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 service-oriented 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. It does not contain any architectural patterns or definitions, which means that it, by itself, is not a sufficient foundation for designing Arrowhead systems. While its definitions are presented as a model, the document does not 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 allows for the capabilities and other attributes of individual systems to be described, shared and exploited dynamically by communicating devices. Most notably, this makes it possible to design distributed systems that are highly resilient to degradations and failures, as well as being cheap, in terms of time and money, to develop, extend, upgrade, retrofit and decommission.

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. We do not present architectural design patterns, system specifications or implementations, other than as illustrative examples. While highly relevant in the wider context of Arrowhead, such topics are out of the scope of this particular document.

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 of Arrowhead systems, or of the Arrowhead framework itself.
- 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; and
5. *has*, meaning that the target is *used by* and must cease to exist without the source.

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.

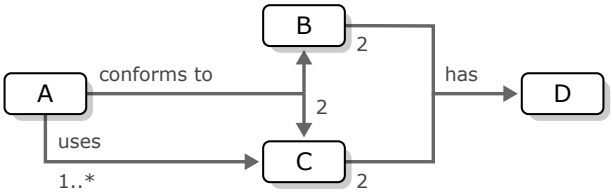

Figure 1: An example graph diagram. **A** *conforms to* 1 **B**, as no quantifier is associated with the arrow from **A** to **B** and 1 is the default quantity. **A** also *conforms to* 2 **C**s, as well as *uses* 1 or more **C**s. Both **B** and **C** *has* 2 **D**s.

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, given that the used reader supports this feature. 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. As they are primarily architectural, 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, introducing its most fundamental concepts and how those concepts relate to each other.

Section 3 An in-depth and formal description of the most fundamental concepts of Arrowhead, as well as other important auxiliary concepts. Each of its subsections is concerned with one 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, which is illustrated in Figure 2, consists of two subframeworks: a framework of ideas and a framework of software. The framework of ideas formulates and frames the *problem domain* the framework of software is meant to help address. By this we mean that the framework of ideas presents the assumptions, concepts, values and practices that should be applied when specifying architectural or other technical documentation and when implementing any kinds of Arrowhead systems or components.

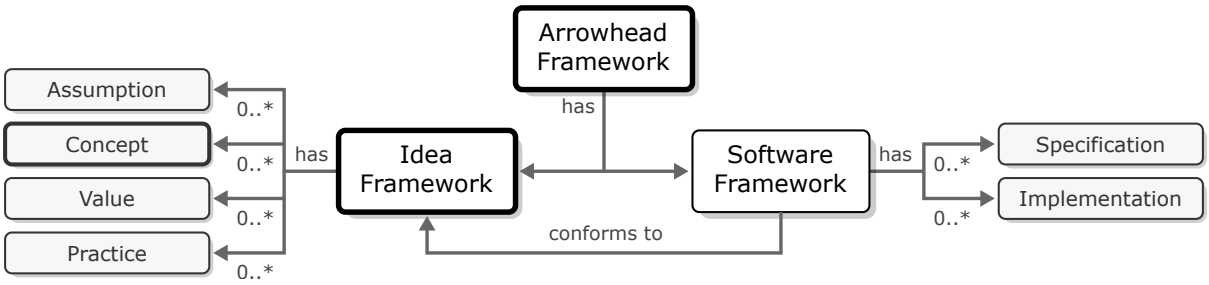

Figure 2: The two subframeworks of the Arrowhead framework, concerned with *ideas* and *software*. The specifications and implementations of the software framework must conform to the assumptions, concepts, values and practices of the idea framework. The concepts of the idea framework are outlined in this document.

This document is part of Arrowhead's framework of ideas. As such, it is primarily concerned with defining concepts. However, before we move on to consider our overview those concepts, we will first present a few examples of other key framework ideas.[1] It is, for example, *assumed* that the framework may be applied in contexts where the primary activity is markedly physical, such as in transportation, mining, manufacturing, electricity generation, healthcare, and so on. One of the system characteristics *valued* by the framework is *resilience*, or the expectation that every system should do its outmost to mitigate and recover from degradations, failures or other contingencies that may affect its ability to perform its designated tasks. Finally, one of its *recommended practices* is that every system-of-systems should be thoroughly documented at every level, from its smallest components up to its most high-level interactions.

The rest of this section gives an overview of the most fundamental concepts of the framework. It is meant to prepare you for the next section, where the same concepts, and other supporting concepts, are presented in greater detail.

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.

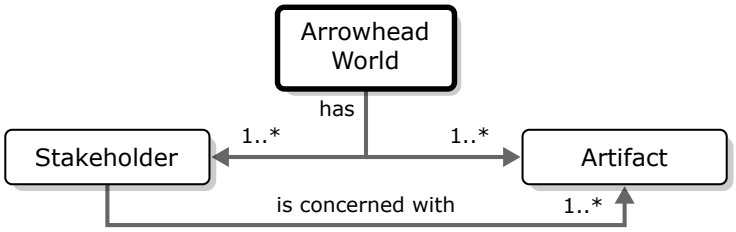

Figure 3: The two kinds of members of the Arrowhead world: stakeholders and artifacts.

[1] At the time of writing, the best source of assumptions, values and practices of the Arrowhead framework is Jerker Delsing's book *IoT Automation: Arrowhead Framework* [3]. The Eclipse Arrowhead project may publish other works of relevance in the future.

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, that are 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 or for its stakeholders. When a system or stakeholder makes use of a service, it is said to *consume* it.

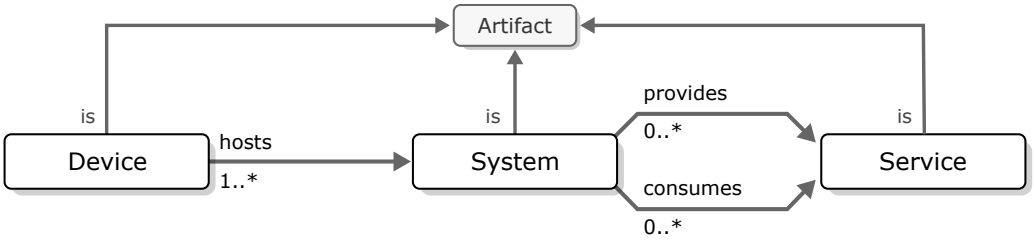

Figure 4: Hardware devices *host* software systems, which *consume* and/or *provide* services.

Every service represents one area of concern its hosting system can address. Examples of such areas of concern could be generating financial statements, replacing propellers on drones, manufacturing bolts or measuring humidity. A service providing control over a door could, for example, make it possible to check if the door is open, to open it and to close it. Each such activity of every service is represented by one service operation, which we will consider more in the next section.

