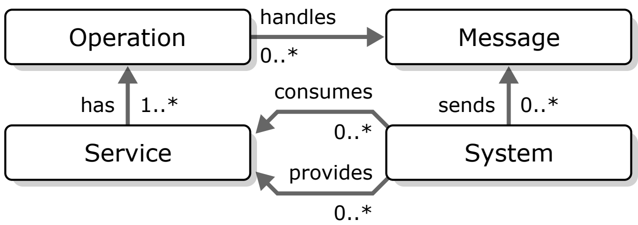

Figure 5: Systems consume services by sending messages to the providers of those services. Those providers then pass on the messages they receive to their service operations, which interpret and handle them.

When a providing system receives a message from a consuming system, it passes it on to the service operation specified in that message, as described in Sections 3.5 and 3.10. The operation receiving the message will then handle it by performing whatever action it describes, given that the message is valid and permitted. This handling may entail sending additional messages to other systems, starting or stopping various kinds of automation routines, reading from sensors, electronically signing contracts, sending notifications to an operator, among many other possible examples.

2.4 System Composition

When certain systems consume each other's services, they form a system-of-systems. Such a system-of-systems is able to perform activities none of its constituent subsystems could perform on its own. Two kinds of system-of-systems have particular significance in the context of the Arrowhead framework. These are (1) the local cloud, and (2) the system-of-local-clouds, both depicted in Figure 6.

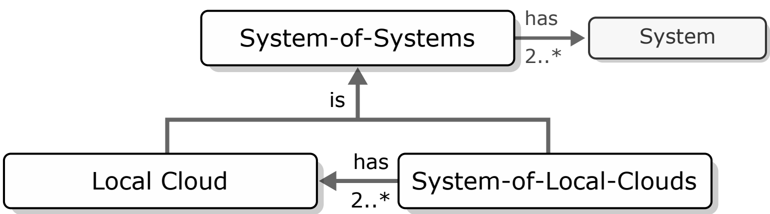

Figure 6: The two primary kinds of Arrowhead systems-of-systems: the local cloud and the system-of-local-clouds.

A *local cloud* is a set of systems engaged in some form of physical activity that makes those systems physically bound to a particular location. Local clouds come in many shapes and forms. They may be completely stationary, completely mobile, or consist of both stationary and mobile devices. Smelting stations, drone command centers, assembly lines, power distribution centers and satellite systems are a few examples of possible local clouds.

A *system-of-local-clouds* is a set of cooperating local clouds, each kept distinct from the other local clouds by some form of boundary. Boundaries may be organizational, physical, security-related, and so on. Every system-of-local-clouds contains at least one local cloud that depends on another local cloud to perform some activity of relevance. A systems-of-local-clouds could be formed by a set of weather stations, the robots of some collaborating parties at a mining site, the various departments of a manufacturing plant, the carriers of a supply chain, and so on.

3 Core Concepts

With the major themes of the Arrowhead framework now established, we proceed to outline its most significant concepts in detail. Each subsection of this section describes one of these concepts, which are as follows:

3.1	**Stakeholder**	A person or organization concerned with an entity or undertaking.
3.2	**Entity**	An artifact that can be distinguished from all other artifacts.
3.3	**Device**	A physical entity with the capability of hosting systems.
3.4	**System**	A software instance able to exercise the capabilities of its hosting device.
3.5	**Service**	A set of operations provided by a system for other systems to consume.
3.6	**System-of-Systems**	A set of systems that jointly facilitate new capabilities.
3.7	**Local Cloud**	A cloud with a local boundary and local resources.
3.8	**System-of-Local-Clouds**	A set of local clouds that jointly facilitate new capabilities.
3.9	**Network**	A set of devices with network interfaces that are able to communicate.
3.10	**Interface**	A boundary that can be crossed by the messages of certain protocols.
3.11	**Protocol**	A description of what messages can be sent between certain interfaces.
3.12	**Policy**	A set of constraints that must be satisfied for a message to be permitted.
3.13	**Profile**	A set of constraints added to a protocol.
3.14	**Encoding**	A data type used to structure and interpret certain data.

3.1 Stakeholder

A stakeholder is a person or organization with stake in an entity or undertaking with relevance to the Arrowhead framework, where *stake* is any form of engagement or commitment. Stake may be concretely expressed by a stakeholder being associated with one or more *roles*, as illustrated in Figure 7.

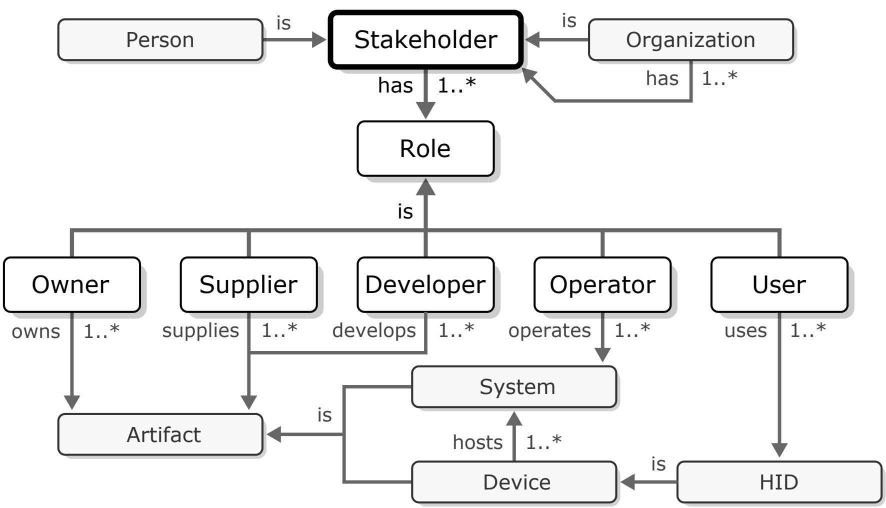

Figure 7: The stakeholder as either a person or organization, where each such stakeholder takes on one ore more distinct roles. The depicted roles are only possible examples. HID is an abbreviation for Human Interface Device.

The roles of a given stakeholder dictates what entities that person or organization will interact with, as well as the nature of those interactions. In Figure 7, (1) owner, (2) supplier, (3) developer, (4) operator and (5) user are named explicitly, but more roles are likely to be relevant, such as (6) acquirer and (7) maintainer, (8) builder, (9) researcher and (10) architect. The listed ten names should be used rather than any synonyms when referring to these particular roles. Please refer to the glossary for their definitions. If this document is read electronically, each role name can be clicked to be taken to its definition.

3.2 Entity

An entity is an artifact that it identifiable, which means that it can be distinguished from all other artifacts. We use the word *artifact* to refer to any object or thing, physical or intangible. As depicted in Figure 8, this means that an entity always has an identity.

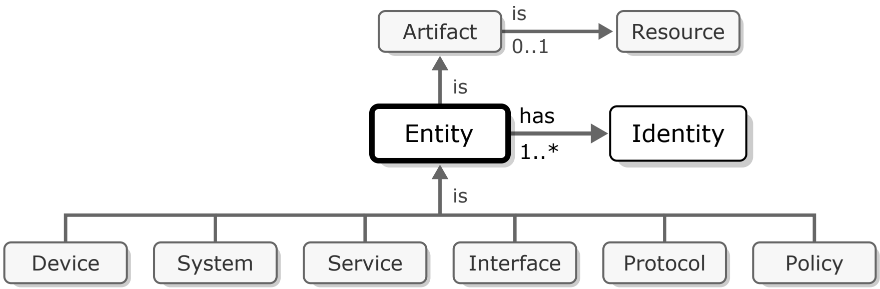

Figure 8: The entity as an artifact with an identity. An entity or artifact may or may not be considered to be a resource, in which case it is deemed to be valuable or useful from the perspective of a stakeholder. The array of artifacts with an *is*-relation to *Entity* is not complete. Other examples include local clouds, profiles and encodings.

