Enterprise Architecture Projects with EAMS

February 2017, Link Consulting

www.linkconsulting.com/eams

Contents

1 Introduction

1.1 Enterprise Architecture

In general, simplifying and streamlining organisations entails not only new processes and systems, but also greater integration with existing ones. More and more integrated processes and systems result not only in an increase in organisation complexity but also in a greater need to know the organisation current and emerging processes and systems. Unfortunately, these two factors feed each other in a positive feedback loop: the more complex the organisation is, the more one needs to know, the harder it gets to know it. Such positive feedback significantly contributes to the increase in costs, risks and execution time for transformations initiatives. So, organizations are becoming more digital every day, but with an increasing in IT complexity as well as in the difficulty to know such IT, yielding many challenges. We name a few:

- Project´s teams spend a significant effort searching and gathering information about organization systems, due to outdated and disperse information.
- Planning ahead becomes a true challenge given the difficulty in knowing the impacts of the decisions taken across the project portfolio. This leads to delays and additional costs due to wrong assumptions and miss information.
- Business stakeholders, Project Managers, IT planners and IT Architects have an hard time to share a common view of the organization and even to adopt a common language about the organization.

The **Enterprise Architecture** is the instrument that enables organizations to mitigate such issues. It enables the alignment of Business and IT, as well as to build a repository of the key elements that makes such alignment explicit and visible to all stakeholders. These elements and their dependencies are the common language about the organization. In Figure 1, we present an artistic view of an organisation, illustrating the key elements types grouped in the three most common layers (Business, Information Systems and Infrastructure) and their typical elements types:

- In Business layer: Actors, Processes, Departments and Products.
- In Information Systems layer: Applications, Repositories and Integrations
- In Infrastructure layer: Nodes and Networks.

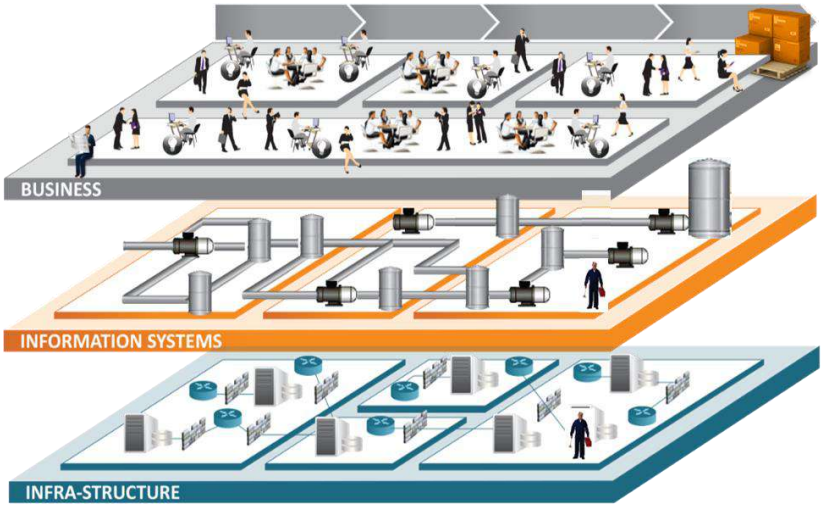


Figure 1 - Enterprise Architecture elements

The above elements types are just an example. One normally follows an **architecture framework** that establishes a rational to classify and structure the elements types one needs to model the organization. As an example, the ArchiMate classifies organisation elements as active, passive, or behaviour descriptive:

- The active elements are the ones that get things done. These are *people* (or *roles*) at the business Layer; *applications* at the information systems layer and *nodes* at the infra-structure layer.
- The passive elements are the ones used as input and output of the work done by active elements. They are mostly information at the different layers, commonly designated as *business objects*, *data objects* and *artefacts*.
- Finally, the behaviour description elements are the ones used describe the behaviour of the active elements. This is basically the *business processes* at the business layer, software *requirements* (or *functions*) at the information system layer and *system software* at the infrastructure layer.

