Plan:

1. The purpose of system architecture is to guarantee required qualities
2. The architecture of a system is described by multiple views
3. The ASM is a view of the system architecture
4. The ASM is the foundation for managing the qualities of the system - see RMIAS

1.

Architecting a software system is an engineering process integrated within the software lifecycle processes, as described in the international standard ISO/IEC/IEEE 12207:2017 (<https://www.iso.org/standard/43447.html>). Architecting a software system is about taking the main technical decisions about realizing the business goals with software elements. The architecture is indeed defined as the “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”. The architecture of a system is an abstraction of the system from which those decisions can be understood and traced back to the needs. The drivers in the decision making process are the expected qualities of the system, either specifically required by the stakeholders, or commonly accepted by the practitioners. Standard catalogues of those system qualities exist, such as the ISO2500 family of standard, which not only defines a nomenclature of qualities but also processes to specify and evaluate them all along the software development lifecycle. They are also referred to as the quality attributes, or the -ilities of the system, and encompass: interoperability, scalability, availability, performance, usability, security, and many more [Software Architecture in Practice, 3rd Edition]. Architecture patterns are usually applied to meet the quality requirements: a pattern is a coded solution to a given problem in a specific context. Patterns in architecture can be an answer to very generic problems (layered architecture to manage dependencies amongst components, service oriented architecture to maximise coherence and minimize coupling), but also to more specific problems (enterprise integration patterns are solutions to technical interoperability amongst various systems). Architecting a software system is therefore a process driven by the need to build software exhibiting qualities, and applying the best practices in the art of software engineering, and the main artefact produced by this process is the so-called software system architecture, i.e. “the fundamental structure of a software system” exhibiting the expected qualities associated with software.

[The process of architecture design has been addressed in many ways, according to the various waves in software engineering, and aligned with the paradigm of each software development method or approach: functional decomposition, aspect-oriented approaches and business process management approaches focus on the functions of the system, while use case and scenario-driven approaches focus on the actors using the system, and data-driven paradigm focuses on the information managed by the system. ]

2.

ISO42010 acknowledges that the architecture of a system needs to be described from multiple viewpoints, and that these viewpoints are not independent of each other. Each architecture viewpoint addresses the concerns of stakeholders having an interest in the system.The standard inherits many concepts from system thinking and complexity management, specifically by leveraging modelling artefacts as first class citizens of the architecture description. This approach in addressing the inherent complexity of today’s socio-technical systems is opposed to reductionism (functional decomposition), and natively integrates the principle of separation of concerns, specifically through the abstraction mechanism. Modelling a system by multiple views is therefore a strong enabler for designing system architecture: the concerns of the many stakeholders are addressed, including the technical concerns in the form of the quality attributes. This approach was already in use before the publication of the standard, and we can consider it as a generalization of well-known approaches in software analysis and design, such as the 4+1 views [Kruchten], but also approaches dedicated to specific domains, such as the conceptual, logical and technical views on data models. The standard does not prescribe what concerns to address, what viewpoint is the uttermost important. It is supposed to support all paradigms, and leaves the identification of the viewpoints and interdependence rules to the specific architecture framework (and architecture description language, ADL). As such it can be considered a general framework for architecture description.

When analysing the architecture of a system, the ArchiMate project [REF] concluded that a system should primarily be described according to 3 dimensions (the aspects of the system): the active structure, the behaviour and the passive structures. The active structure is made of the structural elements of the system capable of performing activity, i.e. the subjects of activity. The behaviour is performed by the active structure (process, function, service). The passive structure is made of the objects on which behaviour is performed, the objects of activity.

The definition of each aspect includes the interdependency amongst the aspects: active structural elements are assigned to behavioural elements, behavioural elements act on passive structural elements.

The ArchiMate language is the outcome of this research project, and does support the description of the system according to those 3 aspects: the language provides constructs to conceptualise elements of each of these dimensions: actor and role, process and function, business and data object. The dependencies amongst the elements of these 3 structures are managed by applying the layering patterns: the business, application and technology layers were initially introduced, complemented later with the strategy (upper) layer and the physical (lower) layer.

As a modern ADL, ArchiMate language supports the description of a system from multiple viewpoints and does conform to the standard 42010.

3.

The principles behind the concept of the ASM represent the basis for the definition of a view: semantic coherence (principle 2.1) and atomicity (principle 2.2) are at the core of addressing the semantic concerns in terms of the system’s architecture. To address semantics, the ASM includes not only the elements of the passive structure (information and data objects), but also the behaviour structure (rules and processes) acting on the passive structure for each element of the active structure (the agents of the system). The coherence principle (principle 2.1) edicts that both aspects cannot be separated, but also establishes rules on how to guarantee that the coherence is maintained. The atomicity principle (principle 2.2) edicts that the scope of the ASM in terms of structural elements, should be delimited by what is required to express a semantic element. Combining those principles guarantees that the foundational design principle of high cohesion is applied when addressing the semantic concerns associated with the system. We therefore state that the ASM itself subsumes the semantic viewpoint of the system and needs to be integrated as such in the architecture description of the system. This viewpoint conforms with ISO42010, and is an instance of the generic viewpoint: it does address the semantic concerns of the stakeholders, and it is described by semantic models covering both the passive and the behaviour structure of the system.