Note that having an identity is not the same as being associated with an identifier, which is a name, number or other value referring to an entity. It is enough that any such identifier is possible to produce for an artifact to count as an entity. That being said, certain identification requirements, perhaps related to security, performance or discoverability, may make it impractical to treat any other artifacts as entities than those with identifiers.

3.3 Device

A device is a physical entity with certain automation and compute capabilities. Examples of capabilities include moving robotic arms, reading from sensors, running software and sending messages. Every device must be capable of *hosting* at least one system, which are communicating software instances. Devices consist of hardware components. While there are no limits to what such components can make up a device, each device must always have (1) memory, (2) compute and (3) network interfacing components, as shown in Figure 9.

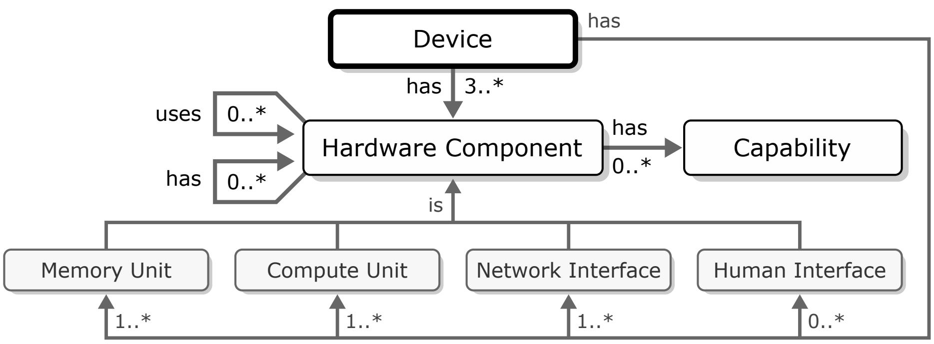

Figure 9: The device as a set of hardware components, each with automation or compute capabilities. Every device must be able to host software components using its compute and memory units, as well as communicate with other devices via its network interfaces. Devices with human interfaces are able to communicate directly with persons. Other examples of hardware components could be sensors, actuators, compute accelerators and batteries.

Every device must be able to host at least one system, or it is to be considered as being a hardware component. While it may seem unintuitive to consider certain machines as components, such as large pumping complexes or vehicles with only manual controls, the Arrowhead framework is meant to facilitate automation through the use of interconnected devices with compute capabilities. If a machine cannot run software, making it able to host systems, that capability must be added before it can play a meaningful role in an Arrowhead context. Consequently, machines without system hosting capabilities must either be considered as components or not be considered from the perspective of Arrowhead at all.

3.4 System

A system is an identifiable software instance that is hosted by a device. As shown in Figure 10, a system consists of software components. Just as hardware components, software components can have various types of automation or compute capabilities. Every system should have the capability of consuming services, providing services, or both. If a given system can do neither, it must be referred to as an *opaque system*.

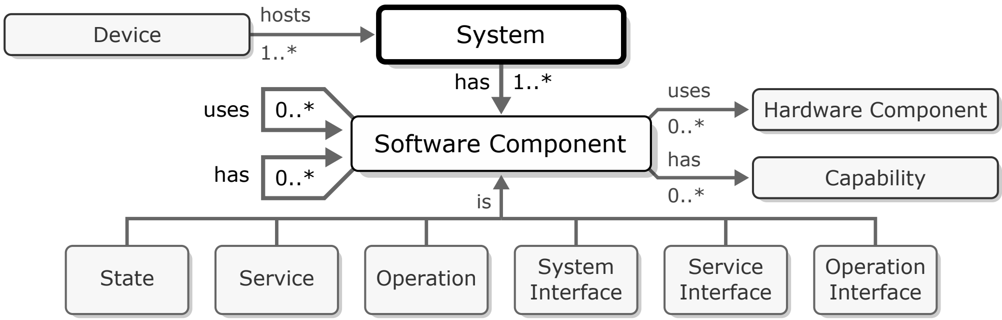

Figure 10: The system as a set of related software components, endowing a hosting device with new automation or compute capabilities. Other examples of software components could be operating systems, files, file systems, software libraries, programming language runtimes, databases and virtual machines.

Note that systems are not required to have any particular relationships to operating system processes, binary formats, virtual machines, and so on. They may be implemented in any way deemed suitable.

3.5 Service

A service is an identifiable set of service interfaces and operations. Each *service interface* had by a service represents one way in which is can receive messages, while each of its *operations* represents one activity the system providing the service can perform, if a valid and permitted message is received. As depicted in Figure 11, service interfaces pass on, or *route*, received messages to operation interfaces, each of which may execute one operation with a message as argument. Those operations may send additional messages via the same or other operation interfaces, which will pass them on toward other operations as described in Section 3.10.

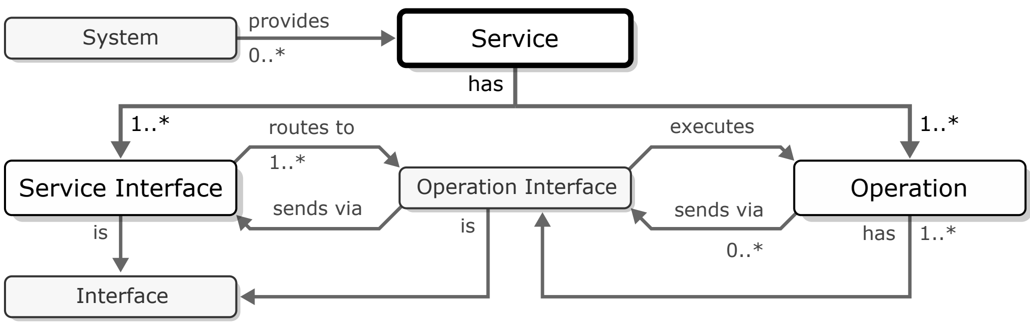

Figure 11: The service as a set of service interfaces and operations, making it possible for a providing system to offer the use of its capabilities to consuming systems via its service interfaces.

As all operations are software components, they may use any capabilities of the devices and systems that host and provide them, respectively. Once successfully consumed, an operation may send any number of messages to service operations provided by other systems, with any kinds of delays or intervals. When a service starts up and shuts down, it may be considered as if receiving an implicit message via an *initialize* or *terminate* operation, respectively. The messages to both of these operations may carry configuration data produced by and/or given to the system providing the service. It should not be possible for other systems to consume the initialize and terminate operations while the service in question is being provided.

3.6 System-of-Systems

A system-of-systems is a set of at least two systems, together facilitating one or more capabilities none of the constituent systems could have on its own. The facilitation of new capabilities is accomplished by the systems providing services and/or consuming each other's services.

While it may seem as if consuming services would hardly be enough for new capabilities to always emerge, it actually is the case. For example, let us assume that a system has the capability of turning on and off a light. That system also provides a service allowing for other systems to request that the light be turned on or off. If a different system can successfully consume that service, it also gains the capability of turning on and off that particular light. As a new system now may control that particular light, a new capability has emerged.

3.7 Local Cloud

A local cloud is an identifiable system-of-systems able to execute given tasks through the use of a pool of resources, each of which is managed by a so-called *supervisory system*. The resource pool of a local cloud could contain 3D-printers, autonomous unmanned vehicles, conventional servers, or anything else producing a value on demand. As depicted in Figure 12, the local cloud is distinct from other types of clouds by having at least one local boundary and one local resource, which means that it is physically tied to a concrete location. A local cloud could be engaged in manufacturing, repairs, heating, electricity distribution, workspace monitoring, drone fleet control, among many other possible kinds of physical activities. A local cloud may be stationary or mobile. A cloud that has no resources or boundaries tied to any particular physical locations should be referred to as a *virtual cloud*.