2.3 Service Provision and Consumption

As we have already established, 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 to their operations, as depicted in Figure 5.

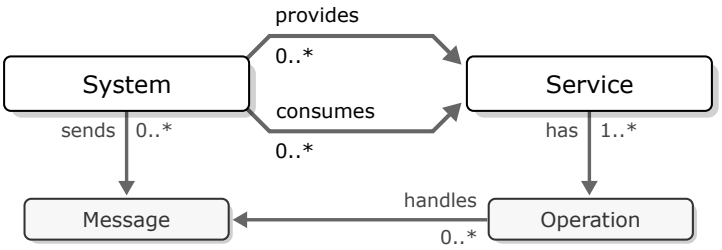

Figure 5: Systems consume services by sending messages to the providers of those services. Those providers then pass on the messages they receive to their service operations, which interpret and handle them.

Also persons can consume services, even though it is not explicitly depicted in Figure 5. This, however, requires that a human interface is attached to the providing system, or that another system with such an interface can act as proxy. The human interface enables the person, via its buttons, prompts or other elements, to send messages to the services in question, just as a regular system would.

When a providing system receives a message from a consuming system or person, it passes it on to the service operation specified in that message, as described in Sections 3.5 and 3.11. 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, sending one or more messages back to the sender, among many other possible examples.

2.4 System Composition

Systems may consume services because it is a necessary part of executing the tasks they were designed to perform. Consider, for example, a scenario in which a number of automated guided vehicles, each of which is a system, are to move items around a factory as directed by a scheduling system. As the scheduling system does not have wheels, engines or other necessary sensors and actuators, it cannot physically move any items by itself. Likewise, the individual vehicles are not capable of themselves deciding what needs to be taken to what location. However, if the scheduling system may consume the services of the vehicles, it gains the ability to physically execute its plans. When systems consume each others' services in this manner, they form a *system-of-systems*.

As depicted in Figure 6, there are different kinds of systems-of-systems with their own characteristics. There are clouds and systems-of-clouds, as well as *local* and *virtual* variants of both.[2]

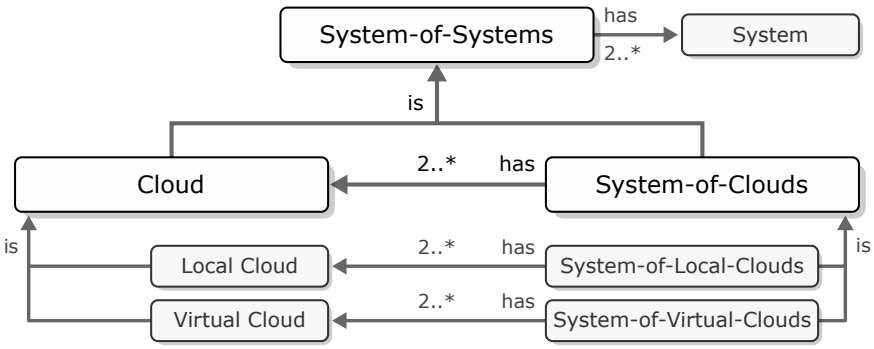

Figure 6: The types of *systems-of-systems*, each of which consists of a number of *systems* that consume each others' services. Clouds are systems-of-systems with *boundaries*, while systems-of-clouds combine multiple clouds.

A **cloud** is a set of systems separated from all other systems by at least one boundary. Such a boundary may be formed via access control policies, firewalls, gateway systems, physical separation, among many other possible examples. Additionally, infrastructure must be in place, of any kind, allowing for any system part of the cloud to send messages to any other system in that cloud.[3]

Because the Arrowhead framework is highly concerned with physical automation processes, we distinguish local clouds from virtual clouds.

A **local cloud** has at least one system that depends on being at a specific location to provide at least one of its services. Examples of local clouds could be smelting stations, drone control towers, assembly lines, power distribution centers, or the components of a satellite. All of these examples involve performing physical activities that depend on occurring at specific physical locations. Also a common data center may be a local cloud, if its exact location matters for reasons such as privacy or performance.

On the other hand, a **virtual cloud** has no systems that depend on being at a specific location to provide their services. In other words, all the resources of such a cloud are virtual. This is the kind of cloud that can be rented by many cloud providers and is part of the greater trend many refer to as *the cloud*. A virtual cloud could provide services for forecasting, analysis, design, planning or communication, among other possible examples. None of these use cases require any other resources than virtual compute, storage and network resources.

Individual clouds may be interconnected to former even larger systems-of-systems, which we then refer to as **systems-of-clouds**. The individual clouds may be owned and operated by different departments, subdivisions or teams at the same company, or even by different legal entities. Some of them may be local, while other may be virtual. Examples of systems-of-clouds may be a set of weather stations operated by the same company, the robots of distinct collaborating companies at a mining site, or the carriers of a supply chain. When a system-of-clouds contain only local clouds, we refer to it as a systems-of-local-clouds. Likewise, a system-of-clouds with only virtual clouds constitute a systems-of-virtual-clouds.

[2] In addition to being local or virtual, a cloud may also be *Arrowhead-compliant*. Such a cloud is referred to as an *Arrowhead cloud* and conforms to various architectural requirements put forth by the Eclipse Arrowhead project in a separate document.

[3] Our term *cloud* must not be confused with *the cloud*, which is a common name for renting virtual compute, storage and networking resources from a so-called *cloud provider*.

3 Concepts

With the major themes of the Arrowhead framework now established, we are ready to define its primary concepts more rigorously. To begin with, we summarize the concepts in Table 1, which presents the categories they fall into, the sections they appear in, their names, as well as summaries of their definitions. You may choose to read the table from beginning to end, use it as a means of deciding what concepts to read more about, or simply skip it if you feel the repetition is unnecessary.

Fundamental Concepts		*The foundation upon which all other primary concepts are defined.*
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.
Systemic Concepts		*The primary building blocks of Arrowhead.*
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	**Operation**	A component of a service that handles messages.
Compositional Concepts		*Significant compositions of systemic concepts.*
3.7	**System-of-Systems**	A set of systems that collaborate by consuming each others' services.
3.8	**Cloud**	A system-of-systems with a boundary and its own resources.
3.9	**System-of-Clouds**	A set of clouds that collaborate by consuming each others' services.
Communicational Concepts		*Building blocks for communication between systemic concepts.*
3.10	**Network**	A set of devices with network interfaces that are able to communicate.
3.11	**Interface**	A boundary that can be crossed by the messages of certain protocols.
3.12	**Protocol**	A description of what messages can be sent between certain interfaces.
3.13	**Message**	A description of how to invoke a certain service operation.
3.14	**Policy**	A set of constraints that must be satisfied for a message to be permitted.
3.15	**Profile**	A set of constraints added to a protocol.
Interpretational Concepts		*Constructs relevant to the formulation and interpretation of messages.*
3.16	**Encoding**	A data type used to encode and decode data.
3.17	**Semantics**	A model used to derive meaning from data.

