CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

The Arrowhead Approach for SOA Application Development and Documentation

Fredrik Blomstedt
Luis Lino Ferreira*
Markus Klisics
Christos Chrysoulas*
Iker Martinez de Soria
Brice Morin
Anatolijs Zabasta
Jens Eliasson
Mats Johansson
Pal Varg

*CISTER Research Center

CISTER-TR-141101

Version:

2014-10-29

The Arrowhead Approach for SOA Application Development and Documentation

Fredrik Blomstedt, Luis Lino Ferreira*, Markus Klisics, Christos Chrysoulas*, Iker Martinez de Soria, Brice Morin, Anatolijs Zabasta, Jens Eliasson, Mats Johansson, Pal Varg

*CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: llf@isep.ipp.pt, chech@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

The Arrowhead project aims to address thetechnical and applicative issues associated with cooperativeautomation based on Service Oriented Architectures. Theproblems of developing such kind of systems are mainly due to thelack of adequate development and service documentationmethodologies, which would ease the burden of reusing services ondifferent applications. The Arrowhead project proposes atechnical framework to efficiently support the development ofsuch systems, which includes several tools for documentation ofservices and to support the development of SOA-basedinstallations. The work presented in this paper describes theapproach which has been developed for the first generation pilotsto support the documentation of their structural services. Eachservice, system and system-of-systems within the ArrowheadFramework must be documented and described in such way thatit can be implemented, tested and deployed in an interoperableway. This paper presents the first steps of realizing the Arrowheadvision for interoperable services, systems and systems-of-systems.

The Arrowhead Approach for SOA Application Development and Documentation

Fredrik Blomstedt[1], Luis Lino Ferreira[2], Markus Klisics[1], Christos Chrysoulas[2], Iker Martinez de Soria[3], Brice Morin[4], Anatolijs Zabasta[5], Jens Eliasson[6], Mats Johansson[6] and Pal Varga[7]

[1]BnearIT AB, Luleå, Sweden, [2]CISTER/INESC TEC, ISEP, Porto, Portugal, [3]TECNALIA Research & Innovation, Bilbao, Spain, [4]SINTEF ICT, Oslo, Norway, [5]Riga Technical University, Riga, Latvia, [6]Luleå University of Technology, Luleå, Sweden, [7]Budapest University of Technology and Economics, Budapest, Hungary

***Abstract* — The Arrowhead project aims to address the technical and applicative issues associated with cooperative automation based on Service Oriented Architectures. The problems of developing such kind of systems are mainly due to the lack of adequate development and service documentation methodologies, which would ease the burden of reusing services on different applications. The Arrowhead project proposes a technical framework to efficiently support the development of such systems, which includes several tools for documentation of services and to support the development of SOA-based installations. The work presented in this paper describes the approach which has been developed for the first generation pilots to support the documentation of their structural services. Each service, system and system-of-systems within the Arrowhead Framework must be documented and described in such way that it can be implemented, tested and deployed in an interoperable way. This paper presents the first steps of realizing the Arrowhead vision for interoperable services, systems and systems-of-systems.**

Keywords—System-of-Systems; Systems; Services; Service-Oriented Architectures; Documentation Templates

I. INTRODUCTION

The term Service-Oriented Architecture (SOA) has gained a lot of interest from the academic world as well as from the industry. SOA is a distributed architectural design pattern with loosely coupled interfaces [1]. A service is the core building block of SOA, and is basically a software application performing some task, with a formal interface described using a standard description framework (shown in Fig. 1). For Web services, Web Service Description Language (WSDL) and Web Application Description Language (WADL) are normally used. A service must hold certain properties, such as being discoverable, composable and loosely coupled from any operating system, programming language and other services. Normally, the use of SOAs is driven by certain properties and requirements, as discussed by Erl in [1]. Some of the most important trade-offs when comparing SOA to legacy client-server models of communicating systems are, as stated in the SOA manifesto:

- Business value over technical strategy;
- Intrinsic interoperability over custom integration;
- Flexibility over optimization.