After having a structure to classify the organisation elements types, the next step is to define precisely the elements types needed to be considered in the architecture and how they are related. This is the purpose of the **meta-model**. It establishes the subjects (elements types), the verbs (relationship types) and the states (adjectives) used to express and make statements about the architecture. Examples of elements types are the *Role*, *Department*; *Information System* or *Business Process*. Examples of relationship types are: *Realises*, *Uses*, *Composes*, etc. Examples of states are: *Productive* or *Decommissioned*. The definition of the meta-model is a crucial step because it

establishes the language used to present (model) the architecture of the organization. In an Enterprise Architecture project, it is the basis to configure the desired reports, analysis, architectural views, etc.

Once the meta-model is defined, one can then make statements about the **organisation architecture**, either to define, present or to evaluate it. A well-formed architectural statement instantiates (materialises) the elements and relationship types defined in the meta-model. In graphical modelling tools, an architectural statement is put forward by placing graphical symbols on a drawing canvas. But, an architectural statement can also be made in a textual, non-graphic form. For example, following the element types presented above, the next sentences are well-formed architectural statements: "Sales *department uses* CRM *information system*" or "Customer Complain *business process uses* CRM *information system*", where the italic words are meta-model defined types, and underlying ones are organisation elements of that type. Architecture can also refer to artefact's states as: "CRM *information system is decommissioned*".

Given its relevance in an Enterprise Architecture project, the degree of configurability of the meta-model is a key issue in Enterprise Architecture solutions and tools. One may argue the need to fully define an organisation specific meta-model since the architecture frameworks and standard already define the meta-model an organisation should follow. However, from our experience, most organisations have significant difficulties in adopting such a generic meta-model for many different and practical reasons, which are out of the scope of this document.

In what regards to the flexibility to customise the meta-model there are two approaches, each with its pros and cons. On the one hand, a fixed meta-model structure tool can provide a rich set of out-of-the-box configurations such as input forms, blueprints and analysis, but one is stuck with the structure of that meta-model. On the other hand, a configurable tool allows to define the elements of the meta-model, but then one has to configure the reports, architectural views and analysis.

An **architectural view** is a graphical representation of the organisation architecture from a given point of view at a given moment in time. We call a **blueprint** to the set of all architectural views from a given point of view over all possible moments in time. Therefore, blueprints are sequence of architectural views from the same point of view. In Figure 2 we present an example of a Business Process layered blueprint over the *Account Management* process shown in two moments in time; on the left, with the time slider positioned at 01-06-2014 and on the right positioned at 01-11-2015.

