Introduction to the Archimate Revolution

In the last decade, a quiet revolution took place in the Enterprise Architecture (EA) domain. This revolution is called Archimate.

This article is the first of a series of articles focusing on some specific aspects of Archimate and practices around the standard.

All Archimate diagrams of this site are done using Archi. This tool is free and great. If you use it, consider making a donation.

A Semantic Modeling Language

Brief Introduction To The Archimate Meta-Model

Archimate is a modeling language that enables to describe and study several aspects of the enterprise:

- Its strategy and motivations,
- Its projects,
- And the 4 core layers of enterprise description:
    - The business layer,
    - The software layer,
    - The technology layer (infrastructure),
    - The physical layer.

All those aspects propose:

- Typed artifacts,
- Typed relationships between artifacts.

Note that several relationships types can interconnect many different types or can be used in many contexts. The detailed description of all artifacts is not in the scope of this article and we can advise the reader to refer to the book *Enterprise Architecture At Work* from Mark Lankhorst.

Archimate also defines interlayer relationships which enables to "connect" the various layers together. This is an absolutely fundamental feature of the language.

A Graph Model

Indeed, an Archimate model is actually a graph. For those who are familiar with the Semantic Web (RDF, RDFS, and OWL), any Archimate model is a semantic graph.

The graph model is the result of the union of all diagrams implemented with the same set of artifacts. For each artifact, the union of all its incoming and outgoing relationships creates the neighborhood of the artifact in the graph model.

When you use Archimate, you represent things in views, using the various artifacts that are available to you. Each element you draw on a certain view (there are many types of views in the standard) will have a certain type, like "Business Process" or "Application Function". Thus, the resulting model will be a set of views, each of them presenting many interconnected artifacts that all are instances of types that have a specific *meaning*.

Viewpoint And Meta-Model

The creation of a semantic language generally implies the creation a meta-model.

Many tools are existing to create meta-models (for instance Eclipse EMF with Sirius, or MetaEdit+ depending if you want to pay or not).

In the history of enterprise architecture, many approaches defined the notion of "view point": in order to act on the enterprise as a whole (or as a system), the first step is to describe the enterprise. In order to describe it, due to the complexity of the task, the architect must use several viewpoints. The union of those viewpoints is defining the model of the enterprise.

The Zachman framework was one of the first publicly available enterprise architecture "framework", enabling to address the many viewpoints describing the enterprise.

The problem with this framework, but also with many other enterprise architecture frameworks, is that they propose many viewpoints, generally each of them proposing a complete separate meta-model for each viewpoint (with artifact types and relationship types), but no consistent view of those various viewpoints.

This is a major issue for the framework because:

- Separate meta-models for separate viewpoints will, most often, enable to create a globally ambiguous model;
- Separate meta-models can create various ways of representing the same reality;
- There can be semantic overlap between various viewpoints and their respective meta-model;
- What should be a single viewpoint can artificially be split into different view points with different meta-models (we'll see later in this article the dramatic consequences of the Longépé Enterprise Architecture model in France).

If the consistency is not "built in" the framework (i.e. in its meta-mode), if it is not part of the primary requirements, then the framework will be very difficult to use.

Indeed, interconnecting various meta-models to be able to create something consistent is not an easy task. Indeed, Archimate seems to be the best illustration of it, and I fear I don't know many other samples [1].

Samples of Non Consistent Modeling Approaches

In IT, we have forgotten about the crucial importance of having a consistent way to describe the "reality", or let's say the concepts that we are manipulating frequently.

Sample #1: UML

UML can be the first example of the non consistency of a meta-model. UML is not consistent because it proposes many meta-models that are not semantically connected together as a whole.

Suppose you made class diagrams and sequence diagrams for a set of classes. You can add a state diagram for a particular process of a specific class `A` of your model. In UML, there is no way to know if this state diagram is consistent or not with the rest of the diagrams that include `A`. This is due to the fact that UML proposes a set of various kinds of views that are not *linked together*.

Each view type has its own meta-model. Some of the meta-models are reusing the same artifacts, which enable modeling tools to connect the artifacts to the views they are used into. But many diagram types are just disconnected meta-models.

This problem can perhaps explain why modeling in UML was progressively abandoned by many projects; because it was not really suited to express in a consistent way what the code should be. If, during design, doing UML model cannot prevent you from making design mistakes, the use of the modeling language is much less interesting [2].

Sample #2: Longépé

Another dramatic inconsistent model is the Longépé French model which defined, instead of Archimate "software layer", two different layers:

- One "functional architecture layer", that could be interpreted as the Application Function part of the Archimate meta-model,
- One "application architecture layer", that could be interpreted as the Application Component part of the Archimate meta-model.

As in Archimate, there is, for Longépé, a business layer and an infrastructure layer, which scopes are almost the same than Archimate's.

In the Longépé model, every layer has a model, and every model can be "derived" from the model of the superior layer. This means that the application layer can be "derived from" the functional layer. This assertion is obscure, misleading and semantically erroneous.