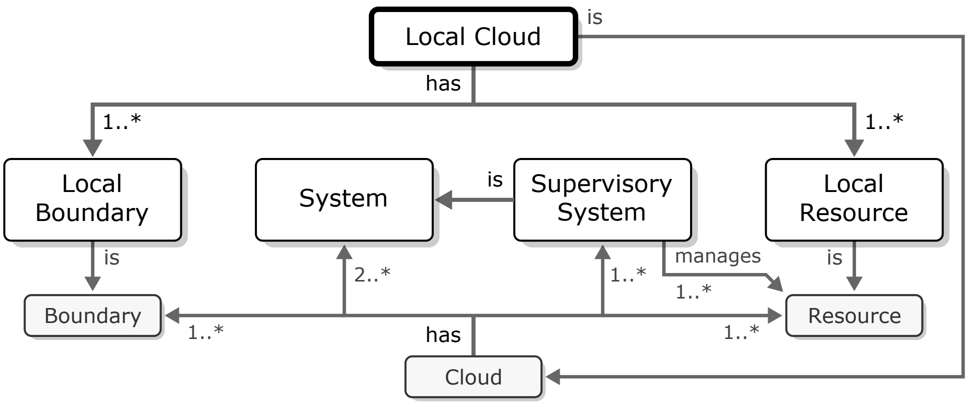

Figure 12: The local cloud as a regular cloud with at least one local boundary and one local resource, apart from any other boundaries or resources. Note that as a local cloud *is* a regular cloud, it also *has* at least two systems out of which at least one must be a supervisory system.

That a system is a supervisory system has no significance beyond that it manages at least one resource, of any type. That a local cloud has a boundary means that a distinction is being made between systems inside and outside the cloud. A boundary being local means that the distinction is being made by a physical attribute, such as device location, type of device, or physical attachment to a certain entity. Boundaries may be protected, which means that measures are in place to guarantee security, safety, real-time characteristics, or other local cloud attributes.

3.8 System-of-Local-Clouds

A system-of-local-clouds is two or more local clouds that consume each other's services to facilitate new capabilities. It is similar to the local cloud, with the exception of its subsystems are local clouds instead of plain systems. A system-of-local-clouds may have its own boundaries in addition to those of its constituent local clouds. Those boundaries are formed by attributes shared by all the constituent local clouds, such as certificates issued by the same organization, or physical attachment to the same network bus. A system-of-local-clouds cannot have resources beyond those of its constituent clouds, however.

3.9 Network

A network is a set of two or more devices, connected via network interfaces such that messages can pass between them. As shown in Figure 13, devices may be interconnected via intermediary devices, examples of which could be routers, switches, hubs, busses and firewalls. The term *end device* may be used to represent any device not being an intermediary device. Any technology able to connect devices is treated as facilitating networks, even if not typically associated with conventional networking methods.

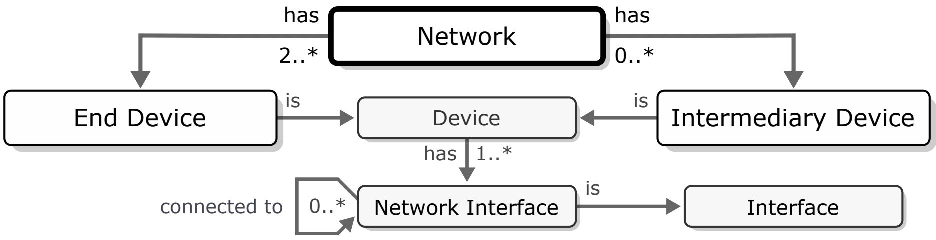

Figure 13: The network as a set of connected end devices, potentially interconnected via intermediary devices.

3.10 Interface

An interface is an identifiable boundary over which messages adhering to a supported protocol can cross, if those messages also satisfy all policies associated with that interface. When considering service provision and consumption, four types of interfaces become particularly relevant. These are (1) network interfaces, (2) system interfaces, (3) service interfaces and (4) operation interfaces, arranged as outlined in Figure 14.

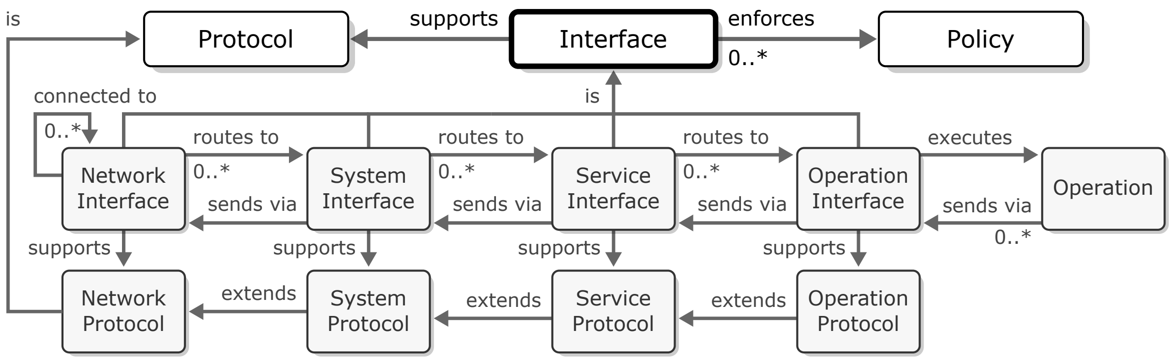

Figure 14: The interface as a set of supported protocols and enforced policies. Devices, systems, services and operations have their own interface types, forming four stages through which messages can be passed.

These four interface types form four stages, beginning with the network interface to the left and ending with the operation interface to the right. When a network interface receives a message, it is *routed* rightwards through each stage until it is found to be invalid, forbidden or reaches an operation. If the message is found to be invalid or forbidden, an error message may be propagated back to its sender. The receiving operation may react by sending additional messages, each of which will then be *sent* leftwards until it reaches a network interface. The network interface will send the message via networks until it reaches a device interface, which will repeat the receiving procedure we just covered. The operation first sending the message should be notified of any errors, both those received as messages and those noticed through other means.

Each interface only supports a single protocol and can only pass on messages of that protocol. For it to be possible to pass messages between stages, the protocol of the left stage must be *extended* by the protocol of the right stage. If, for example, a network interface supports the IP protocol [6] and system interface the TCP/IP [7] protocol, messages can pass between the two as the latter protocol is an extension of the former. The interface at each stage may elect to base its routing decision on any protocol details, including those used by earlier stages. This means that all three of the network, system and service stages may base their routing decisions on IP addresses, for example, if relevant to whatever use case is being targeted.

3.11 Protocol

A protocol is an identifiable set of message and state types, useful for determining what messages may move through the interfaces that support them. *Message types* determine what data conformant messages must and may contain, while each *state type* dictates when certain messages are acceptable in relation to a certain state. As shown in Figure 15, a protocol may also be defined as an extension of another protocol, conform to certain profiles and use certain encodings. Profiles and encodings are described in Sections 3.13 and 3.14.

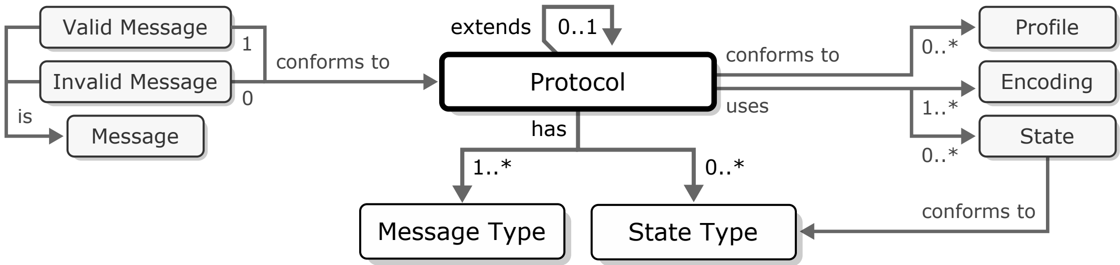

Figure 15: The protocol as set of message and state types, conforming to certain profiles.