The use of SOA is driven by a need for re-use of code and systems over the use of specialized building blocks, optimized for a certain application or scenario. A service should hide any internal logic, which could be implemented using any (suitable) programming language on any (suitable) operating system. A service should be loosely coupled, with no predefined connections. Existing services can also be composed into new services using either Service Orchestration or Choreography.

Fig. 1. Service-Oriented Architecture overview

However, even if the use of SOA is a very powerful development paradigm, the use of a potentially large number of functionalities distributed over many different systems, over different protocols and implemented in several programming languages imposes certain problems when it comes to document how services and systems should be described and their interactions maintained.

The Arrowhead project targets five business domains; Production (process and manufacturing), Smart Buildings and infrastructures, Electro mobility, Energy production and Virtual Markets of Energy. In these domains there is a number of technological architectures used for implementing SOA-based solutions. One of the grand challenges of Arrowhead is to enable interoperability between systems that are natively based on different technologies. One main objective is to achieve that, thus keeping the advantages of SOA, e.g., the flexibility obtained by the loose coupling. The strategy to meet that objective is to identify what are the least common denominators needed and select the most suitable common solutions. In the process, four central SOA questions for guiding the work have been identified: i) How does a system that is a service provider make its services known to service consumer?; ii) How does a system that is a service consumer discovers services it wants to consume?; iii) How does a system that is a service provider decides if a system that wants to consume its services, is authorized to do that?; iv) How to orchestrate system of systems, i.e. enabling an orchestration body to control which of the provided service instances a system shall consume? The answers to these questions are collected under the generic label of the Arrowhead Framework, which is briefly described in Section III.

The Arrowhead Framework contains common solutions for the core functionality in the area of Information Infrastructure, Systems Management and Information Assurance as well as the specification for the application services carrying information vital for the process being automated.

A developer needs to know how to develop, deploy, maintain and manage Arrowhead compliant systems. Therefore, it is crucial that there are common understandings of how Services, Systems and System-of-Systems are defined and described. To address these issues, the framework also includes design patterns, documentation templates and guidelines that aim at helping systems, newly developed or legacy, to conform to Arrowhead Framework specifications. This paper focus is on the documentation and development methodology part of the framework.

This paper is structured as follow; Section II presents related work in the area. Section III provides an introduction to the Arrowhead Framework and the proposed SOA-based architectural meta-model. Section IV presents System-of-Systems level description, followed by Sections V and VI which present System and Service level descriptions, respectively. Section VII presents the proposed service repository. Finally, conclusions and suggestions for future work are presented in Section VIII.

II. Related Work

Shaw and Garlan [2] provide a simple definition of what software architecture really stands for: "The architecture of a software system defines the system in terms of computational components and interactions among those components". Bass et al [3] provide a summary of what is needed for documenting a software architecture. They identify three motivating factors, which are to enable communication of the software architecture among stakeholders, to capture early design decisions, and to provide re-usable abstractions of software systems.

The IEEE 42010-2011- ISO/IEC/IEEE, "Systems and software engineering - Architecture description" standard [4], includes a conceptual framework to support the description of architectures, and the required content of an architectural description. The standard is developed from a consensus of current practices. It includes the use of multiple views, reusable models within views, and the relationship of architecture to the system context. It defines the main notions such as Architectural Description, Stakeholders, Concerns, Views, Viewpoints and Models. The majority of contemporary works utilize concepts and approaches defined in the mentioned IEEE standard.

In [5] May, described a framework for comparing viewpoint models. The analysis of the models was accomplished by comparing each of their viewpoints against the framework reference lists. By identifying the framework elements related to each viewpoint, it is able to summarize the framework coverage by viewpoint model and compare them by the different elements they address. The viewpoints from different models, when combined into an optimum set, can provide greater coverage of the software architecture domain than any of the individual viewpoint models.

Some authors prescribe a fixed set of views with which to engineer and communicate an architecture. For example, Kruchten [6] described the "4+1" view model, which consists of multiple, concurrent views that allow the different stakeholder concerns to be addressed separately. The model includes details of how the views are related and a process for developing the complete set of views. The architect can then identify the key abstractions from the problem domain and model these in the Logical View. Furthermore, the views of the "4+1"view model conform reasonably well to the viewpoints of the IEEE 42010-2011.