In Archimate, the application function is "assigned to" a application component. That enables to manage the good and the bad assignment, what is called the "urbanisation" in French, notion of good or bad function positioning in the global IT systems:

- If some functions are well positioned in the IT systems, that means that they communicate easily and can evolve in a natural way without questioning the full architecture;
- If some functions are badly positioned in the IT systems (so assigned to the wrong components), we will generate useless dependencies between applications, useless web services, difficulties in terms of evolution, costs and complexity.

The problem of "function positioning" is not in the scope of this article, but it is at the heart of the Longépé's reflection, despite the invention of a very bad meta-model that totally confuses the notion through the concept of *derivation*.

This artificial split of one layer (the application layer of Archimate) into two layers which dependency is erroneously defined, caused a *huge number of French IT projects to fail*, and numerous errors of interpretation and understanding for French architects. The book of Christophe Longépé was at the origin of many misjudgments, errors, confusions and money loss in the French market since the publication of its first edition in 2001.

Sample #3: Projects Creating Their Own Modeling Framework

In consulting missions, I also saw strange practices, as the one of creating a project-specific enterprise architecture modeling framework that evolved throughout the project. Most of the time, the internal framework was incorporating progressively various inconsistent meta-models coming from various modeling standards (such as UML or BPMN or TOGAF).

For sure, most architecture works in the evolving project-specific EA framework were confusing and not usable by software engineers. This often led or contributed to lead the project to its failure.

Archimate, a Consistent Approach

Archimate is proposing a consistent meta-model. This is the first fundamental characteristic which is at the heart of the usability of the modelling standard.

I deeply encourage the meta-model creators to think about it, because it defines the real power of the multiple-views paradigm: multiple views on the same reality can be used *provided there can be consistency between those views*, the consistency being in a unique meta-model.

Note that proposing a consistent meta-model does not imply that the modeling framework is closed and will not evolve. Archimate proposes, since its version 3, a way to extend the meta-model.

We could say that the multiple views consistency-enable graph model is the revolution #0 of the Archimate language.

The Many Revolutions of Archimate

Revolution #1: The Language Just Works

Being consistent is not sufficient for a meta-model.

The quality of the meta-model lies in the pertinence of the semantic artifacts that it proposes (being nodes or relationships). Those artifacts must be as semantically clearly defined and must not bring confusion or multiple alternative representations. In particular, semantic overlapping must be avoided as much as possible [3].

In other terms, if the quality of the meta-model is high, then the modeling will be good. However, if the meta-model is bad, the modeling will be very bad and will cause damages in the project.

Indeed, Archimate propose artifacts that enable the non-ambiguous description of the enterprise processes, strategy and IT.

More: it pushes the architects to describe the *reality in a sane way*, which means in a way that will push for problem exploration and make visible the possible solutions.

For instance, in the case of function positioning, it is easy to count the dependencies between two functions (through derived relationships). If the count is low and always directed in the same way, that probably means the functions are well positioned. If the count if high and/or the directions go both ways, that probably means that the two functions should be one single function.

Archimate language helps and in that sense, it is doing the job. In other terms: the language just works.

This is, really, a revolution. In some cases, meta-models induce architects to think badly, to force themselves to think in an inconsistent model where the semantics are confusing. In Archimate, this is not the case.

Revolution #2: Architects Can Share And Propose Auditable Works

Even if Archimate will not guarantee that 2 enterprise architects will produce the same modeling when representing the same things, using the same standardized language enable each of them to understand the modeling of the other, to challenge it and to discuss it.

We can forget Visio or Powerpoint schema based modeling, that are very ambiguous at several levels. The works become auditable by other Archimate architects.

For sure, documenting textually the views and the artifact themselves is very helpful. As in all modeling languages, without an effort, some views may be a be difficult to understand if we don't know what question it is supposed to answer.

This is a revolution in Enterprise Architecture but also in IT or business architecture. We can work on something *shareable* and *auditable* and so begin to work the simplistic boxes and arrow diagrams.

Enterprise Architecture is becoming more an engineering discipline when it was considered, too often, as a blur esoteric and confusing witchcraft.

Revolution #3: Managing Complex Representations

In Archimate, you can tackle very complex problems.

Due to the fact that you can study several granularity of problems at the same time in the same model [4], the studied complexity can be at several levels:

- You can address very big IT systems or one company or of a group of companies;
- You can analyze in a very detailed way sets of very small intertwined functionality;
- You can study highly distributed systems.

Indeed, big models for IT transformation will quickly contain thousands of artifacts. Note that to ensure consistency on those models, there is a huge systematic work to do.

This is a revolution. The last big program that I did without Archimate was in 2008-2009. It was big, functionally very complex and with a lot of architecture problems. I struggled with UML and BPMN and had to manage the functional consistency by hand. Archimate would have been of great help at the time.