A protocol may, when relevant, be considered as a function useful for testing if a given message is valid or invalid with respect to a current state. If the message is valid, the function returns the type of the message, which is needed to interpret the contents of that message. If the message is invalid, the function returns an indication of why the message failed to satisfy the message and/or state types of the protocol.

Protocols should only be concerned with the *destination* and *interpretation* of messages, not with whether they are *permitted* or not. This means that states should be associated with protocols only to ensure that received messages can be passed on or understood correctly. If, for example, a service controlling a door receives an message telling it to open its already open door, what does the sender of the message expect to happen? Nothing at all? That it closes and opens again? This ambiguity can be avoided by having the protocol be aware of the state of the door. Messages received when their interpretation is unclear can then be rejected.

3.12 Policy

A policy is an identifiable set of constraints, useful for determining if given messages are permitted or forbidden, as depicted in Figure 16. Policies may be concerned with authorization, contracts, economic goals, and so on.

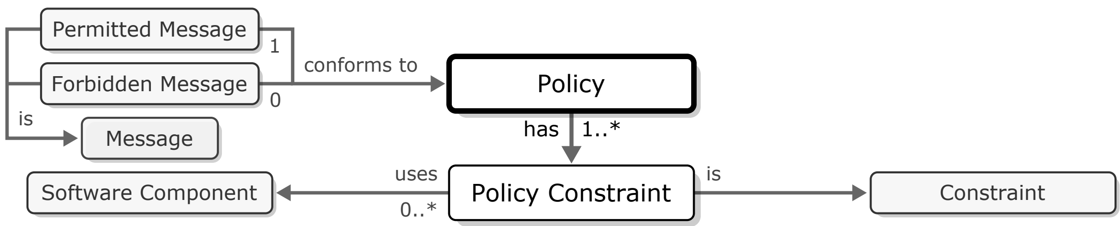

Figure 16: The policy as a set of constraints, useful for determining if messages are permitted or not.

Every policy may, when relevant, be regarded as a predicate function useful for testing if given message is permitted with respect to any kind of information. Policies are typically enforced by interfaces, as described in Section 3.10. If a message is forbidden with respect to one or more of the policies of an interface, those policies should be listed in any error message returned to the sender of that message.

While protocols help determine if a given message can be passed on or interpreted correctly, *policies* are meant to determine if the activity described by that message would occur under desirable conditions. For example, an interface may receive a message requesting that a certain pump be started. While the interpretation of the message may be clear, there may still be other conditions that make it undesirable for the pump to activate. If the pump is on fire, turning it on may present a safety hazard; if the system attempting to start the pump is unauthorized, the risk is higher for sabotage and other wasteful behaviors; and so on.

3.13 Profile

A profile is a set of constraints that can be added to a protocol, which can be used to require that certain flags or headers are included in certain messages, among other possible examples. While a protocol may only extend up to one other protocol, it may conform to any number of profiles. In Figure 17, two significant types of constraints are illustrated, the protocol constraint and the encoding constraint.

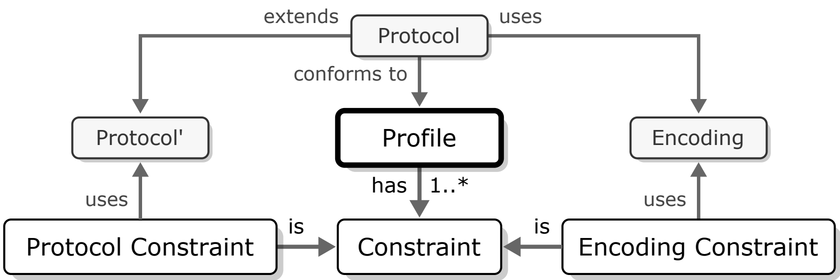

Figure 17: The profile as a set of protocol constraints, which may, for example, be concerned with protocols or encodings. *Protocol'* represents any protocol that *Protocol* could extend, directly or by any extended protocol.

A *protocol constraint* is defined in terms of a protocol that must be extended by any protocols conforming to its owning profile. For example, a service interface may support a custom extension of the HTTP protocol [8]. Such an extended HTTP protocol would, among other things, specify how a message will be routed from a system interface to a service interface, as described in Section 3.10. If that custom protocol is meant to be conformant to a certain profile, the constraints of that profile must be formulated in terms of HTTP without the extension, or any other protocol HTTP extends, namely TCP [7] or IP [6]. The profile in question may require that certain HTTP headers be included in every message, that certain TCP flags not be used, and so on.

An *encoding constraint* is defined in terms of an encoding that must be used by any protocols conforming to its owning profile. Such a constraint could be used to force message payloads to adhere to a certain semantics, such as SenML [9].

3.14 Encoding

An *encoding* is a set of data types that make up a language or structure in which data can be formulated and interpreted. The term is typically only used when considering encoders and decoders, which transform data from being expressed in one encoding into another. More specifically, an *encoder* turns data from an encoding suitable for processing into another suitable for transmission and/or storage, while a *decoder* performs the reverse operation. As implied by Figure 18, encodings suitable for transmission and/or storage are typically used to express messages.

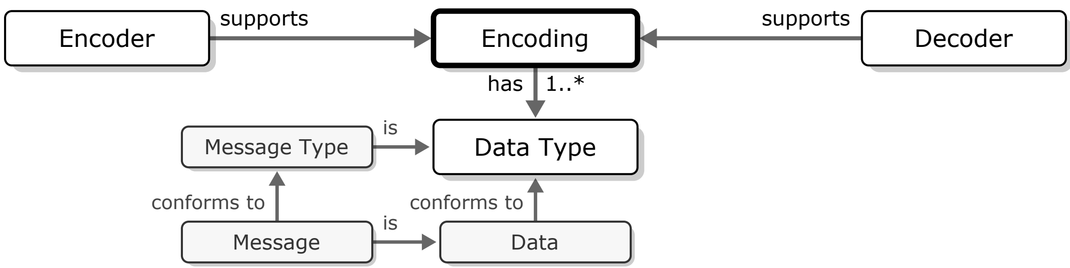

Figure 18: The encoding as set of data types, supported by certain encoders and decoders.

An encoding useful for *transmission* could, for example, be JSON [10], while an encoding useful for *storage* could be some kind of file or database format. An encoding useful for *processing* could be the format employed by a compute unit, virtual machine or computer language runtime, among other possible examples.

4 Conformance Requirements

For a document, model, or other artifact, to be allowed to claim conformance to *this work*, the following must be observed by that *derived work*:

1. At least one of the concepts defined in this work must be part of that derived work.
2. The derived work must make it explicit what concepts are taken from this work.
   (a) How this is done most suitably depends on the type of derived work. A document may include a normative reference to this document, while a model may want to give all relevant entities and relationships an attribute with the identity of this document, for example.
3. Every concept taken from this work must be represented by the name it is given here.
   (a) If important to be able to distinguish an Arrowhead concept from other such of relevance, concepts from this work may be qualified by the leading word "Arrowhead", as in, for example, "Arrowhead system" or "Arrowhead service function".
   (b) Note that some concepts defined here are given more than one name. In some cases one of these names may be designated as being preferred. Preferred names should be used by derived works. Whether or not a name is preferred is noted in the Glossary of Section 5 by it not referring to any other name as being preferred. If a referred name is designated as synonymous, it or any other name of the concept in question may be used.
4. Concepts taken from this work may be *specialized* and/or *simplified*, but must never be *contradicted*.
   (a) *Specialization* means that more constraints are applied to it than are presented here. For example, a certain derived work may require that all devices have compute units supporting a certain instruction set, or that every system provides a specific monitoring service, and so on.
   (b) *Simplification* means that entities, relationships or attributes introduced here are omitted due to being outside the scope of the derived work. For example, a technical document may not be concerned with stakeholder roles, while a model of certain types of local clouds may not be concerned with whether or not artifacts are resources or not, and so on.
   (c) *Contradiction* means that an attribute or other constraint is introduced that makes it impossible to reconcile the concepts presented here with those in the derived work. A derived work must not, for example, demand that no devices ever host systems. Contradictions generally only occur when some relationship or attribute is both demanded to exist and not to exist at the same time.
5. If a different graph notation is used than the one described in Section 1.3.1, the derived work must either describe how its notation maps to the notation here, or refer to a work making such a description.
   (a) The graph constructs that have to be mapped are as follows:
      i. *entities*, which are boxes with solid lines and names inside them;
      ii. *relationships*, which are unidirectional arrows with *names* and *quantifiers* that denote association;
      iii. *attributes*, which are special properties expressed in text only.