Rational Architectural Description Specification (ADS) [10] expands the "4+1" model to enable the description of more complex architectures, such as enterprise, e-business, embedded systems and non-software systems. It features a formal definition of requirements evolution and architecture testability, and utilizes the Unified Modelling Language (UML) notation where possible.

The Views and Beyond (V&B) model of software architecture documentation has been proposed in [7-8]. The process of documenting the architecture prescribes documenting the relevant views and any additional information that corresponds to more than one view. Some views are too complex to show in one representation, so they can be broken down into a number of view packets. V&B model includes a template for the contents of a view packet, therefore it complies with the IEEE 42010-2011.

Siemens Four View Model [9] is the outcome of a study into the industrial practices of software architecture. The authors come to the conclusion that the structures used to design and document software architecture fall into four broad categories, namely conceptual, module, execution and code structures. Each category addresses different stakeholder concerns. The Siemens four view model ignores the explicit concerns of the stakeholders, other than the software architect. However, the other stakeholders may be addressed implicitly by the concerns satisfied by each of the views. This reflects the focus of the model on the architect's design approach and not on the documentation for communication.

The model proposed in by Gross [11] (namely COBRA) was developed with the purpose of flexible composition and reuse of software artifacts is inspired by object-oriented and component-based methods. The method uses UML as primary model-based notation for all analysis and design activities. The model understands a System or a System-of-Systems as a component that can interact with others through interfaces and can be decomposed in other Systems or components. The model implementation is founded in three modelling activities: context realization, component specification and component realization.

Arrowhead architecture approach in some way utilizes Cobra model approach, but it goes beyond of it. The required architecture has to make significant emphasis on the variety of stakeholders and their needs, due to specific of the System-of-Systems, which to be able seamlessly consolidate projected, existent and legacy systems.

III. The Arrowhead Framework

The Arrowhead Framework consists of what is needed for anyone to design, implement and deploy an Arrowhead compliant system. The Arrowhead Framework aims at enabling

all of its users to work in a common and unified approach – leading towards high levels of interoperability.

A. *Overview of Arrowhead Framework*

The Arrowhead Framework includes principles on how to design SOA-based systems, guidelines for its documentation and a software framework capable of supporting its implementations.

The design guidelines provide generic "black box" design patterns on how to implement application systems to be Arrowhead Framework compliant. Furthermore, these guidelines allow making legacy systems Arrowhead Framework compliant.

The documentation guidelines include templates for service, system and, system-of-systems descriptions (to be detailed in the following sections of this paper). Due to its complexity there is also a "Cookbook" for hands-on instructions on how to use the framework.

The software framework (Fig. 2) includes a set of Core Services which are capable of supporting the interaction between Application Services. The Core Services handle the support functionality within the Arrowhead Framework to enable Application Services to exchange information. Examples are services for Discovery, Authorization, Orchestration, and System Status. An Application Service handles the data exchange between specialized devices (those that the system is special at). Examples are services for sensor reading, billing, energy consumption, weather forecasts, etc.

The Core Services (Fig. 2) are further divided into three different groups: i) Information Infrastructure (II); ii) Systems Management (SM); and, iii) Information Assurance (IA).

The II services are the set of core services and systems in charge of providing information about the services and how to connect to them. This includes services like Service Discovery, Application Installation and Setup, Service Metadata, etc. The SM services are the set of core services and systems providing support for late binding and solving system-of-systems composition. The SM provides logging, monitoring and status functionality. It also addresses orchestration, software distribution, Quality of Service (QoS), configuration and policy. Finally, such a software framework can only operate if the system is able provide adequate security and safety levels. Those functions are assured by the IA services, supporting secure information exchange. Example services include those for authorization, authentication, certificate distribution, security logging and service intrusion.

The software framework also addresses the design and prototype implementations of gateways/mediators for making legacy systems Arrowhead compliant.

Finally, the Arrowhead Framework provides a set of rules and principles to: i) address technical property requirements; ii) Arrowhead conformity requirements and, iii) a set of tool(s) for conformity test and verification.