Revolution #4: Aggregate Various Sources of Knowledge

When you are doing a big projects, you have many sources of information, some of them oral (like end users or IT people interviews) and some other being documents or diagrams or wikis, or existing code. It is very easy to forget important stuff or weak signals that hide structural constraints.

With Archimate, you can define views per source of information and work in back office on the consistency of all the information provided when it begin to touch the same domains or software. At the same time, you can formalize what was said in a specific workshop (process, functions, software, etc.) and recreate this consistent view from the graph model.

Despite the fact that it is suite a work, the semantic modeling of Archimate enables to highlight what is consistent and what is not and what complementary information you would need to complete the assessment.

Very often in missions, I can say that my model contains all relevant information that I found.

Revolution #5: Managing Dependencies

Dependencies are the 21rst century problem. Companies have existing IT systems that grew sometimes in a great chaos. Changing stuff is, objectively, complicated, risky, costly.

With Archimate, you can work on dependencies, and so address those risks before the project is started. You can even "objectify" the difficulty or a certain project. If you change a system that is not connected to many other systems, it can be easier than trying to change the core system without knowing about the impacts and the problems that will be induced by this change.

Dependency management are at the heart of the complexity of digital transformation. Because operations must go on, enterprise and IT architects cannot do whatever they want. They have to work on a credible plan that will create a roadmap of transformation taking dependencies into accounts.

Revolution #6: Modeling Transformation, Modeling Time

Modeling digital transformation is a real piece of work if we want to do a serious job. *But it is possible* - and that's a revolution that Archimate enables.

Yes, it is possible to create, pilot, manage, anticipate, huge and complex enterprise digital transformation with Archimate. And frankly, I cannot see what other approach can do that.

At a certain point, reading the forums, I am not convinced that many architects really understood this point. We can now create pretty accurate scenarios of transformation inside Archimate models, taking into account the organization, the strategy, the IT systems, and so on.

Maybe we have here the real role of the enterprise architect. Maybe the enterprise architect should be named the *enterprise transformation architect*.

Enterprises should realize that transforming the processes and the IT systems can become an engineering discipline *at last*, and that using Archimate for this kind of critical topics enables to avoid many troubles and to spend unnecessary money.

Modeling transformation is modeling time and so conventions must be chosen to tag artifacts as being existent or not at a certain phase of the transformation process. As of now, the Archimate modeling tools do not support a temporal view of models (like Franz Gruff AllegroGraph visualizer), we have to take conventions [5].

Revolution #7: Using Archimate In Many Software Activities

Indeed, many software activities can use Archimate on a regular basis, even if the language is more targeting architecture purposes. When we look carefully at nowadays software, most software are highly interconnected to other software or within the enterprise or in the Cloud. Archimate can help analyzing the structural impacts of those interconnections inside the software itself.

Cartography of Systems

Archimate can be used to create cartographies of IT systems but many architects should realize that Archimate models are working tools more than poster tools.

Conclusion

Archimate is, for me, the engineering revolution of digital transformation of the last 15 years. It redefines the enterprise architect role as an enterprise transformation architect role that is able to study with all stakeholders the operational application of a company digital strategy and its impacts on the business, the processes and the IT systems. Once the study is done, Archimate models are of great help to tackle the complex programs, manage the project dependencies and optimize the whole transformation, in terms of time and in terms of efficiency and costs.

We must never forget that the enterprise architect should be a crucial change agent that proposes solutions to a strategic problem and should work in the plain knowledge of the business processes. Most digital transformation offices should also carefully consider using Archimate to manage their transformation plans and try to be in control of the costs and schedule.

See also

- Archimate recipes

Notes

[1] The only modeling approach that look like Archimate is the Aris methodology. Even if the methodology did not cover the full scope of Archimate, the theoretical approach was similar.

[2] We could compare a UML design tool with the CAD tools. The use of the modeling language must enable to work on the *model* of the programs before coding in order to create the best design possible, a design that will be implementable quite straight forwardly. Unfortunately with UML, the objective is not reached.

[3] Some big Enterprise Architecture frameworks used in the military world are proposing very complex meta-models that are proposing, at the same time, sets of disconnected meta-models and semantic overlap between artifacts. Those frameworks are often at the center of big project failures in the military industry.

[4] In projects, I often use or advise to use "level" (or scale) indications of some artifacts like business processes or business functions. The indication can be put in the name of the artifact or as an attribute. For instance, `Accounting` will become `(L1) Accounting` to indicate that, in some views, the business functions will be represented in a high level way. All detailed functions inside this one (aggregation or composition link) will be flagged `(L2)` such as `(L2) Centralization`. This enables to have various levels of relationships between concepts and to be able to drill-down inside the model in order to see more detailed (and more accurate) representations.

[5] The temporal conventions can be as the ones we mentioned for scale management, something like `(P3)` for phase 3. Tagging the artifact name enables to have possibly many instances of the same artifact. Instances that will evolve. The conventions must be chosen very carefully to be able to see what changes and what is remaining the same throughout the whole change process.

(January 2018)