Table 1: A summary of the primary concepts outlined in this section.

We now proceed to present the primary concepts in the same order they appear in Table 1.

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.

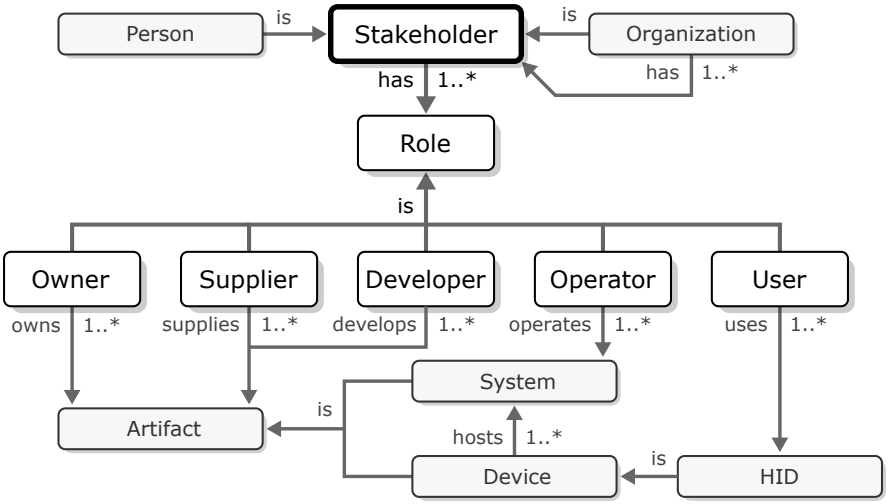

Figure 7: The stakeholder as either a person or organization, where each such stakeholder takes on one ore more distinct roles. The depicted roles are only possible examples. HID is an abbreviation for Human Interface Device.

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.

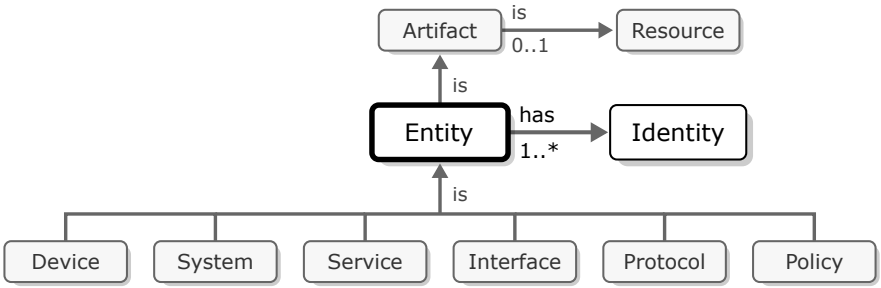

Figure 8: The entity as an artifact with an identity. An entity or artifact may or may not be considered to be a resource, in which case it is deemed to be valuable or useful from the perspective of a stakeholder. The array of artifacts with an *is*-relation to *Entity* is not complete. Other examples 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 or virtual 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 software system. Devices consist of hardware components. When a device is virtual, its hardware components are also software components. Each device must always have (1) memory, (2) compute and (3) network interfacing components, as shown in Figure 9.

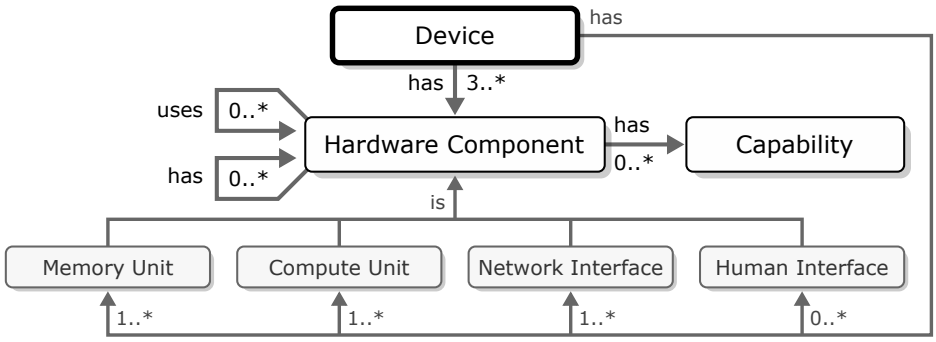

Figure 9: The device as a set of hardware components, each with automation or compute capabilities. Every device must be able to host software components using its compute and memory units, as well as communicate with other devices via its network interfaces. Devices with human interfaces are able to 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 must be considered as a hardware component. While it may seem unintuitive to consider certain machines as components, such as 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 be considered as components or not 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 and provide at least one service, as well as have at least one *system interface* through which it can send and/or receive messages for its services. If not, it must be referred to as an *opaque system*.

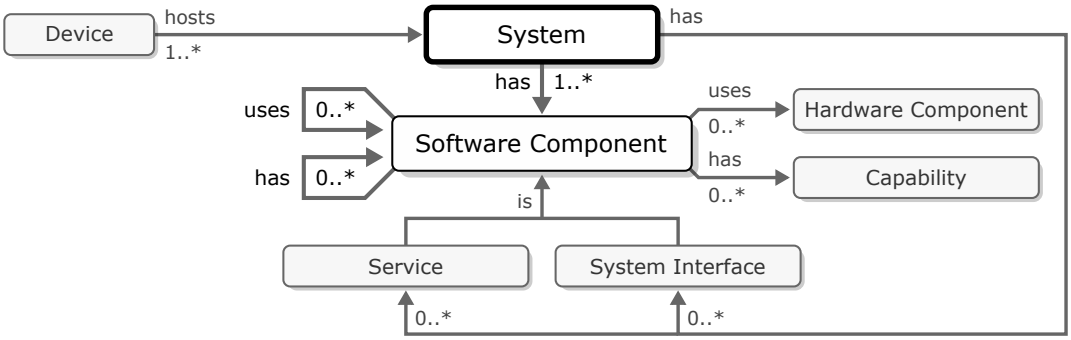

Figure 10: The system as a set of related software components, endowing a hosting device with new automation or compute capabilities. Other examples of software components could be operating systems, files, file systems, software libraries, programming language runtimes, databases and virtual machines.