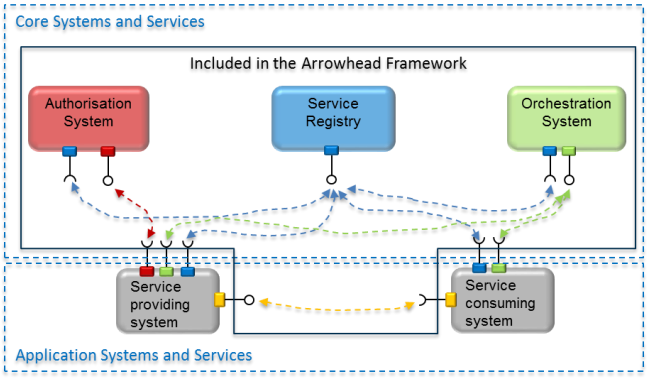

Fig. 2. Arrowhead Framework core and application services

B. *The Arrowhead Framework documentation approach*

The Arrowhead Framework states a common approach of how to document SOA-based systems. The documents structure is built on three levels, namely: System-of-Systems, System and Service level. These are depicted in Fig. 3, showing the links between documents, as well.

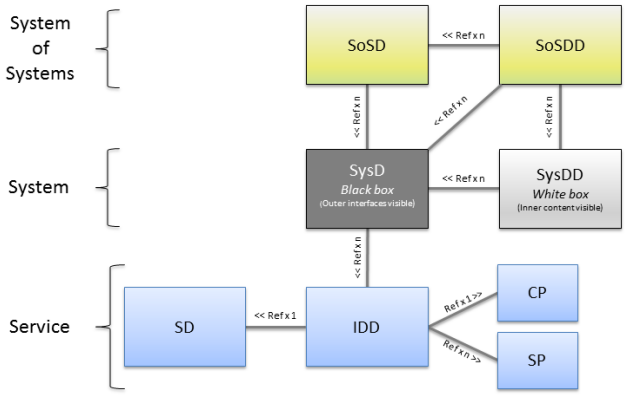

Fig. 3. The Arrowhead Framework documentation relationships

The approach is to apply the terms "black box" and "white box" only to the System level since it is well known what it means. However, these concepts are not used at the Service level, where such division is rather meant to be about technology independence/dependence.

At the System-of-Systems (SoS) level, a concrete "System-of-Systems type" is defined in the System-of-Systems Description (SoSD) document. Thus, the particular "system type" needed to fulfill our SoS goals can be implemented. The correct way of working is assured thanks to the "black box" representation of all Systems in the System Description (SysD). Therefore, each "system type" can talk to each other or identify the gateways/mediators' needs.

In the System-of-Systems Design Description (SoSDD) a SoSD instance can be created. All the participating "white box" Systems (SysDD) must be enumerated and the entire setup must be explained, including infrastructure description (network configuration, VPNs, etc.), domain structure, startup behavior etc. This is a deployment description and it describes the SOA installations.

In the System level, a proper "black box" description of Systems in the SysD document is specified. This entails a listing of all the provided/consumed services with references to their Interface Design Description (IDD). This approach enables the general description of all system interfaces without actually knowing the exact implementation/solution. "System types" can be described this way, correctly fitting to the definition of "black box" design.

In the System Design Description (SysDD) document, the solution (i.e. "white box") that fulfills one or more SysDs is remarked. This actually means that one "white box" design can implement more than one "system type". For example, a solution can be a product that fulfills both a Temperature System and a Heat Meter.

At the Service level, the IDD describes how to realize the service identifying explicitly the technologies to be used. This document contains pointers to the Communication Profile (CP), the Semantic Profile (SP) and the Service Description (SD). The CP is composed by the transfer protocol (e.g., CoAP), security for the transportation of data (e.g., DTLS) and the data format (e.g., EXI). The SP contains information about how a specific measurement (e.g., a temperature value or a pressure value) is coded (e.g., in a standard such as XML or SenML). Furthermore, the SD includes the abstract information model, functions and interfaces in order to describe an abstract architecture of the service.