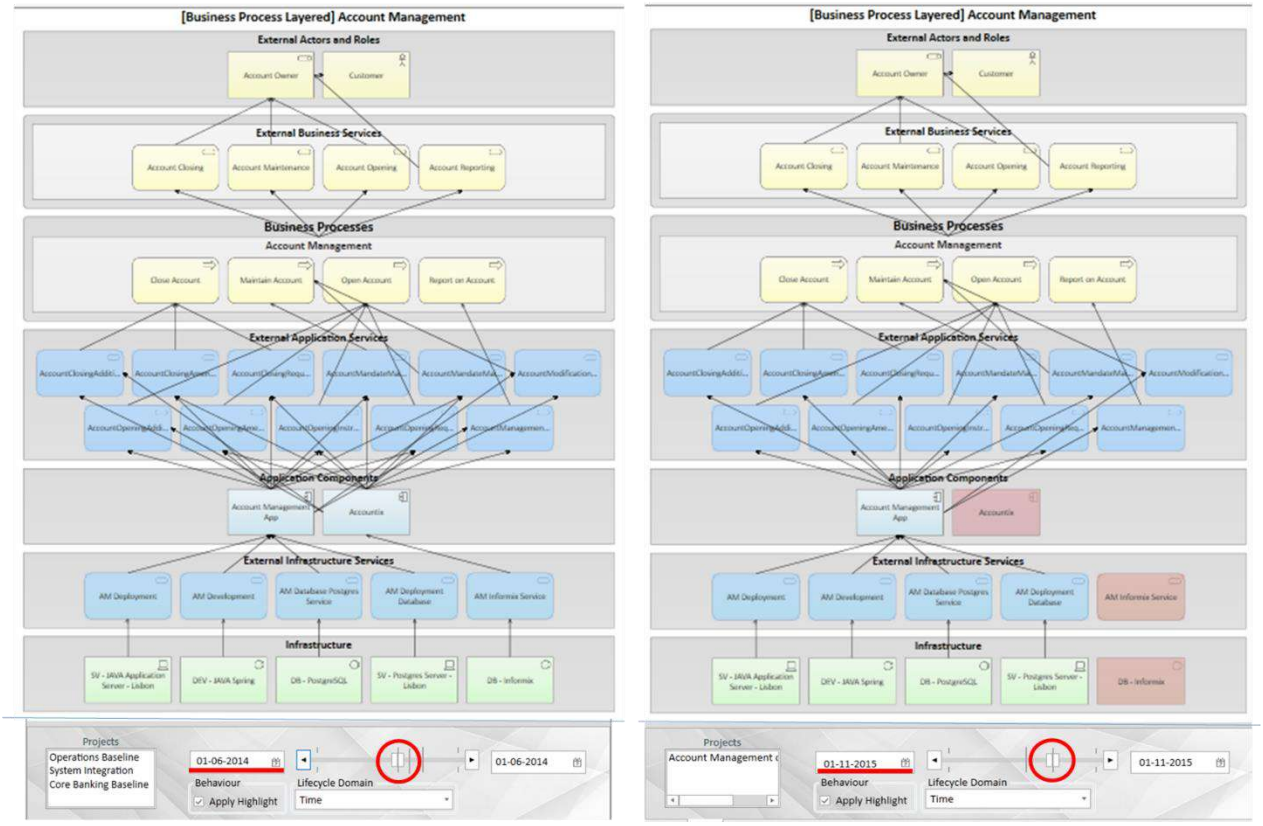


Figure 2 - The Business Process Layered View as an example of an Architectural View

The **repository** is the data base holding the definition of the meta-model, the set organization architecture (set of interrelated objects) and the definition of the blueprints, reports and analysis.

1.2 EAMS

1.2.1 EAMS Rational and Overview

EAMS is an Enterprise Architecture solution that was designed with two key goals in mid regarding blueprints (architectural views):

- They must be kept up-to-date as effortless as possible.
- They should be understood by all stakeholders, regardless their level of modelling skills.

The designs decisions we took to implement the goals stated above were deeply influenced by two findings verified during decades of professional consulting in real organizations using traditional Enterprise Architecture tools:

- *Hand-made models will become obsolete since they must be updated manually*. So, in EAMS all blueprints are generated automatically, and there is no screen to manually draw graphic models.
- *Assessing the AS-IS in a permanent transformation organization is an endless task. It is much easier and useful to map the TO-BE instead*. Easier because people that are transforming the organisation know better what they are working on today (the organisation TO-BE) than what they did in the past (the organisation AS-IS). Useful because the TO-BE is what people need to plan the next transformation projects. In fact, to plan a transformation project that starts 3 months ahead, one needs to know the organisation as will be in 3 months and not as it is today.

So in EAMS:

- All architecture elements have a lifecycle that is defined for each element type. A simple lifecycle can have only alive and dead states. A more complex lifecycle can include conception, gestation, productive, deprecated, decommissioned and finally removed states.
- All blueprints have a time slider that allows the visualization in some point in time. One can go from AS-WAS, to AS-IS to TO-BE simply by moving the slider to the desired date. At each position of the time slider, the blueprint shows the architecture elements in the state corresponding to that point in time.

EAMS collects and stores architectural statements from multiple sources and produces blueprints. Since architectural statements may refer to past, present or future

situations, the contents of generated blueprints presented at a given position of the time slider matches the architectural statements valid at that moment. In the next figure we illustrate EAMS receiving architectural statements from various sources and generating blueprints.