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 operations, where each operation represents one set of activities the system providing the service can perform in response to a message. A provided service always exposes all of its operations. Examples of activities an operation could perform include sending messages, moving robotic arms, reading value from sensors and generating a statistical reports. For it to be possible for the service to receive messages, it must have at least one *service interface*, as depicted in Figure 11. When a message is received by a particular service interface, it must determine what operation to route the message to. This is made possible by every message being required to stating what exact operation it targets.

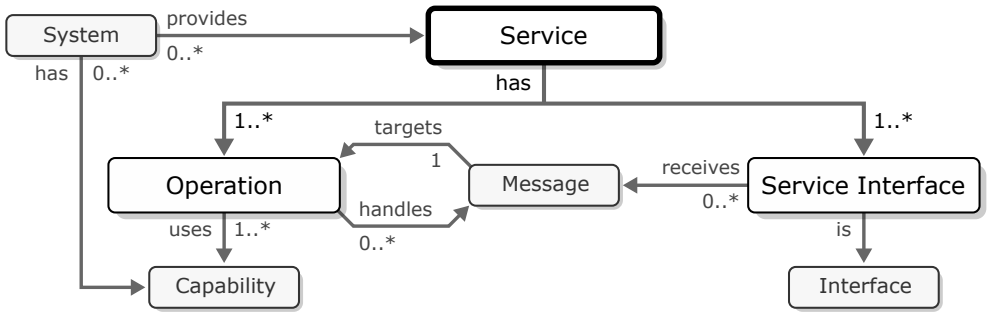

Figure 11: The service as a set of operations, making it possible for a providing system to offer the use of its capabilities to consuming systems via its service interfaces.

The primary reason why the service exists as a concept at all is to allow for related operations to be grouped together. Most significantly, such groups enables the designer of an operation to require that other, complementary, operations are also made available at the same time.

3.6 Operation

An operation is an individually identifiable component of a service that can handle given messages. Every operation is *exposed* via a service and may use any capabilities of the system providing it, as depicted in Figure 12. Operations receive messages via *operation interfaces*, which, in turn, receive them via service interfaces. If an operation needs to send messages, it may do so via the operation interfaces of service stubs representing the services it targets. How interfaces send and receive messages is described in detail in Section 3.11.

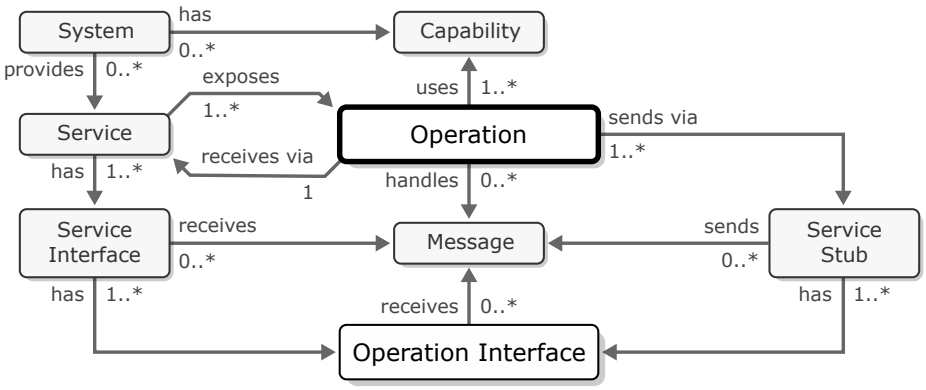

Figure 12: The operation as an entity exercising the capabilities of its system in response to messages it receives via its operation interfaces, which, in turn, receives messages via service interfaces. Operations send messages via the operation interfaces of service stubs.

While the operation is conceptually similar to a regular function in a programming language, there are important differences. Firstly, an operation accepts only one argument, which is a message of a well-defined data type. Secondly, an operation may choose to respond the messages it receives any number of times, including not at all, at any points in time.

3.7 System-of-Systems

A system-of-systems is a set of at least two systems that collaborate by at least one system consuming at least one service provided by another system in the same set. The definition is depicted in Figure 13.

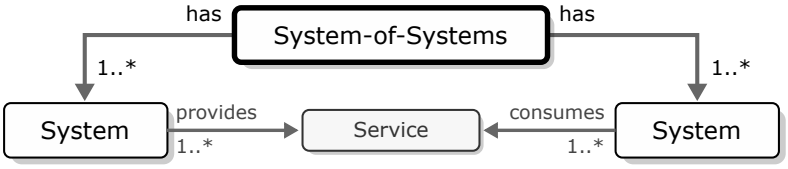

Figure 13: A system-of-systems, in which at least one system consumes the service of another.

As exemplified in Section 2.4, when a system-of-systems form, its constituent systems may become able do things collectively that none of them could do on its own. Systems-of-systems are often designed to take advantage of such emergent capabilities.

3.8 Cloud

A cloud is an identifiable system-of-systems with at least one boundary. Examples of boundaries include access control policies, firewalls and gateway or border systems. I must be possible for every system within the boundary to send messages to any other system also inside the boundary, even if the sender in question does not support any services of the receiver. This means that for a system to be considered to be added to a cloud, it must be within its boundary and be able to communicate with all other systems inside it. Clouds can be either *local* or *virtual*, depending on if the value they produce are bound to specific physical locations or not. Refer to Section 2.4 for an overview of the differences between local and virtual clouds.

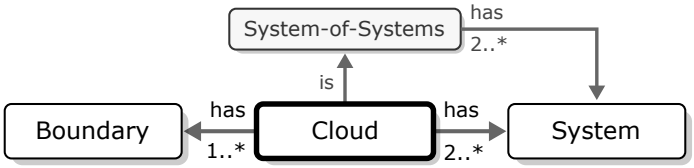

Figure 14: The cloud defined as a system-of-systems with at least one boundary.

A cloud could be engaged in manufacturing, repairs, heating, electricity distribution, workspace monitoring, drone fleet control, among many other possible kinds of physical activities. A cloud may be stationary or mobile.

3.9 System-of-Clouds

A system-of-clouds is a set of at least two clouds that collaborate by at least one cloud consuming at least one service provided by another cloud in the same set. The definition is depicted in Figure 15. It is similar to the system-of-systems, with the exception of its subsystems are clouds instead of plain systems.

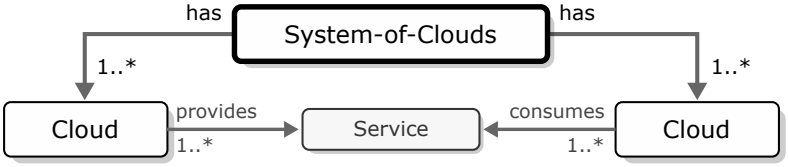

Figure 15: A system-of-clouds, in which at least one cloud consumes the service of another.