      Each relevant relationship name and attribute of this document must be mapped to an equivalent construct in the target notation.
   (b) In practice, only text documents claiming to adhere to the graph diagram notation of Section 1.3.1 are exempt from having to describe or refer to such a notational mapping. As mappings to this document will be hard to produce rigorously without text, we expect all such mappings to be described in text documents.

4.1 ISO/IEC/IEEE 42010

The ISO42010 [4] standard provides a uniform way for system architects to produce architectural descriptions, viewpoints, frameworks and description languages. In the context of ISO42010, this work can be used as a *metamodel* part of an model kind, as illustrated in Figure 19.

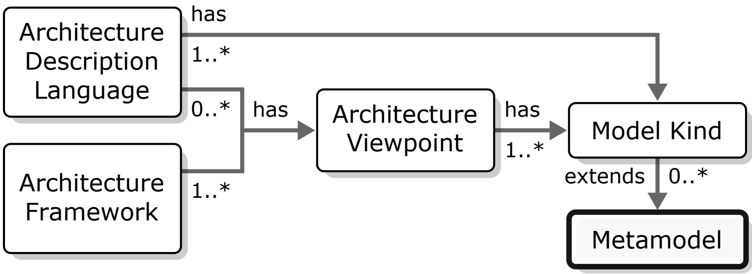

Figure 19: The metamodel as a part of an ISO42010 model kind, which in turn may be referenced by architecture description languages and architecture frameworks.

Using this work as a metamodel largely entails referencing a work that maps the concepts of this work to a relevant modeling language, as discussed in conformance requirement 5. The mapping work must, of course, satisfy all conformance requirements outlined earlier in this section. The use of metamodels is described more fully in Annex B, Section B.2.6 of ISO42010 [4]. If you want to learn more about the standard and how to use it, please refer to the standard itself or other relevant learning resources[1].

[1] At the time of writing (2021-12-15), guides to ISO42010 were available at `http://www.iso-architecture.org/42010`.

5 Glossary

This section provides an alphabetically sorted list of all significant terms introduced or named in this document. Each term consisting of more than one word is sorted by its final, or qualified, word. This means that the definition of service protocol, for example, is found at Protocol, Service.

Many of the definitions are amended with notes and references to IoTA:AF [3], ISO42010 [4], SOA-RM [1] and RAMI4.0 [5], which are always listed after the definition they amend. Regular notes are numbered, while those making a comment on a definition in IoTA:AF, ISO42010, SOA-RM or RAMI4.0 are introduced with the abbreviations just listed.

Acquirer

A stakeholder in the process of acquiring, or considering to acquire, a system or system-of-systems with the intent to operate and/or use it. See Section 3.1.

Architect

A stakeholder who seeks to improve upon or extend the Arrowhead framework itself, by, for example, writing core documentation or producing architectural descriptions. See Section 3.1.

Architecture

A model of a system-of-systems defined in terms of a certain (1) goals, ambitions or other principles; (2) an environment, either abstract or concrete; (3) as well as significant life-cycle events, such as construction, maintenance or decommissioning.

> **ISO42010** defines architecture as “<system> fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”. Our definition should be interpreted as being equivalent. Note that ISO42010 uses the term “element” to refer to what we call a *model entity*.

> **SOA-RM** defines software architecture as “the structure or structures of an information system consisting of entities and their externally visible properties, and the relationships among them”. That definition is equivalent to our definition of model, with the exception that the thing being modeled has to be an information system. As our definition is concerned with a model and a system-of-systems, which must be an information system, we regard out definition as compatible but more specific.

> **RAMI4.0** defines architecture as the “combination of elements of a model based on principles and rules for constructing, refining and using it”. We consider “combinations of elements of a model” to be a “model of a system-of-systems” and to be “based on principles and rules for constructing, refining and using it” as being concerned with principles, an environment and life-cycle events. Our definition should be interpreted as being compatible but more specific.

Architecture, Software

Prefer Architecture.

Arrowhead

The name of the initiative part of which this document and the rest of the Arrowhead framework is being produced.

Artifact

A thing or object, tangible or intangible.

Asset

Synonymous to Resource.

> **RAMI4.0** defines asset as an “object which has a value for an organization”. See Resource for a comparable term.

Attribute

A name/value pair of data, associated with either an entity or a relationship.

Note 1 A attribute is a form of metadata.

Automation

The control of a process by a mechanical or electronic apparatus, taking the place of human labor.

Boundary

A point or border where either two or more artifacts meet or one artifact ends.

Boundary, Cloud

A boundary separating the artifacts belonging to a cloud from those not belonging to it.

Note 1 A cloud boundary can be local or virtual, depending on if the boundary is formed by physical or virtual attributes.

Boundary, Local

A boundary that exists in the physical world.

Note 1 Local boundaries can be facilitated by walls, locations of operation, attachment to certain vehicles or power sources, and so on.

Boundary, Virtual

A boundary that exists only virtually.

Note 1 Virtual boundaries can be facilitated by cryptographic secrets, identifiers, ownership statements, contracts, and so on.

Builder

A stakeholder constructing Arrowhead automation systems by assembling and preparing devices, as well as installing systems on those devices. See Section 3.1.

Capability

A task, of any nature, that can be executed by an artifact.

Note 1 The term must be understood in the most general sense possible. It includes the abilities of hosting systems, reading from sensors, triggering actuators, among many other possible examples.

SOA-RM defines a capability as “a real-world effect that a service provider is able to provide to a service consumer”. Our definition is more general in the sense that not only service providers are allowed to have capabilities. See also Capability, System.

Capability, Device

A capability facilitated by the hardware components of a device. See Section 3.3.

Capability, System

A capability facilitated by the software components of a system. See Section 3.4.

Cloud

A bounded system-of-systems able to independently execute given tasks through the use of a pool of resources.

Note 1 When the term “cloud” is used elsewhere, it often refers to clouds with only virtual resources, such as compute, storage and software-defined network utilities. Here, we refer to such clouds as virtual clouds. By making the unqualified word “cloud” less specific, it becomes more clear how our local cloud concept shares similarities with other types of clouds.

Cloud, Local

A cloud bound to a physical location due to its acting on or producing local resources. See Section 3.7.

IoTA:AF provides an introduction to the local cloud concept in its second chapter, as well as an architectural definition in its third chapter. The following is an excerpt from the introduction:

> *The local cloud concept takes the view that specific geographically local automation tasks should be encapsulated and protected. These tasks have strong requirements on real time, ease of engineering, operation and maintenance, and system security and safety. The local cloud idea is to let the local cloud include the devices and systems required to perform the desired automation tasks, thus providing a local "room" which can be protected from outside activities. In other words, the cloud will provide a boundary to the open internet, thus aiming to protect the internal of the local cloud from the open internet.*

The third chapter contains the following:

> *In the Arrowhead Framework context a local cloud is defined as a self-contained network with the three mandatory core systems deployed and at least one application system deployed [...]*

Both of these descriptions are practical, in the sense that they emphasize engineering aspects. As this document is a reference model, engineering aspects are out of scope. The more general terms "geographically local", "room" and "boundary" clearly highlight the physicality of the local cloud itself, while the depiction of "devices" performing "automation tasks" makes it apparent that some kind of physical activity is involved, such as manufacturing. Finally, the local cloud being "encapsulated", "protected" and "self-contained" indicates that it is understood to exhibit a degree of independence with respect to the tasks it is given, which we expect all kinds of clouds to exhibit. Our definition should be interpreted as a summation of these characteristics.