System-of-Systems and the System layers have high level definitions. In order to make them more easy-to-understand, use-cases, sequence diagrams and several other types of diagrams are provided to Arrowhead Framework users. On one hand, the UML use-case diagrams and sequence diagrams are suggested. In addition to graphical representation, other information must be written such as unique id, primary actors, secondary actors, main flow, alternative flows, etc. On the other hand, this useful information should be completed with other views of the system. Therefore, structure and behavior diagrams are recommended to use with different standards such as BPMN, SysML or UML. Depending on the document, the user of the Arrowhead Framework can decide to define the system with an Activity, Sequence, Component or Block Definition diagram as a choice.

It is important to note that the documentation procedures proposed by Arrowhead are agnostic in relation to the design methodologies to be used, although some are recommended and considered more adaptable to the case, like UML and BPMN.

IV. System-of-Systems Level

A system-of-systems can be characterized by five main characteristics in relation to other very large and complex but monolithic systems [12]: i) operational independence of its systems; ii) management independence of the systems; iii) evolutionary development; iv) emergent behavior and, v) geographic distribution.

Operational independence means that a system is an independent part of a system-of-systems and that it can be used to compose other systems-of-systems. Additionally, each system could be managed independently.

The system-of-systems can evolve with functions and purposes added, removed, and modified along its lifetime. Additionally, its functionalities cannot be mapped into any component of the system, i.e., the behaviors are emergent properties of the overall system-of-systems and cannot be assigned to any of its components. By geographic distribution we mean that system components are physically separated.

Two documents are proposed to be used in order to document the system-of-systems level. The SoSD, which shows an abstract view of a SoS and the SoSDD which shows an implementation view of the SoS with its technologies and deployment views.

A. System-of-Systems Description (SoSD) Template

This document should contain an abstract high level view, describing a SoS main functionalities and its generic architecture. It will mainly be used to describe one System-of-Systems in an abstract way, without instantiating into any specific technologies. Examples of its usage are on the description of generic SOA-based installations, like building automation systems or a factory automation system.

The document starts by presenting a high level overview of the system-of-systems, presenting its main building blocks as independent systems. The use of a UML component diagram showing the structural relationships between the systems is suggested. Each of its constituent systems must be further detailed on a System Description document (discussed in Section V).

As usual on a high level description, such document also includes use-cases, which help to understand the expected behavior. Based on these use-cases, the document should include behavior diagrams which express a high level view of the SoS. The use of UML activity diagrams or Business Process Model and Notation (BPMN) [13] or System Modelling Language (SysML) [14] activity diagrams are proposed. The UML Activity diagrams are used to model a higher-level business process or processes. The BPMN diagrams provide a notation that is easily understandable by all business users. Thus, the business analysts can create the initial drafts of the processes to the technical developers responsible for implementing the technology that will perform those processes. The SysML diagrams support the specification, analysis, design, verification and validation systems that include hardware, software, data, personnel, procedures and facilities.

In this document, it is also of paramount importance to include information about the non-functional requirements. The non-functional requirements are related to QoS, usage of resources, reliability, etc. Since most of the SoS, targeted by Arrowhead, can work on a best-effort strategy, the definition of these requirements is optional.

Due to the nature and sensibility of the SoS being targeted by Arrowhead, security is treated separately from other non-functional requirements. This includes the definition of security principles that the SoS needs to follow on a non-technical and generic level, the security objectives, the assets which need to be protected and a list of non-technical security requirements.

B. System-of-Systems Design Description (SoSDD) Template

This document describes how a "System-of-Systems Design Description" document has been implemented on a specific scenario, describing the technologies used and its setup. Therefore, this document points out all necessary Black Box SysD and White Box SysDD documents, which describe the systems used in this realization.

The document starts with an abstract high level view of the SoS realization, describing how its main functionalities can be logically implemented, i.e. which technologies have been used and how it is physically implemented. Specific use-cases are described next, followed by structure and behavior diagrams which clearly identify the technologies used and the setup of this SoS realization. If required, the document can optionally include a description of its physical implementation including, for example, information about location of devices and physical constrains.

The non-functional requirements implemented by this realization must be listed and also its security features. To support the validation of the security attributes of this SoS realization, it is also necessary to include information identifying the data flows in the system as well as its threads and vulnerabilities.

V. System Level

The System Level consists of a black box design document, namely "SysD Template – Black Box Design" and a white box design document, namely "SysDD Template – White Box Design".