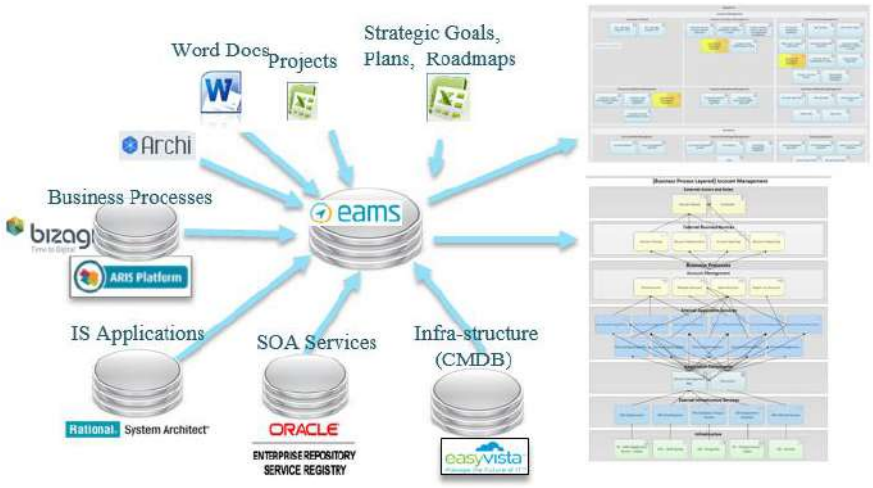


Figure 3- An example of EAMS aggregating information from multiple sources and generating blueprints.

Given the above features, EAMS allows the implementation of an Enterprise Architecture that is:

- **Alive**. The Architecture is be perceived as something that accompanies the dynamics of the organization, translating at any moment the impact of the ongoing projects to those still in pipeline.
- **Facing the Future**. Transformation of organisations happens project by project. EAMS allows the future foreseen in each project to be consolidated and presented into a single, organization wide architecture, and how it evolve day by day according to projects completion. In other words, with EAMS, the architecture is like a crystal ball, showing the future resulting from the consolidation of the promises of the projects yet to be completed.
- **Up-to-date**. Transformation of organisations takes place day by day from project to project. EAMS allows this transformation to be materialized in Architectural blueprints continuously, from AS-IS to the numerous TO-BEs that result from projects yet to be completed.
- **Universal**. EAMS provides simple and intuitive interfaces supporting the sharing of knowledge between multiple technical and non-technical areas. Even though EAMS can import models in different notations (Archimate, BPMN, etc) final users do not have to master such notations, because EAMS provides blueprints and navigation between them on a profile basis.
- **Easy**. EAMS allows to accelerate the process of transformation of the information needed to import from the various sources of information, in particular the

information associated with the projects, from the beginning to the deploy of the artefacts placed in the CMDB, regardless of whether it refers to the past, present or To the future. This capability reduces the effort required to maintain representations.

- **Flexible.** EAMS allows a full configurable meta-model and element life-cycles. One can define relationship types, element types or change existing ones, regardless of the fact that there might exists a large population of objects of that class in the repository.

1.2.2 EAMS Architecture

EAMS is an object oriented system, providing classes and objects. Persistency is provided by *EADB*, an Object Oriented Database optimised for object reference´s graph traversal and time analysis, developed on top of a relational database.

The EAMS components are illustrated in the figure below:

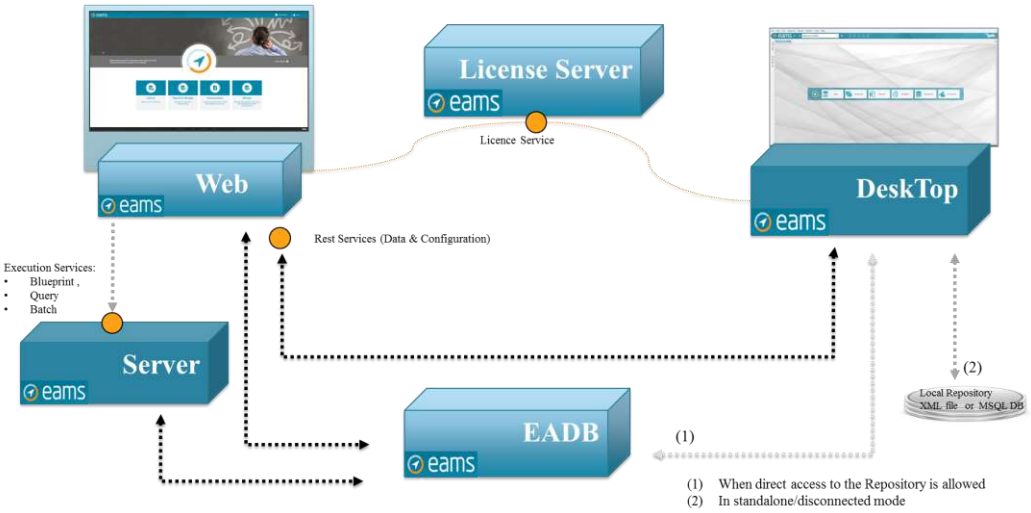


Figure 4 EAMS illustration of Architecture

EAMS Solution consists of five components:

- **EAMS Web** is the web client version of EAMS Application that supports the tool frontend capabilities and makes them available through all organization.
- **EAMS Worker** is the EAMS server that supports the back office workload associated with blueprint generation, batch processing and query execution.
- **EAMS Repository** refers to the enterprise architecture repository that is flexible and adaptable to all organization models and stakeholders needs.
- **EAMS Application** is the Microsoft Windows application that supports most of the tool configuration and integration capabilities. It may work connected

to o EAMS repository or in standalone mode where it uses a local XML file or a MySQL database.

- **EAMS License Server** is a server application that handles and validates all the license details necessary by all EAMS Solution components.

1.2.3 EAMS Licence types

EAMS supports three licenses Types:

- **Explorer**. This is the weakest license as it does allows only to explore the repository data and blueprints without making changes to it contents. However it allows for write a proposal for a change of the repository contents. Typical examples are when proposing a project, the explorer licences allows for write an *architectural scenario* stating the changes resulting from that project. By default, explorer uses the EAMS WEB interface.
- **Editor** Besides exploring, the editor licence allows a user to change the organization model, either by editing data directly in the repository, by approving and publishing architectural scenario, or by running batches that change the repository contents, such as importation batches. As with explorer, the default editor uses the EAMS WEB interface.
- **Designer.** The designer licence is required for change the organization meta-model. It can change classes, blueprints, and other configuration items such as batches, default queries, reports and so on. There are some configurations to the meta-model that can be done in the web, the default interface is the EAMS application.

Functionality \ License type	Explorer	Editor	Designer
Blueprint Explorer and Visualizer	x	x	x
Time slider for time-based visualizations.	x	x	x
Word & PowerPoint Reports(1)	x	x	x
Import & Export Data using XML file format.		x	x
Data Editor		x	x
Architectural Scenarios Editor	x	x	x
Architectural Scenarios Approval Workflow		x	x
Blueprint, Query, Report & Batch Designer			x
Metamodel Designer			x

2 Setting up an Enterprise Architecture Project

To set up an Enterprise Architecture project, one needs to have a clear view of its goals, being the most common one to support an IT governance within the organization. Regardless of such goal, there are a couple of preliminary mile-stones that one must achieve first:

- **Build Architecture repository**. The first mile-stone is to build the architectural repository that is able to generate architectural views to stakeholders. This entails the definition and configuration of the meta-model, and then the filling with the architecture information. The initial filling of the repository is actual optional, since one may consider and incremental approach, where the repository is filled up on-going.

- **Track Architecture changes.** The second mile-stone is to implement an efficient and effortless process of update the repository, to sustain the delivery of up-to-date architectural views of the organisation with minimal human effort. As said previously, it is not fundamental if the project concludes with a complete and accurate repository, given the effort it endeavours. What is fundamental is that the repository can be updated effortless, so that missing parts can be filled over the time.

These mile-stones are necessary conditions for an organisation to know their current and emerging AS-IS. We call Enterprise Cartography to projects aiming at deploy organisation processes and tools to allow an organisation to know their architecture. Enterprise Cartography to projects are necessary step prior to other more advanced goals such as supporting IT Governance or IT Solutions Architecture.