Cloud, Local Automation

Prefer Cloud, Local.

Cloud, Virtual

A cloud unbound by physical location by only acting on or producing virtual resources.

Communication

The activity of sending and/or receiving messages.

Communication, Service-Oriented

Communication described in terms of the provision and consumption of services.

Component

An artifact that can be part of another artifact and contribute to it facilitating its capabilities.

Note 1 The term "component" should never be used to refer to a system being a constituent of a system-of-systems. Such a system should be referred to as being a subsystem.

RAMI4.0 makes no practical distinction between components and systems, as is done here. See System for more details.

Component, Hardware

A physical component that can only be part of a device. See Section 3.3.

Component, Software

A virtual component that can only be part of a system. See Section 3.4.

Configuration

A set of changeable attributes that directly influence how a system exercises its capabilities.

Configure

To update a configuration.

Connection

An active medium through which attached interfaces can communicate.

Constraint

A attribute that imposes constraints, or limits, on an entity or relationship.

Note 1 The presence of constraints enable validation.

Note 2 Perhaps a bit counterintuitively, a constraint *adds* information to its target by reducing the ways in which it could be realized.

Constraint, Encoding

A constraint imposed on an encoding. See Section 3.13.

Note 1 An encoding constraint could require that an optional data field be present or omitted, require that the values of certain fields satisfy a certain predicate function, and so on.

Constraint, Policy

A constraint imposed by a policy. See Section 3.12.

Constraint, Protocol

A constraint imposed on a protocol. See Section 3.13.

Consumer, Service

A system currently consuming a service by sending a message to one of its operations.

Note 1 The term may also be used to refer to a stakeholder, in which case the stakeholder must be interpreted as if consuming services via systems.

SOA-RM defines a service consumer as "an entity which seeks to satisfy a particular need through the use [of] capabilities offered by means of a service". We require that the one consuming the service is (1) a system rather than just any entity, as well as (2) that the capabilities of the consumed service be exercised by invoking a function.

Consumption, Service

The act of consuming a service by sending a message to one of its operations. See Consumer, Service.

Data

A sequence of datums recording a set of descriptions via the structure superimposed by a data type.

Note 1 Let us assume that some data is going to be sent to a drilling machine. The type associated with the data requires that it always consists of 8 bits, organized such that the first 4 bits indicate the speed of drilling in multiples of 100 rotations per minute, while the latter 4 determine how much to lower the drill in multiples of 5 millimeters. A state that could be expressed with those 8 bits is `0100 1101`. If each of the two sequences of 4 bits is treated as a big-endian integer with base 2, they record 4 and 13 in decimal notation. This would indicate that the drill should spin at $4 * 100 = 400$ rotations per minute and be lowered $13 * 5 = 65$ millimeters.

Note 2 Without knowledge of the types and context associated with some data, that data cannot be interpreted.

Datum

A variable expressing one out of a set of possible values. See also State.

Note 1 A familiar example of a datum may be the bit, or *binary digit*. Its possible set of symbols is $\{0, 1\}$.

Decode

The act of transforming data from being expressed in a encoding suitable for transmission or storage to another encoding suitable for interpretation.

Note 1 Decoding is the reverse of encoding.

Note 2 The term can also be used to express the act of a human interpreting data.

Decoder

An entity capable of decoding data.

Description

Facts about an entity or class of entities, expressed in the form of a model, a text, or both.

Design *(noun)*

Every document, model and other record describing how a certain artifact can be implemented.

Design *(verb)*

The activity of producing designs.

Developer

A stakeholder developing the components that make up devices and/or systems. See Section 3.1.

Device

A physical entity made from hardware components with the significant capability of being able to host systems. See Section 3.3.

IoTA:AF defines device as “a piece of equipment, machine, hardware, etc. with computational, memory and communication capabilities which hosts one or several Arrowhead Framework systems and can be bootstrapped in an Arrowhead local cloud”. The definition provided here should be interpreted as being equivalent.

Device, Connected

A device that is connected to at least one other device via their network interfaces, enabling them to communicate.

Device, End

A connected device being the intended recipient of a message.

Device, Human Interface

A device with sensors and actuators that together make up a human interface.

Device, Intermediary

A connected device that receives and forwards messages toward end devices.

Encode

The act of transforming data from being expressed in a encoding suitable for interpretation to another encoding suitable for transmission or storage.

Note 1 Encoding is the reverse of decoding.

Note 2 The term can also be used to express the act of a human recording data.

Encoder

An entity capable of encoding data.

Encoding *(noun)*

A data type used to structure and interpret certain data. See Section 3.14.

Entity

An artifact with an identity, allowing for it to be distinguished from all other artifacts. See Section 3.2.

Note 1 An entity being uniquely identifiable does not necessarily mean that it is associated with a certificate or identifier. It only means that a description can be rendered that unambiguously refers to the entity in question.

SOA-RM mentions the word "entity" nine times, but provides no explicit definition. We assume their definition to match that of a regular English dictionary, such as "something that has separate and distinct existence and objective or conceptual reality" [11]. Our definition should be interpreted as being equivalent.

RAMI4.0 defines entity as an "uniquely identifiable object which is administered in the information world due to its importance". Our definition should be interpreted as being equivalent.

Entity, Class of

A set of entities that share a common attribute.

Framework

A set of ideas and software artifacts that frame and address a problem domain of a certain community of stakeholders. See Section 2.

ISO42010 defines architecture framework as "conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders". Our definition of *framework of ideas* should be interpreted as being compatible with that of ISO42010.

SOA-RM defines framework as "a set of assumptions, concepts, values, and practices that constitutes a way of viewing the current environment". Our definition of *framework of ideas* should be interpreted as being equivalent to that of SOA-RM.

Framework, Architecture

Prefer Framework.

Framework, Arrowhead

Either of the framework of ideas and the framework of software maintained by the Arrowhead project. See Section 2.

Framework, Idea

A set of assumptions, concepts, values and practices that frame a certain problem domain. See Section 2.

Framework, Software

A set of software specifications, implementations and other artifacts meant to help address the problem domain of a certain framework. See Section 2.

Function

A conceptual mathematical construct that transforms given input values into output values.

Note 1 Most functions can be implemented as software.

Function, Predicate

A function whose output value must be a *Boolean variable*. An output value of *true* indicates that the function is *satisfied*, while an output value of false indicates it being *violated*.

Note 1 A *Boolean variable* can only be either of the two mentioned values, *true* and *false*.

Hardware *(adjective)*

The property of being physical, as opposed to being virtual. See Software *(adjective)*.

Hardware *(noun)*

A physical artifact. See Hardware *(noun)*.

HID

Abbreviation for Device, Human Interface.

Hosting, System

The act of making a service available for consumption by running its software and giving that software access to a network.

Human

Prefer Person.

Identifiable

The property of being possible to distinguish a certain artifact from all other artifacts.
Being identifiable is the same as being an entity.

Identification

The process through which an entity determines and/or verifies the identity of another entity.

Identifier

Data associated with an entity that allows for it to be identified.

Identity

The aspect or aspects, such as identifiers, that makes an entity distinct from all other entities.

Image, Software

A data artifact comprised of instructions that could be executed by a compatible compute unit or virtual machine.

Implementation

The realization of a design as a set of artifacts.

Implementation, Software

An implementation comprised of software artifacts.

Note 1 The term may also be used to refer to all software artifacts part of an implementation.