The SysD describes the System as a Black Box, documenting its provided and required interfaces, which are defined in the IDD documents, and the corresponding technical solutions. The SysDD describes the System and what it consists of, i.e., how the Black Box will be used to fulfil its task. The White Box SysDD document is optional, since it might expose the knowledge of the company which implemented the system.

A. System Description (SysD) Template – Black Box Design

This document provides the main template for the System Description of Arrowhead compliant Systems. It should be used to define the main services and interfaces of a system, without describing its internal implementation. In the System Description document a proper black-box description of a system is presented, where all the produced/consumed services (with references to the IDD's) are listed. In this way a clear picture of how to interface the system is provided.

While the Arrowhead Framework does not impose any formalism for describing the SysD, it encourages the usage of formal or semi-formal models to provide a rich semantic-based description in order to assess and enable the interoperability of different systems. For example, UML component diagrams provide the right level of information and enable the unambiguous documentation of the provided and required interfaces of a system. This structural view can be complemented with a high-level behavioral view such as sequence diagrams. While sequence diagrams are not sufficient to implement the system, they can precisely describe typical usages of the system and how the different services interact. They can, in addition, serve as test cases for the system, e.g., using the UML Testing Profile.

B. System Design Description (SysDD) Template – White Box Design

This document provides the main template for the description of Arrowhead Systems, technological implementations. In this document the solution for the actual System implementation is pointed out.

This document is optional. If the actual implementation is not needed or cannot be revealed, the usage of the SysD document would be enough for anyone to connect to such a service.

Again, the Arrowhead Framework does not impose any formalism for SysDD, but encourages the usage of formal or semi-formal models in order to enable the automatic generation of code from the specifications as much as possible. When automation is not possible, the SysDD should be precise enough to guide developers towards an implementation that unambiguously matches these specifications. For example, state machines provide proper concepts and notations to yield fully operational code.

VI. Service Level

The Service Level consists of four documents: the SD Template, the IDD Template, the CP Template, and the SP Template, see Fig. 4. The Service Description is technology independent, thus allowing every interested party using it, free of any technological constraints. Technical aspects of a service are revealed in the three remaining documents.

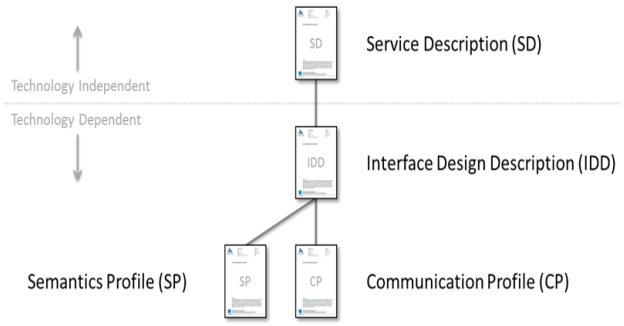

Fig. 4. The Service Description relation

The SD handles the abstract view of the Service. The IDD states the actual solution. The format and protocols for achieving successful information exchange by using a specific technology are presented. The CP contains all the information regarding the transfer protocol, the security mechanism and the data format to be used. The SP defines all the information needed to describe the data format by pointing out what is the type of the encoding (e.g., JSON, XML, EXI compressed XML).

IDD uses the Semantic Profile and the Communication Profile to reduce possible multiple definitions. This means that at a highly mature state of the Arrowhead Framework realization

only a few, technology-specific CPs will exist, and they will be used by various IDDs and Services.

A. Service Description (SD) Template

A Service Description document provides an abstract description of what is needed for systems to provide and/or consume a specific service. SD's for Application Services are created (specified) by the developers of any Arrowhead compliant system and by the developers of the Core Arrowhead Framework services.

The SD shall make it possible for an engineer with technical programming knowledge to achieve an Arrowhead compliant realization of a provider and/or consumer of description of how the service is implemented/realized by using the Communication Profile and the chosen technologies.

The document starts with an overview section describing the main objectives and functionalities of the service and follows on defining the Abstract Interfaces and an Abstract Information Model.

On the Abstract Interfaces section all interfaces should be detailed using UML or SysML Sequence diagrams. The Abstract Information Model section must provide a high level description of the information model with types, attributes and relationships, based on UML Class diagram or SysML Parametric diagram.