An *Enterprise Cartography* project with EAMS is structured around the following phases:

2.1 Phase 1 - Identify project goals

In general, the number of types of artefacts that can be qualified as relevant to include in the meta-model is quite large. So, it is fundamental to clarify the problems that we intend to respond to with the project, as a way to limit the complexity of the meta-

model to the minimum strictly necessary. Thus, this stage boils down to a set questions whose answers the project should be able to provide.

2.2 Phase 2 - Define and configure the meta-model

After limiting the scope of the project, we can define the meta-model by identifying the most relevant types of artefacts and their interdependencies required to produce the desired answers. This is an entirely straightforward exercise that starts with a standard meta-model (TOGAF Content Model, ArchiMate, among others) and extend or reduce it as required. The result is the definition of the repository meta-model, and consequently, the set of possible architectural sentences one needs to capture into the repository to be able to provide the expected answers.

As the maturity level of the organisation in these domains evolve, one expect the repository meta-model to evolve accordingly. Traditionally, evolution is in the sense of extending the meta-model to new areas, with more detail and with more sources of information. This is a well-known issue and the TOGAF Content Model allows it, foreseeing the extension of the meta-model in different areas.

Our recommendation is to start with one of the EAMS out-of-the box metamodels and then extend according to your needs.

2.3 Phase 3 - Identify the best sources of information to feed the repository

Once the artefacts types and their possible interdependencies are known, one needs to identify the sources of information that allow the collecting of information about the artefacts as effortless as possible.

It is important to separate the issues related to the content from those related to the form of the information. In the next sections we also distinguish between the entities in the meta-model that will have EAMS as its master repository, from the ones that have an external master repository.

2.3.1 Information Source Contents

For each meta-model element type, you should identify:

- If EAMS is the master repository or if it will have an external master repository, in which case EAMS is just a replica that needs to be updated periodically.
- What is the easiest and reliable way to capture state changes in the artefact's life cycle.
    - For the artefacts having EAMS as the master repository, at the minimum, one should identify the information sources holding the TO-BE states (planned ones).
      Consider the case where one wants to use EAMS as a master repository for applications catalogue within the organisation. In this case, one should identify where to know in first hand that a new application is planned to be brought into production or to be decommissioned. This advance knowledge is normally found the plans of the projects (see section Tracking Architecture Changes in chapter Updating the EAMS Repository).

    - For the artefacts having an external master repository, one needs to identify the best sources of information to capture the changes in the artefact change life cycle.

2.3.2 Information Source Format

Regarding the form, we consider relevant to present the 3 most typical scenarios: structured repositories, office documents; models in some notation.

2.3.2.1 Structured information sources

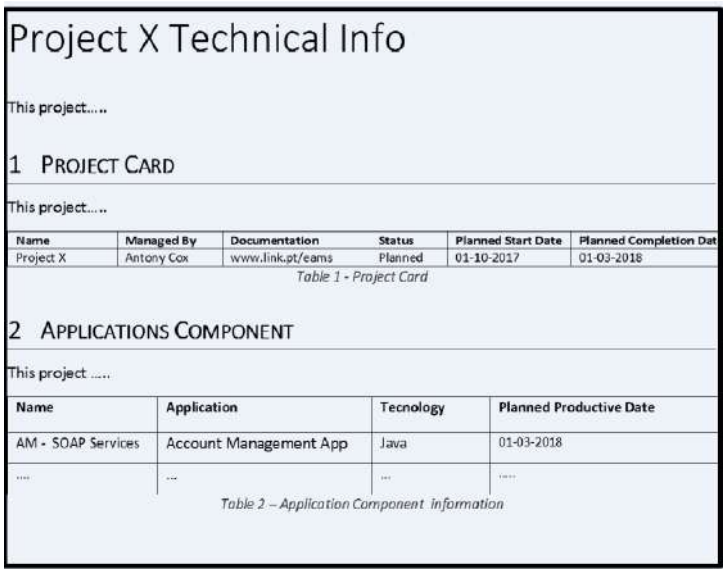
Structured repositories systems are natural information sources candidates. For the artefacts having the EAMS as master repository, such sources can be used for a first importation. For artefacts with an external master repository, one needs to establish the update frequency and necessary transformations.