Implementation, Hardware

An implementation comprised of hardware artifacts.

Note 1 The term may also be used to refer to all hardware artifacts part of an implementation.

Industry 4.0

The fourth industrial paradigm, primarily characterized by high degrees of computerization, digitization and interconnectivity. See also [5].

Instance, Software

A software image currently being executed by a compute unit or virtual machine.

Note 1 The same image can be executed any number of times, even in parallel. Each execution of that image is its own instance, distinct from all other instances.

Interconnection

A connection that passes through one or more intermediary devices.

Interface

A boundary where messages of certain protocols can pass between a connection and an entity, between two entities, or between an entity and a . See Section 3.10.

Interface, Human

An interface through which a may send and/or receive messages to/from an entity.

Interface, Network

An interface through which a device could communicate with other devices, or with itself, over a network.

Interface, Operation

An interface through which a certain operation of some service can be consumed.

Interface, Service

An interface through which a certain service can be consumed.

Note 1 Consuming a service requires that messages be passed from its device to its system, and then from its system to the service itself. As the software making up the service is owned by the system, it is the system that is understood to produce any responses. Those are passed on via its device.

SOA-RM defines service interface as “the means by which the underlying capabilities of a service are accessed”. Our definition should be interpreted as being equivalent.

Interface, System

An interface through which a system may send and/or receive messages via its hosting device.

Kind, Model

A description of how to produce a certain kind of model.

ISO42010 defines model kind as “conventions for a type of modelling”. It also provides “data flow diagrams, class diagrams, Petri nets, balance sheets, organization charts and state transition models” as examples of what model kinds could establish. Our definition must be considered as being either equivalent or incorrect.

Language, Architecture Description

A formal language in which architectures can be described.

Maintainer

A stakeholder involved in maintaining devices and systems, primarily by repairing and upgrading devices and updating system software. See Section 3.1.

Message

Data sent or received via an interface. Every message must identify a target service operation and may contain metadata and a payload.

Note 1 The metadata of a message represent the details it contains about its transmission and interpretation. Metadata are always concerned with the protocol of the message and with satisfying the policies of the interfaces that must be passed on its journey to its target operation. Metadata may be added, modified and/or used by interfaces when messages pass through them.

Note 2 A payload is an input to a service operation. Operations are not required to expect input data, which is why having a message payload is optional.

Message, Error

A message indicating why the request expressed by some other message could not be executed.

Note 1 We expect the receiver of most error messages to be the respective senders of the messages that could not be executed.

Message, Forbidden

A message that fails to satisfy a policy of concern and, therefore, will not be executed. See Section 3.12.

Message, Invalid

A message that fails to satisfy a protocol of concern and, therefore, will not be executed. See Section 3.11.

Message, Permitted

A message that does satisfy a policy of concern and, therefore, will be executed if all other policies are also satisfied. See Section 3.12.

Message, Valid

A message that does satisfy a protocol of concern and, therefore, will be executed if it is permitted. See Section 3.11.

Metadata

Data describing other data.

Metamodel

A basic set of model constructs that can be extended by other models.

Note 1 A metamodel can be thought of as a general language in which more specific models can be expressed. Just as a given sentence in a human language can be determined to be valid or invalid, a model can also be verified to be correct in relation to its metamodels, if any.

ISO42010 defines metamodel as what "presents the [architectural description] elements that comprise the vocabulary of a model kind". It further adds that a "metamodel should present entities[,] attributes[,] relationships [and] constraints". Our definition must be considered as being either equivalent or incorrect.

Model

A representation of facts in the form of a graph, consisting of entities, relationships and attributes.

Note 1 Models can be expressed or recorded in many ways, including as visual diagrams, spoken words, text and binary data.

Note 2 Models can be human-readable, machine-readable, or both.

Network

A set of two or more end devices, connected in such a manner that any systems they host are able to communicate. See Section 3.9.

Operation

An activity a service can perform if receiving a valid and permitted targeting that operation. See Section 3.5.

Operation, Service

Prefer Operation.

Operator

A stakeholder responsible for the configuration and oversight of systems and the resources those systems manage. See Section 3.1.

Organization

A stakeholder comprised of an organized body of other stakeholders and/or other persons.

Owner

A stakeholder that owns significant resources and/or other artifacts. See Section 3.1.

Person

A human being.

Policy

A set of constraints, of any nature, that must be satisfied by all messages passed on by an interface. See Section 3.12.

SOA-RM defines policy as “a statement of obligations, constraints or other conditions of use of an owned entity as defined by a participant”. Our definition should be interpreted as being equivalent.

Policy, Message

Prefer Policy.

Profile

A set of constraints superimposed on a protocol. See Section 3.13.

Note 1 A profile *never* introduces more messages or states to a protocol. It adds constraints to existing messages and states.

Note 2 A profile could, for example, introduce an authentication mechanism to a protocol by requiring that a certain type of token be included in each message. It could demand that a certain protocol be extended, or that a particular kind of encoding be used for message bodies, and so on.

Profile, Protocol

Prefer Profile.

Project, Eclipse Arrowhead

The effort of the Arrowhead community to increase the utility of the Arrowhead framework.

Protocol

A model of communication defined in terms of states and messages. See Section 3.11.

Note 1 The states, if any, dictate the outcomes of sending certain messages. For example, let us assume that some state can be either `BUSY` or `READY`. If the former state would be the active when a certain message is received, the designated response could be an error message. If, however, the `READY` state would have been active, the state could be transitioned to the `BUSY` value and a success response be provided to the sender.

Protocol, Extensible

A protocol allowing for subprotocols to be formulated in terms of its messages.

Note 1 Every new message introduced by a subprotocol must be a valid message of its superprotocol.

Note 2 Many of the currently prevalent protocols are designed with the intent of being extensible. For example, HTTP [8] provides provisions for an extending protocol to define its own set of directory operations, to simultaneously support multiple encodings, and so on.

Note 3 As long as a given protocol provides at least one message whose contents can be arbitrary, a subprotocol can be produced. This means that even protocols not designed to be extended can, in some contexts, be meaningfully used to define subprotocols.

Protocol, Network

A protocol implemented by an network interface. See Section 3.11.

Protocol, Operation

A protocol implemented by an operation interface. See Section 3.11.

Note 1 An operation protocol is always an extension of a service protocol.

Protocol, Service

A protocol implemented by a service interface. See Section 3.11.

Note 1 A service protocol is always an extension of a system protocol.

Protocol, System

A protocol implemented by a system interface. See Section 3.11.

Note 1 A system protocol is always an extension of a network protocol.

Provider, Service

A system that makes services available for consumption to other systems.

Note 1 If used to refer to a stakeholder, the term must be interpreted as if that stakeholder provides services via systems it controls.

SOA-RM defines a service provider as "an entity (person or organization) that offers the use of capabilities by means of a service". Our definition is equivalent only if referring to a stakeholder as a service provider, as described in Note 1.

Provision, Service

The act of making services available for consumption. See Provider, Service.

Relationship

A named uni-directional association between two model entities.

Researcher

A stakeholder involved in the analysis or development of significant entities, particularly with the ambition of facilitating attributes or use cases that cannot be realized without refining, extending or replacing those entities. See Section 3.1.

Resource

An artifact that is of value to a stakeholder or of use to another artifact.

Note 1 Any type of artifact can be a resource, which includes everything from local resources, such as raw materials or devices, to virtual resources, such as systems or data.

Note 2 An artifact stops be a resource when it is perceived as having no value or use, at which point it may be destroyed, recycled or sold to someone that does perceive it as a resource, for example.

Resource, Local

A resource whose value or utility is inextricably tied to at least one physical attribute.

> **Note 1** Examples of local resources could be raw materials, drills, pumps, power stations, or drones.

Resource, Virtual

A resource whose value or utility is not derived from any physical attribute.