A system-of-clouds may have its own boundaries in addition to those of its constituent 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-clouds cannot have resources beyond those of its constituent clouds, however.

3.10 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 16, 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.

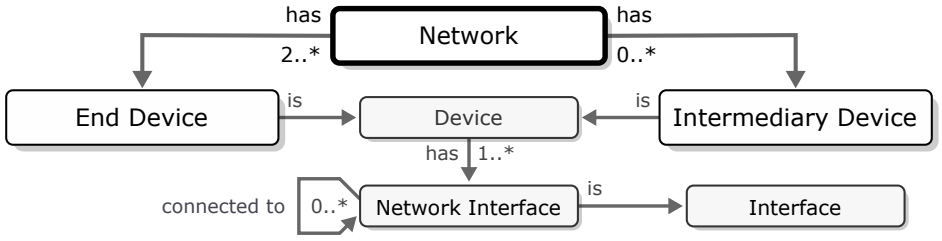

Figure 16: The network as a set of connected end devices, potentially interconnected via intermediary devices.

3.11 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. From the perspective of service provision and consumption, four types of interfaces are particularly relevant. These are (1) network interfaces, (2) system interfaces, (3) service interfaces and (4) operation interfaces, which relate as outlined in Figure 17.

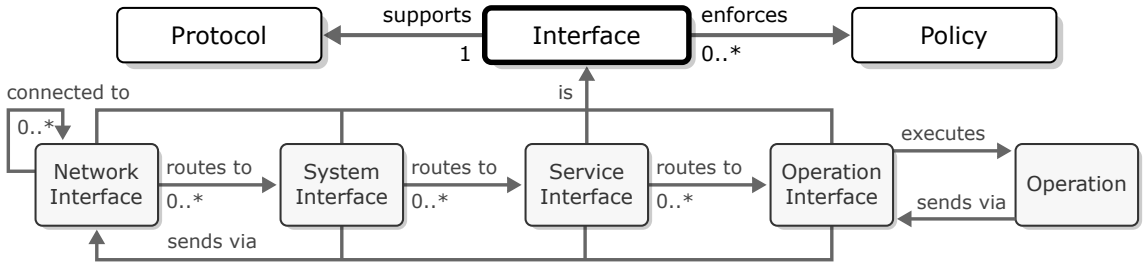

Figure 17: The interface as an implemented protocols and a set of enforced policies. Devices, systems, services and operations have their own interface types, forming four stages through which inbound messages can be routed and outbound messages sent toward the operations they target.

These four interface types form four conceptual stages, beginning with the network interface to the left and ending with the operation interface to the right.[4] When a network interface receives a message, it is *routed* rightwards through each stage until it reaches an operation. If the message is found to be invalid, forbidden at any stage, an error message may be sent back to its sender by the stage in question. The stages are conceptual in the sense that they are understood to exist even if they are not explicitly implemented. An operation receiving a message may send its own messages via the operation interfaces of service stubs, which represent other services. Those operation interfaces pass on received messages primarily to network interfaces, but may pass them on directly to system or service interfaces when the targeted operation resides on the same device or system, respectively.

As each interface only supports a single protocol, it can only pass on messages of that protocol. To make it possible for messages to pass between the four interface stages, each of the system, service and operation interfaces extends the protocols of the stage to its left. This can be thought of as each stage requiring additional details to know where to route the messages they receive. For a message to arrive at a network interface, for example, only network details are required. To route that message to a system, the device owning the network interface must know what system the message targets, and so on.

[4] For those familiar with the Internet [6] and OSI [7] network models, the *network interface* represents their *transport layers*, while the *system, service and operation* interfaces represent the *application layer* of the Internet model and the *session, presentation and application layers* of the OSI model. The system and service interfaces do not correspond directly to the OSI session and presentation layers.

3.12 Protocol

A protocol is an identifiable set of message and state types, used when formulating and interpreting messages. The *message types* describe what data messages must contain, while the *state types* describe when received message are acceptable in relation to states protocol implementations are expected to maintain. As shown in Figure 18, a protocol may be defined as an extension of another protocol, conform to certain profiles and use certain encodings. Profiles add constraints to protocols, such as requiring that certain metadata by added to messages, while encodings are used to encode and decode messages.

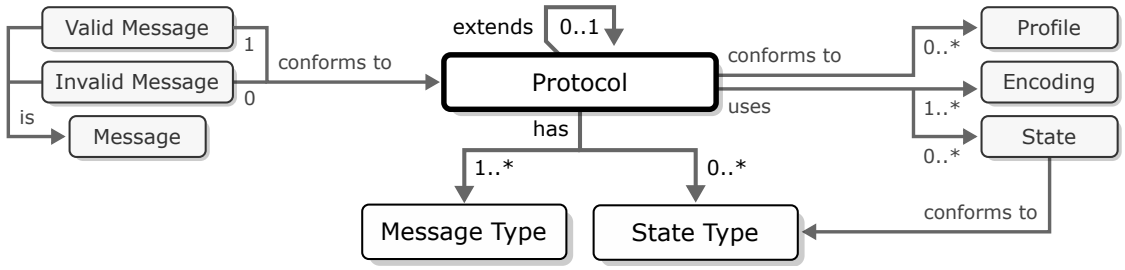

Figure 18: The protocol as set of message and state types, conforming to certain profiles and using certain encodings.

Protocols are exclusively concerned with if given messages are valid or invalid. A valid message can be correctly interpreted and acted upon by the implementation of the protocol, while an invalid message cannot. Invalid messages do not conform to all of the encodings, profiles and state types of the protocol. Protocols are *never* concerned with what messages are *permitted*, which rather is the concern of the policy.

States can be used to account for scenarios where the context in which a message is received impacts how it must be interpreted. Consider, for example, a scenario where a service is provided for opening and closing a door. Let us assume that the door is currently open, and a message is received by the service that instructs it to open it. As the door is already open, it cannot be opened any more than it already is. The message could, therefore, either be interpreted as being already handled or be rejected, depending on what the use case makes most relevant. If the protocol of the message would not have accounted for the state of the door, open and close messages would have to always be accepted.

3.13 Message

A message is data that identifies an operation, among any other metadata, any may include a payload, as depicted in Figure 19. In other words, a message describes how it is to be sent to a particular operation, made available by a particular service, system and device, as well as including any payload necessary for that operation to be able to execute as intended.