Structured information sources are mostly a source for productive artefacts. For example: products price list, application services in the services bus platform, applications in the helpdesk support system, and so on. Such systems can be a trustful source of the AS-IS, but they rarely hold information regarding the TO-BE. An exception are the systems supporting the IT developing and deploying process, namely those supporting Agile development paradigm and DevOps tools, such as Jenkins or Sonar. Another exception are the test and quality environments since artefacts being tested will likely become productive.

2.3.2.2 Office Documents as information sources

Office documents are quite common information sources, both regarding the past, present and expected future of the artefacts. However, information is mostly in natural language statements or images that cannot be automatically processed. If such documents are relevant information sources, they must be structured so that they can be processed and information extracted automatically.

EAMS supports importing in MS-Excel and tables in MS-Word documents. For example, the figure below shows an example for MS-Word document that describes a project planned to start in some future date, with a first table for general data and a second one for applications components affected. The table metadata identifies the artefact type and the first column of each table identify the property to be loaded.



Project X Technical Info

This project.....

1 PROJECT CARD

This project.....

Name	Managed By	Documentation	Status	Planned Start Date	Planned Completion Dat
Project X	Antony Cox	www.link.pt/eams	Planned	01-10-2017	01-03-2018

Table 1 - Project Card

2 APPLICATIONS COMPONENT

This project .....

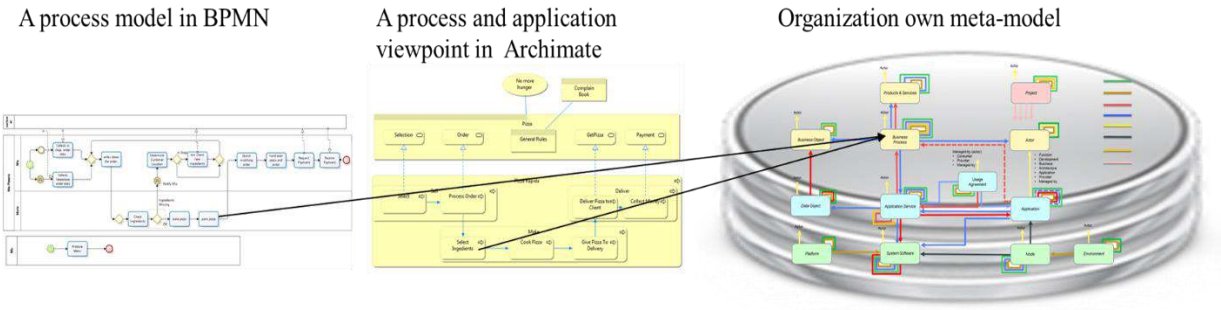
Name	Application	Tecnology	Planned Productive Date
AM - SOAP Services	Account Management App	Java	01-03-2018
....	...	...	.....

Table 2 – Application Component information

By structuring project plans, one can feed the repository with preliminary information about the TO-BE artefacts lifecycle.

2.3.2.3 Models as information sources

Models are also a common source of information regarding the artefacts being involved in projects, and thus are quite often sources of the TO-BE architecture. Common notations are UML, ArchiMate, or BPMN. Each of these models have a structure defined by the notation meta-model, which will likely be different from the one used in the Enterprise Architecture repository, both in the artefact types as well as the relationship types. Thus, it is necessary to define the mapping rules to use during importation as illustrated in the figure below.



2.4 Phase 4 - Configure the Architectural Maps, Integrations and State Propagation rules

After clarifying the objectives of the project, the meta-model necessary to maintain the required information to fulfil these objectives and the sources of information needed to fill the repository, it is time to configure the EAMS.

The configuration of EAMS focuses on the configuration of the following elements:

- Blueprints and navigation rules
- Integration batches
- State propagation batches
- Profiles and users

2.4.1 Define and Configure Blueprints

The blueprints must be defined taking into account the objectives defined for the project and the needs of the stakeholders involved. The EAMS Blueprint generator can be configured to produce a wide range of different maps, including most ArchiMate viewpoints. These can be revised and extended to better fit the project objectives.