> **Note 1** Examples of virtual resources could be compute, storage, or software-defined network utilities. While all of these resources are facilitated by physical entities, namely various types of computer equipment, they do not depend on any particular machines. They can be moved to different machines without loosing their value or utility.

Role

An assignment, objective, or other responsibility, that makes a person or organization into a stakeholder.

Role, Stakeholder

Prefer Role.

Routing

The act of forwarding a message towards the service operation it targets.

Routing, Message

Prefer Routingrouting.

Service

A set of operations that can be provided by a system via one or more service interfaces. See Section 3.5.

> **IoTA:AF** defines a service as "what [is] used to exchange information from a providing system to a consuming system". It further adds that "in a service, capabilities are grouped together if they share the same context". The definition presented here should be interpreted as being compatible but more specific about how information is exchanged and capabilities are exercised.

> **SOA-RM** defines a service as "the means by which the needs of a consumer are brought together with the capabilities of a provider". Our definition is more specific about how the capabilities of a service are made available.

> **RAMI4.0** defines a service as "separate scope of functions offered by an entity or organization via interfaces". Given that our understanding of "operation" is compatible with the RAMI4.0 definition of "function", our definition of "service" should be considered as being equivalent.

Software *(adjective)*

The property of being virtual, as opposed to being physical. See Hardware *(adjective)*.

Software *(noun)*

A set of sequences of instructions that can be executed by a compute unit.

> **Note 1** A software does not necessarily have to be expressed in the instruction set native to the compute unit expected to execute it. Virtual machines, interpreters and other utilities may be used to execute instructions, which means that our definition of 'software" may be more open-ended than what initially may seem to be the case.

Specification

A detailed description, outlining the design of some artifact of concern.

Specification, Software

A specification concerned only or primarily with software.

Stake

Any type of engagement or commitment.

Stakeholder

A person or organization with one or more roles, which gives that stakeholder at least one relationship to one artifact. See Section 3.1.

State

One out of all possible sequences of values that could be expressed by the datums of some data.

Note 1 If the data would consist of a sequence of bits, each of which can only have the values 0 and 1, a state becomes a pattern of zeroes and ones those bits could record. Given four bits, possible states could, for example, be `0010` and `1001`.

Note 2 The term is often used as a wildcard for any kind of storage construct, including bit flags, state machines and graph databases.

State, Protocol

The state of a protocol in active use, determining what messages it currently deems valid. See Section 3.11.

Subprotocol

A protocol that is realized as an extension of another protocol.

Subsystem

A system or system-of-systems being a constituent of a larger system-of-systems.

Superprotocol

A protocol that is extended by another protocol.

Supplier

A stakeholder in the process of supplying, or considering to supply, artifacts, such as devices and systems, to an acquirer.

System

A software entity capable of providing services, consuming services, or both.

IoTA:AF defines a system as "what is providing and/or consuming services". It further adds that "a system can be the service provider of one or more services and at the same time the service consumer of one or more services". The definition presented here should be interpreted as equivalent.

System, Automation

Any kind of system, compatible with the Arrowhead framework or not, meant to facilitate some form of automation.

System-of-Local-Clouds (SoLC)

A set of local clouds that consume each other's services in order to facilitate a capability none of the constituent local clouds could provide on its own. See Section 3.8.

System, of-Systems (SoS)

A set of systems that consume each other's services in order to facilitate a capability none of the constituent systems could provide on its own. See Section 3.6.

IoTA:AF defines a system-of-systems as "a set of systems, which [...] exchange information by means of services". It further adds that "when Arrowhead compliant systems collaborate, they become a System of Systems in the Arrowhead Framework's definition". While we clarify here that the desired outcome of collaboration is the facilitation of new capabilities, the definitions should be interpreted as being equivalent.

System, Opaque

A system that is unable to either provide or consume services.

System, Supervisory

A system that is tasked with managing one or more resources beyond its direct control.

> **Note 1** All systems are managing the resources provided to them by their hosting devices, such as primary memory, compute time, and so on. This term is meant to capture the systems that are engaged in overseeing and/or managing resources beyond those directly provided. Examples of such scenarios could be a single system being responsible for provisioning other devices, or a system using its robot device to collect and handle raw materials.

Type

A description of how datums are to be arranged to encode certain facts. See also Data.

> **Note 1** While this definition may seem foreign, it does capture how integer types, classes, enumerators and other types are used in the context of a programming language or encoding. In the end, all data are bits or other symbols. From our perspective, types serve to group those symbols and assign them meaning.

> **Note 2** A type provides only syntactic, or structural, information about data. While knowing the type used to code some data is required for its interpretation, contextual knowledge is also needed. For example, a type may specify a `name`, but it will not indicate when or why that name is useful. That information would have to be provided via documentation or some other means.

Type, Data

Prefer Type.

Type, Message

The type dictating the structure of the data in a message.

Type, State

The type specifying a set of possible states and transitions between them.

Unit, Compute

A hardware component able to execute software compatible with a certain instruction set.

Unit, Memory

A hardware component maintaining a set of changeable datums, which are primarily useful for maintaining states.

User

A stakeholder taking, or trying to take, advantage of the end utility of a certain entity. See Section 3.1.

> **Note 1** The activity of *using* an entity is not related to its coming into existence, maintenance, decommissioning, or any other peripheral activity. When a stakeholder uses an entity, that entity produces whatever end value it was designed to produce.

Validation

The process through which it is determined if a model satisfies a constraint.

Viewpoint, Architecture

An description of a problem domain, specifying (1) concerns, (2) conventions and (3) model kinds.

> **ISO42010** defines architecture viewpoint as "work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns". Our definition is meant to express this definition and must be considered as being either equivalent or incorrect.

6 References

[1] C. Bashioum, P. Behera, K. Breininger *et al.* “Reference Model for Service Oriented Architecture”. *OASIS Standard soa-rm*, Organization for the Advancement of Structured Information (OASIS), October 2006. URL `https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html`. Version 1.0, accessed 2021-10-05.

[2] “OMG Systems Modeling Language (OMG SysML™)”. *OMG document*, Object Management Group (OMG), November 2019. URL `https://www.omg.org/spec/SysML/1.6/`. Version 1.6, accessed 2021-10-06.

[3] J. Delsing (editor). *IoT Automation: Arrowhead Framework*. CRC Press, 2017. ISBN 9781498756761.

[4] “Systems and software engineering – Architecture description”. *International Standard 42010:2011*, ISO/IEC/IEEE, December 2011. doi:10.1109/IEEESTD.2011.6129467.

[5] P. Adolphs, S. Berlik, W. Dorst *et al.* “Reference Architecture Model Industrie 4.0 (RAMI4.0)”. *DIN SPEC 91345:2016-04*, Deutsches Institut für Normung (DIN), April 2016. doi:10.31030/2436156.

[6] S. Deering and R. Hinden. “Internet Protocol, Version 6 (IPv6) Specification”. *RFC 8200*, RFC Editor, July 2017. doi:10.17487/RFC8200.

[7] J. Postel. “Transmission Control Protocol”. *RFC 793*, RFC Editor, September 1981. doi:10.17487/RFC0793.

[8] R. Fielding *et al.* “Hypertext transfer protocol (HTTP/1.1): Message syntax and routing”. *RFC 7230*, RFC Editor, March 2014. doi:10.17487/RFC7230.

[9] C. Jennings, Z. Shelby *et al.* “Sensor Measurement Lists (SenML)”. *RFC 8428*, RFC Editor, August 2018. doi:10.17487/RFC8428.

[10] T. Bray. “The JavaScript Object Notation (JSON) Data Interchange Format”. RFC 7159, March 2014. doi:10.17487/RFC7159.

[11] Merriam-Webster. “Entity”. URL `https://merriam-webster.com/dictionary/entity`. Accessed 2021-10-12.

7 Revision History

7.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1				

7.2 Quality Assurance

No.	Date	Version	Approved by
1			