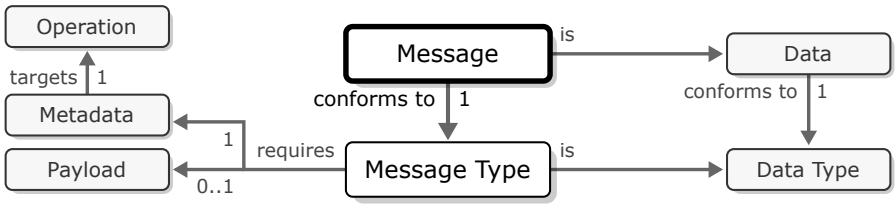

Figure 19: The message as data that must include metadata targeting an operation, as well as a payload.

The metadata of a message specify the details required for it to arrive at the operation it targets, as well as additional details required to interpret the message and its payload correctly, such as by stating what encodings are used. The metadata may also include details required to satisfy 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.

3.14 Policy

A policy is an identifiable set of constraints, useful for determining if given messages are permitted or forbidden, as depicted in Figure 20. Policies may be concerned with authorization, contracts, economic goals, and so on.

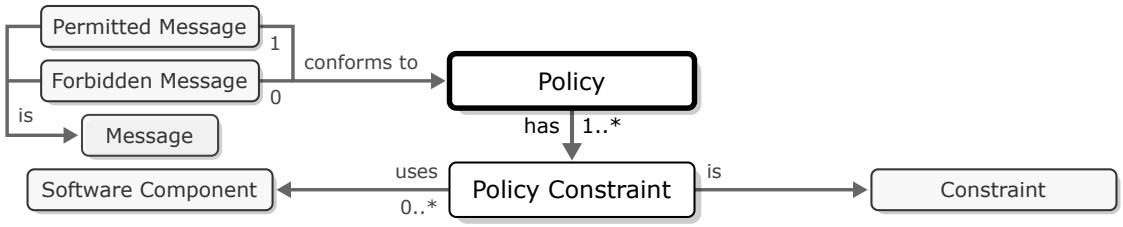

Figure 20: The policy as a set of constraints, useful for determining if messages are permitted or not.

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 3.11. 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.

3.15 Profile

A profile is a set of constraints that can be added to a protocol. A profile may require that certain flags, headers, or other metadata, are included in the messages of the protocols that conform to it, among other possible examples. While a protocol may only extend up to one other protocol, it may conform to any number of profiles. The constraints of a profile must be based on a protocol extended by every protocol conforming to that profile, as we show in Figure 21.

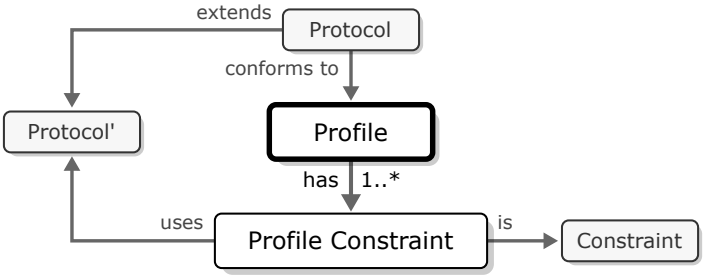

Figure 21: The profile as a set of protocol constraints, which may, for example, be concerned with protocols or encodings. *Protocol'* represents any protocol that *Protocol* could extend, directly or by any extended protocol.

While perhaps a bit difficult to grasp initially, unless this was the case there would not be anything to base the constraints on. Consider, for example, a service with an interface that supports a custom extension of the HTTP protocol [8]. Such an extended HTTP protocol could, among other things, specify how one would formulate request lines to target the operations of the service. An example of such a request line could be `GET /pump-7b/pressures/a14 HTTP/1.1`, which we can imagine targets an operation that fetches a pressure value from a pump. If our 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 of the service interface. As HTTP extends TCP [9] and IP [10], our constraints may also target details of those protocols. An example profile could require that certain HTTP headers be included in every message, or that certain TCP flags not be used, among many other possible examples.

3.16 Encoding

An *encoding* is a data type that makes 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. How encodings, encoders and decoders relate is depicted in Figure 22.

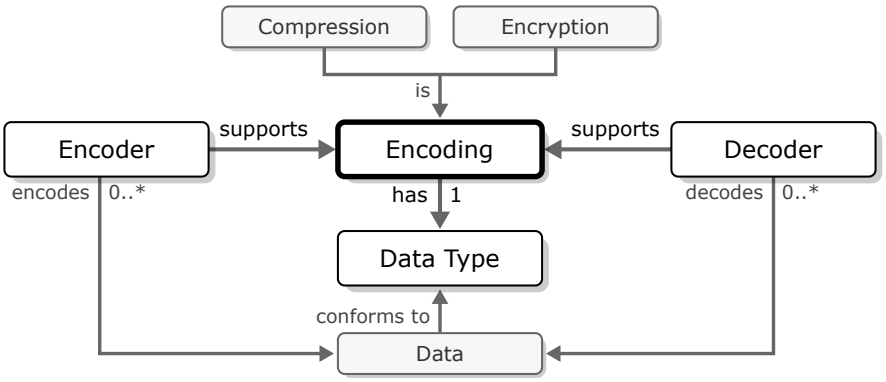

Figure 22: The encoding as a data type, supported by certain encoders and decoders. The compression and the encryption are treated as special kinds of encodings.

An encoding useful for *transmission* could, for example, be JSON [11],[5] while an encoding useful for *storage* could be some kind of file or database format. An encoding useful for *processing* could be the binary format employed by a compute unit, virtual machine or computer language runtime, among other possible examples.

There are two kinds of encodings that deserve special attention. These are *compressions* and *encryptions*. While perhaps not often treated as if being encodings, they do fit the description of encoding we have just presented. They transform data, often treated as plain byte arrays irrespective of their original structures, from a form suitable for processing into another form suitable for transmission and/or storage, and vice versa. However, they differ from other encodings in that the transmission and storage form of the data is expected to gain special properties, namely to require less space or become undecipherable without knowledge of specific secrets.

3.16.1 Compression

A compression is an encoding used to make data representable using less datums during transmission or storage, which can improve performance by requiring less transmission time or storage space. Once compressed, the data typically cannot be interpreted in any meaningful way. To use the data again, it must first be decompressed, which restores it to its original form, or some approximation of its original form. Compressions that can restore data to their original forms are often referred to as being *lossless*, while those that deal with approximations are referred to as being *lossy*. Lossy compressions are typically used for images, audio, video and other media, while lossless compression is typically used for messages.

3.16.2 Encryption

An encryption is an encoding used to turn data into a form that cannot be interpreted by a third party during its transmission or storage. Once encrypted, there should be no practically computable means of reverting, or decrypting, the data to its original form. To make it possible for the owner of the data, or its intended receiver, to revert it back to its original form, some form of secret numbers, passwords or other keys are typically used.