Each service also has a different set of non-functional requirements which have to be described and related with higher layer documents. Non-functional requirements regarding QoS, response time, resources and reliability should be presented in the Non-functional Requirements section.

B. Interface Design Description (IDD) Template

An Interface Design Description provides a detailed description of how a service is implemented/realized by using a specific Communication Profile and specific technologies. An IDD is always related to a Service Description; therefore it contains pointers to SD documents.

The Interfaces section describes each of the interfaces in a separate subsection, presenting the correlation among communication profiles and interfaces. Every function included in each interface should include UML Sequence diagrams and the exchanged data structures using UML Class diagrams, UML Component diagrams, SysML Parametric diagrams or SysML Block Definition diagrams. Every function that is included in the interfaces must be described to the necessary extent. The usage of tables to provide that kind of information can be used, for instance, describing method's names, types, input parameters and output information. Every function should also be described in separated sections in order to for someone to easily implement it.

The Information Model section contains detailed information about the data formats used by the Interface. Any metadata information must also be included in this section.

C. Communication Profile (CP) Template

The Communication Profile is identified by the following three characteristics: transfer protocol (e.g., CoAP) security mechanism (e.g., DTLS), and data format (e.g., XML).

The document describes the types of message exchange patterns using this communication profile (examples of Message Exchange Patterns are: request-response, publish-subscribe, one-to-many). Another important issue is to define in detail how the Communication Profile handles security issues, regarding authentication and encryption, based on the protocol specifications. For instance in the use of CoAP, DTLS is enabled.

The document is also used to define the parameters of an endpoint using this communication profile. As an example, in a CoAP based installation, the IP address, the UDP port, and the Resource identifier should be provided. Additionally, it is also required to define the kind of encoding being used (e.g., XML, EXI, JSON).

The Description Format section describes which documentation artifacts are required for the description of a service utilizing this communication profile. For instance, in a CoAP based example, the functions of a service using this communication profile should be documented. Furthermore pointers for the external document containing the format of input and output data should be included (like XML Schemas).

Finally, the document must also contain a list of specifications and standards used by the Communication Profile.

D. Semantic Profile (SP) Template

The Semantic Profile is describes the data format by pointing out what is the type of the encoding (e.g., JSON, XML, EXI compressed XML) and how a specific piece of data (e.g., a temperature value or a pressure value) is encoded.

Obviously, there are a few standards that already define adequate semantic profiles for a width variety of usages, which should also be named.

VII. The Service Repository

Arrowhead's Service repository is a key component of the architecture. The repository contains documentation and design of all core services and systems in Arrowhead, as well as the application-specific systems and services being developed. The repository is composed of a number of elements: i) a folder structure (see Fig. 5), ii) templates for services and systems, and iii) a cookbook on how to use the repository.

The repository folder structure is divided into several classes which represents the maturity of a system or a service. During the lifetime of an Arrowhead system or a service, it is transferred from WORKING_PROPOSAL to APPROVED via FOR_REVIEW. When a system or service is no longer in the official repository, it is moved to OBSOLETE and should not be used in new designs or applications. The repository can store system design, dependencies, service interfaces, communication and semantic profiles, Arrowhead core services and many other design artifacts. The uniform layout of the repository, together with pre-defined templates for services and systems makes it relatively easy for a software engineer to start working on new applications or on modifying existing Arrowhead-compliant applications.

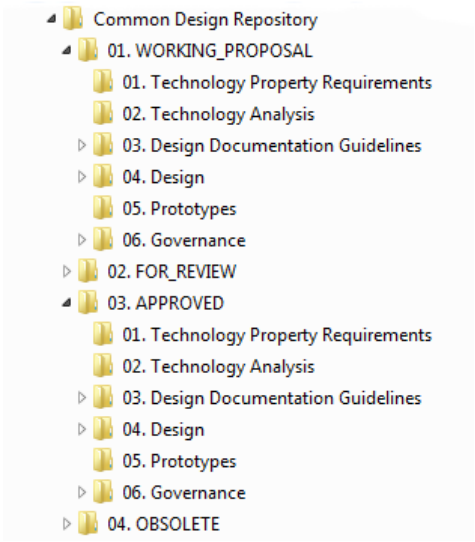

Fig. 5. Repository folder structure

During the course of the Arrowhead project, the repository structure, templates and manuals will be refined and improved. There will also be added test cases on how newly developed services or systems can be tested for Arrowhead Common framework compliance.