ADLs such as ArchiMate only partially support the ASM viewpoint: they provide the required construct to identify the data (passive structure) and the data processing (behaviour), as well as the agent (active structure) assigned to the behaviour. They however lack the support for actual semantic description. It is quite common for ADLs to be considered a higher level modelling language, and to be complemented with lower-level language (such as BPMN in the description of business processes). The principle 3.1 brings ontological commitment as the foundation for language appropriateness. We consider this as the bed for a language specification dedicated to the design of models belonging to the semantic view. It is the model kind that the ISO42010 standard defines as part of the definition of a viewpoint, which captures the conventions for a type of modelling: it requires that the models belonging to a view are expressed in a modelling language specified in the governing viewpoint. The language appropriateness principle therefore even strengthens the definition of a viewpoint according to the standard, and the ASM defines a viewpoint, but also provides with the requirements for the language supporting the description of a view according to the governing viewpoint.

[Finally, the domain appropriateness (principle 3.2) acknowledges that the semantic concerns may themselves be addressed from multiple viewpoints, one for each domain.]

!! not understood. How to integrate principle 3.2

Rereading my text on both principles it becomes clear that their distinction is very artificial. Principle 3.1 is very meager in only highlighting *that* each language’s underlying ontological commitment impacts the semantics of the model: “thou shall consider the OC”. Contrarily, principle 3.2 shows 4 construction issues that should be taken into consideration when selecting (or building your own domains specific) language. By talking in 3.2 about construction issues, it already implies that one needs to consider the OC from 3.1. Hence, that you cannot verify it makes perfect sense because it doesn’t add anything new.

That said, my conclusion is that both principles fail to address **independent** aspects, in fact they consider the one single aspect of the oc’s of languages. Consequently, 3.2 should be merged into 3.1.

Then the question remains what to do with section 3.2? I think that I can further elaborate on faithfulness, not in terms of language but of models. I can do this in terms of the semantic meaning, but need to include the pragmatic meaning into it. I will take 2-4 days to come up with an alternative. If I fail to achieve anything in that period I’m afraid we should skip it and identify it as future work, in favor of actually submitting this paper.

4.

Considering the ASM as an architecture view of the system has major impacts: not only is it a first class description of the system, but it also better supports the links with the other concerns of the system's architecture. Designing the architecture of a system indeed boils down to addressing the expected qualities of the system, the famous –ilities: security, scalability, extensibility, and many more. There are many approaches to system’s architecture design, but they all converge to ensuring the quality of the system. Architecture as quality assurance was a main R&D topic in the 90’s. It is still today, although it may seem these concerns vanished. The platforms now hide the quality aspects.

The quality concerns always relate either to the entities of the domain, or to the functional aspects of the system. When we address security of the system, we mean the security of the information managed by the system. When we address the scalability of the system, we mean the capacity of the functions of the system to support an increase in its usage.

Example of the RMIAS

# **Discussion**

Addressing the semantic concern of a system’s architecture is not new [REFS], and most approaches that integrate the principle of multiple viewpoints mention semantics at various levels but fail to address it in more than abstract terms. Enterprise modelling is founded on the multiple views on enterprise. TOGAF, the standard enterprise modelling framework from The Open Group, introduces both the business architecture and the information architecture. At the European level, the European Interoperablity Reference Architecture addresses the IOP concern from 4 perspectives: legal, organizational, semantic and technical. The semantic viewpoint however only deals with the data and the meaning of data, but does not directly support the semantic coherence principle: there are obviously relationships between the semantic and the organizational view, but as they are not part of the same view, the cohesion is not ensured.

The notion on semantic coherence can be seen in object-orientation (OO) as well, where the class can be seen as a construct similar to a semantic monolith. Indeed, OO does a very good job at enforcing a reciprocity between data and data operations. However, as a model OO can only provide for an informal “representation” relation with reality as depicted in Fig. 2-2 due to the absence of a formal underlying framework and DoI. Consequently it cannot respond to principle 3.1. Furthermore, OO indeed implements Principle 2.1 on coherence, but Principle 2.2 ….

Shortcomings or consequences of the principles: do we see some? Then these are to be discussed here as well…

For instance:

* Distinction and separation between ontology and the data it holds, and the data that are hold by other components in local databases: what should be separated into the semantic view, what can remain under control of the components, how is their relationship?