Encryption is the means through which systems-of-systems can be made secure. It facilitates security in the sense that it makes it practically possible to guarantee the privacy, integrity and authenticity of transmitted messages and other stored data.

[5]The data type of the JSON encoding is a union type of seven variants: *object*, *array*, *string*, *number*, *true*, *false* and *null*. Other encodings provide other data types. The data type of XML [12] is, for example, its *document* type, which in turn uses *elements*, *texts* and other types.

3.17 Semantics

A semantics is a model used to derive meaning from data. More specifically, it *disambiguates*, or makes the interpretation clear, of data conforming to any out of one or more data types, as illustrated in Figure 23. All data that are known to conform to any of these data types then become possible to interpret. In other words, unless you are able to associate a given data type with a semantics, you cannot understand or act on any data conforming to the type.

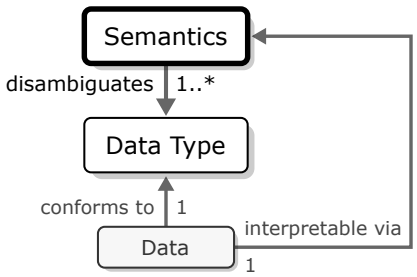

Figure 23: The semantics as a model used to disambiguate data types, making it possible to interpret data conforming to those types.

All data types disambiguated by the same semantics must express the same intrinsic meaning. Consider, for example, the JSON [11] object `{"v":21.30}` and the XML [12] element `<rd temp=70.34/>`. Both of these are data, both of them conform to a data type, and both of those data types conform to the same semantics. The JSON object represents a temperature value in celsius and the XML element represents the same temperature expressed in fahrenheit. Despite having the same semantics, however, their structures are quite different.

3.17.1 Semantics Profile

When a semantics disambiguates two or more data types, it can be used to construct a semantic profile. Such a profile requires that the every conformant protocol has a relevant encoding whose data type is disambiguated by a certain semantics.

A common application of semantic profiles is to make different protocols express the same meanings, even though they use different encodings. For example, a certain protocol may be based on HTTP [8] and JSON [11], while another is based on CoAP [13] and CBOR [14]. If the payloads of both protocols would conform to the same semantic profile, the messages of both protocols would express the same intents and meanings despite being structured differently.

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 should be represented by the name it is given here.
   (a) If not possible or desirable for the names to be the same, it is enough for the concept in the derived work to unambiguously refer to the corresponding concept in this work.
   (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 24.

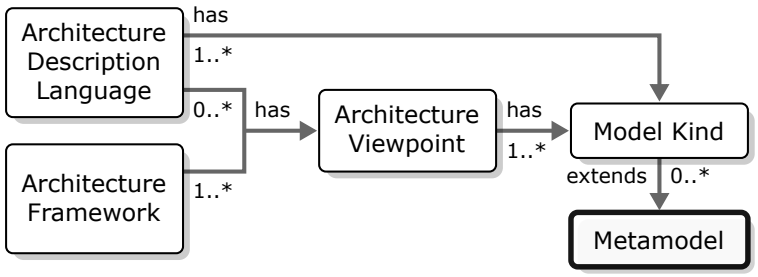

Figure 24: The metamodel as a part of an ISO42010 model kind, which in turn may be referenced by architecture description languages and architecture frameworks.

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[6].

[6] At the time of writing (2023-02-17), 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. The works in question are introduced in Section 1.4. 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 designs or specifies Arrowhead systems or systems-of-systems, or who extends 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 (1) goals, ambitions and other design 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.

Assumption

The taking for granted that some fact or statement is true.

Note 1 Assumptions can be an important method of imposing delimitation on a model or framework. If, for example, assuming that a suitable means of secure communication will exist, a model of communication can be formulated without having to specify how its messages are secured.

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. We do only, however, explicitly define one way in which capabilities can be exercised, namely via the operations exposed by services. The two definitions may, therefore, be considered practically equivalent. See also Capability, System.

Capability, Device

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

Capability, Emergent

A capability facilitated by the formation of a system-of-systems. See Section 3.7.

Capability, System

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

Cloud

A cloud is an identifiable system-of-systems with at least one boundary, dedicated to producing or acting on a significant resource. See Section 3.8.

Cloud, Arrowhead

A cloud that conforms to a set of architectural requirements specified by the Eclipse Arrowhead project.

Note 1 These requirements are outside the scope of this document and are, as a consequence, not specified here. Please refer to the architectural documentation of the Eclipse Arrowhead project for more information Arrowhead-compliant clouds.

Cloud, Local

A cloud bound to a physical location due to its acting on or producing local resources. See Section 3.8.

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 architectural rather than conceptual, which means that only some of the aspects they cover are inside the scope of this document. 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. See Section 3.8.

Cloud, Virtual

A cloud unbound by physical location by only acting on or producing virtual resources. See Section 3.8.

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 not be used to refer to a system being a constituent of a system-of-systems. Such a system should rather be referred to as being a subsystem.

RAMI4.0 makes no practical distinction between components and systems. We approach something akin to a practical distinction by only defining devices and systems as having components. We do not, however, forbid other, more generic, uses of the word.

Component, Hardware

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

Component, Software

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

Component, Virtual Hardware

A software component that represents what normally would be a hardware component.

Compress

To encode data from its original form to a space-efficient form. See Section 3.16.1.

Note 1 Comressing is the reverse of decompressing.

Compression *(noun)*

An encoding that describes a space-efficient data format. See Compress, Decompress, Section 3.16.1.

Concept

The description of a generic or abstract idea.

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

An 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, Policy

A constraint imposed by a policy. See Section 3.14.

Constraint,Profile

A constraint imposed by a profile. See Section 3.15.

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 consuming a service via a human interface.

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 or a stakeholder rather than just any entity, as well as (2) that the capabilities of the consumed service be exercised via an operation. Our definitions should be considered as compatible but more specific.

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 could, for example, indicate that the drill should spin at $4 * 100 = 400$ rotations per minute and be lowered $13 * 5 = 65$ millimeters.

Note 2 Both the type and semantics of given data must be known for the data to be possible to interpret.

Datum

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

Note 1 An example of a datum familiar to many could 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 person interpreting data.

Decoder

An component capable of decoding data.

Decompress

To decode data from a space-efficient form to its original form. See Section 3.16.1.

Note 1 Decomressing is the reverse of compressing.

Decrypt

To decode data from an undecipherable form to its original form. See Section 3.16.2.

Note 1 Decrypting is the reverse of encrypting.

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.

Device, Virtual

A device that exists only virtually. Examples of virtual devices can be application containers, virtual machines or emulated machines.