VIII. Conclusions

The development of a SOA-based system-of-systems can be a complex task, which has to deal with different systems and services, developed by different people or companies, many times using different principles for describing elements and their connections.

The work being developed on the Arrowhead project aims to fill some of those gaps. In this paper we describe the basic principles of the Arrowhead Framework, focusing on the documentation mechanisms created for the development of SOA systems. While following these principles and structures, system architects and developers allow easier understanding of their systems, as well as easier integration of systems and services developed by others who also followed the Arrowhead Framework.

The Arrowhead documentation framework is structured upon three documentation levels namely: System-of-Systems, System and Service Level. The Arrowhead Framework provides general design descriptions for each level, and description templates for documents at each level. On the System Level the framework contains both the black box and the white box description. Besides having a technology independent description for each service on the Service Level, technology dependent descriptions of the interface design (semantic profile and communication profile) are also part of the framework. This approach of the service descriptions allows having one, general, technology independent service description for each service – for which there could exist various technology-dependent realizations.

The design and documentation of services compliant to the Arrowhead Framework are stored in a common repository. The folder structure reflects the lifetime of the services, which evolves through several states, from the first draft of the documents until becoming an approved version and after evolving to obsolete. The services within the Arrowhead Framework are de-signed for interoperability. In case it is proven, these systems and services can get the statement of Arrowhead compliance.

Acknowledgment

This work is supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and the EU ARTEMIS JU funding, within project ARTEMIS/0001/2012, JU grant nr. 332987 (ARROWHEAD).

References

[1] Thomas Erl, “SOA Design Patterns“, Prentice Hall PTR, ISBN: 0136135161, 2009.

[2] M. Shaw, D. Garlan, “Software Architecture: Perspectives on an Emerging Discipline”, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, ISBN 0-13-182957-2, 1995.

[3] L. Bass, P. Clements, R. Kazman, “Software Architecture in Practice”, Addison Wesley, Boston, MA, USA, 2nd edition, ISBN 0-321-15495-9, 2003.

[4] IEEE 42010-2011 - ISO/IEC/IEEE, “Systems and software engineering - Architecture description”, December 2011.

[5] N. May, “A survey of software architecture viewpoint models”, 6th Australasian Workshop on Software and System Architectures (AWSA 2005), Swinburne University of Technology, Melbourne, Australia, pp. 13–24, 2005.

[6] P. Kruchten, “Architectural Blueprints - The “4+1” View Model of Software Architecture”, IEEE Software, 12(6):42–50, 1995.

[7] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and Judith Stafford, “A practical method for documenting software architectures”, http://repository.cmu.edu/compsci/671/, viewed on 3rd February, 2014, August 2002.

[8] P. Clements et al., “Documenting Software Architecture: Views and Beyond”, Addison Wesley, Boston, MA, USA, 2nd edition, ISBN 978-0321552686, 2010.

[9] C. Hofmeister, R.Nord, D.Soni, “Applied Software Architecture”, Object Technology Series. Addison Wesley, Boston, MA, USA, 1st edition, 2000. ISBN 0-201-32571-3.

[10] D. Norris, “Communicating Complex Architectures with UML and the Rational ADS”, In Proceedings of the IBM Rational Software Development User Conference, 2004.

[11] H-G. Gross, “Component-Based Software Testing with UML”, Springer-Verlag Berlin Heidelberg, ISBN 3-540-26733-6, 2005.

[12] M. W. Maier, “Architecting Principles for Systems-of-Systems”, In Systems Engineering, volume 1, issue 4: pp. 267-284, 1998.

[13] Business Process Model and Notation (BPMN) specification: Available on-line: http://www.omg.org/spec/BPMN/, accessed April 2014.

[14] System Modelling Language (SysML) specification: Available on-line: http://www.omgsysml.org/, accessed April 2014