Domain, Problem

All aspects, known and unknown, that influence the potential for solving a particular problem.

> **Note 1** When you *address* a problem domain, you provide a solution to its problem that accounts for all of its known aspects.

> **Note 2** The problem domain of the Arrowhead framework consists of all aspects, known and unknown, that influence the potential for computer systems to exchange the information they need to execute tasks they have been assigned. Addressing its problem domain means that software specifications or implementations are produced that describe or handle such information exchanges, respectively.

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 person recording data.

Encoder

An component capable of encoding data.

Encoding *(noun)*

A data type used to structure and interpret conformant data. See Section 3.16.

Encrypt

To encode data from its original form to an undecipherable form. See Section 3.16.2.

> **Note 1** Encrypting is the reverse of decrypting.

Encryption *(noun)*

An encoding that describes an undecipherable data format used to hide data from others that its intended receivers. See Encrypt, Decrypt, Section 3.16.2.

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” [15]. 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 or both of the framework of ideas and the framework of software maintained by the Arrowhead project. See Framework, Section 2.

Framework, Idea

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

Function

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

> **Note 1** Most functions can be practically implemented as software, in which case they can also be referred to as computer functions.

Function, Computer

A sequence of instructions executed by one or more compute units in response to an invocation with a set of arguments. The result of executing such a function may be that the state of the computer is updated and/or that the function returns a value. Compare with Function.

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* or *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.

Note 1 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.

Implementation, Hardware

An implementation comprised of hardware artifacts.

Industry 4.0

The anticipated 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 3.11.

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. See Sections 3.5, 3.11.

> **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 describing how to invoke a certain operation. See Section 3.13.

Message, Error

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

Message, Forbidden

A message that fails to satisfy a policy of concern and, therefore, will not be passed to the operation it targets. See Section 3.14.

Message, Inbound

A message that it currently being routed away from a network interface and towards an operation. See Section 3.11.

Message, Invalid

A message that fails to satisfy a protocol of concern and, therefore, will not be passed to the operation it targets. See Section 3.12.

Message, Outbound

A message that it currently being sent towards a network interface as part of its journey to an operation. See Section 3.11.

Message, Permitted

A message that does satisfy a policy of concern and, therefore, will be passed to the operation it tagets—if all other policies are also satisfied. See Section 3.14.

Message, Valid

A message that does satisfy a protocol of concern and, therefore, will be passed to the operation it targets—if it is permitted. See Section 3.12.

Metadata

Data describing other data.

Metamodel

A set of model constructs that can be used to formulate 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 3.10.

Operation

A component of a service that handles given messages by using the capabilities of its system. See Section 3.6.

Operation, Exposed

An operation part of a service that is currently being provided by a system. See Section 3.6.

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 that must be satisfied by all messages passed on by a conforming interface. See Section 3.14.

> **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.

Practice

A particular way in which some task or activity is carried out.

> **Note 1** The Arrowhead framework is, among other things, concerned with shaping practices surrounding the development and maintenance of systems-of-systems.

Profile

A set of constraints superimposed on a protocol. See Section 3.15.

> **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.

Profile, Semantic

A profile requiring that the data type of each relevant protocol encoding is disambiguated by one specific semantics. See Section 3.17.1.

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 3.12.

> **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 3.12.

Protocol, Operation

A protocol implemented by an operation interface. See Section 3.12.

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

Protocol, Service

A protocol implemented by a service interface. See Section 3.12.

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

Protocol, System

A protocol implemented by a system interface. See Section 3.12.

> **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.

Proxy

An entity representing the agenda or desires of another entity or stakeholder.

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 away from a network interface towards the service operation it targets.

Routing, Message

Prefer Routing.

Semantics

A model used to derive meaning from data. Data wihout a known semantics cannot be interpreted. See Section 3.17.

Semantics, Data

Prefer Semantics.

Semantics, Message

The semantics of a message.

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 3.12.

Stub, Service

The means through which a system consumes a service of another system. Every service stub has its own operation interfaces. If one of those operation interfaces is provided with a message, the service stub will attempt to pass on that message to the service represented by the stub. 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-Clouds (SoCl)

A set of at least two clouds that collaborate by at least one cloud consuming at least one service provided by another cloud in the set. See Section 3.9.

System-of-Local-Clouds (SoLC)

A system-of-clouds where every constituent cloud is a local cloud. See Section 3.9.

System-of-Systems (SoS)

A set of at least two systems that collaborate by at least one system consuming at least one service provided by another system in the set. See Section 3.7.

> **IoTA:AF** defines a system-of-systems as “a set of systems, which [...] exchange information by means of services”. Our definition must be interpreted as being equivalent or as being incorrect.

System-of-Virtual-Clouds (SoVC)

A system-of-clouds where every constituent cloud is a virtual cloud. See Section 3.9.

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 than its own, or a system using its robot arm to collect and handle raw materials.

Type

A description of how datums are to be arranged to encode certain structures. 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. Formally, the contextual knowledge of a data type is represented by its semantics.

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.

Value

Something, such as a principle or quality, assumed to be intrinsically desirable.

> **Note 1** In the context of a technical framework, values capture priorities assumed to be had be the users of the framework. Values could concern the importance of being able to meet realtime deadlines, technical complexity, material costs, and so on.

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.

Virtual

The property of having ones existence maintained by a computer.

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 2023-02-17.

[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 2023-02-17.

[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] R. Braden. “Requirements for Internet Hosts – Communication Layers”. RFC 1122, October 1989. doi:10.17487/RFC1122.

[7] “Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model”. *International Standard 7491-1:1994*, ISO/IEC/IEEE, November 1994.

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

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

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

[12] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau. “Extensible Markup Language (XML) 1.0 (Fifth Edition)”. *Technical report*, ISO/IEC/IEEE, 2008.

[13] Z. Shelby, K. Hartke and C. Bormann. “The Constrained Application Protocol (CoAP)”. RFC 7252, June 2014. doi:10.17487/RFC7252.

[14] C. Bormann and P. Hoffman. “Concise Binary Object Representation (CBOR)”. RFC 8949, December 2020. doi:10.17487/RFC8949.

[15] Merriam-Webster. “Entity”. URL `https://merriam-webster.com/dictionary/entity`. Accessed 2023-02-17.

7 Revision History

7.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2022-01-01	0.1	Initial proposal.	Emanuel Palm
2	2022-02-11	0.2	Made minor improvements to various figures and the Overview and Concepts sections.	Emanuel Palm
3	2023-02-18	0.3	Revised and extended the Overview and Concepts sections in response to Sinetiq review.	Emanuel Palm

7.2 Quality Assurance

No.	Date	Version	Approved by
1			