It is important to clarify that not all stakeholder wants to deal with the complexity of the repository meta-model. Therefore, the blueprints should act as a *View* in a relational DBMS, allowing navigation and exploration under a perceived meta-model specifically designed to each stakeholder in that blueprint. A typical situation is to reduce the number of entities one need to cross to find the dependency between two concepts, showing only the derived relations instead of the existing ones in the meta-model. For example, in a Repository with the ArchiMate meta-model, users of a given blueprint could experience and navigation over simplified meta-model where business processes

are related applications directly without going through the services and functions in between.

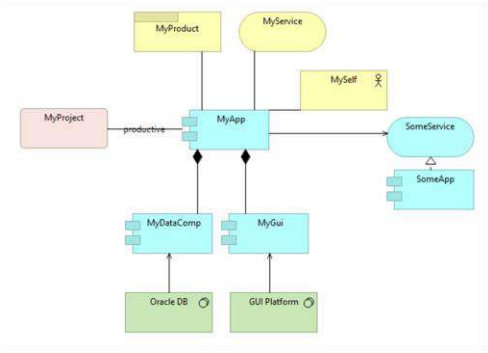
In addition to the maps one must also configure the navigation rules between these and which profiles have access to which maps.

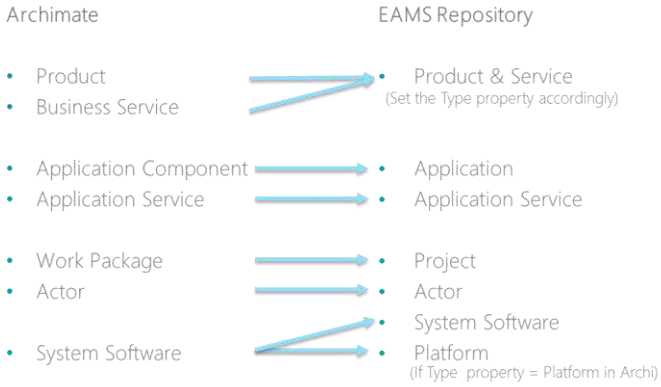
2.4.2 Define and Configure Integrations

EAMS supports the concept of jobs and batches that allow you to define rules for importing and exporting information from a particular source to the repository and vice versa. A batch can have multiple jobs, and a job can be of one of the following types:

- **Export Jobs** - allows the configuration of the information to be exported from the repository to xml files, allowing selection by classes, attributes and also export per instance.
- **Import Jobs**- allows the configuration of the information to be imported from xml files to the repository, as well as the mode of importing the information, among 6 possible ones. For example, if you just update what already exists in the repository, if you just load what does not already exist in the repository, you can import everything by ignoring what's in the repository, and so on.
- **Transformation Jobs**- allows the transformation of import and export intermediate files. There are 2 types of transformation jobs:
    - Transformation of the XML format. In this type, the job only deals with the format of the files, according to the rules coded in a xpath script, thus allowing the conversion between different formats. EAMS includes as examples XML transformation jobs of several formats for the EAMS XML, such as Archi format ".archimate" format files and "XPDL" format files with BPMN information.
    - Transformation of the model and instances. In this type of jobs, classes, properties, references, types of references, and instances inherent to the transformation process are processed according to rules defined.

      In figure below, we present an example of some transformations required between an ArchiMate model and the Repository





In this example,

- Instances of ArchiMate classes of *Product* and *Business Service* are to be mapped into a single class in the repository *Product & Service.*
- Instances of the ArchiMate *System Software* class are mapped into Repository as System Software or as Platform according to the value of the Type variable defined in Archi.
- The other transformations in this example are just the transformation of the names of the classes and the names of the properties.

- **Execution Jobs**. In execution jobs, it is possible to some program or C# code in the transformation process, for example, to perform transformation dependent on the result of services or external information to the repository.
- **Validation Jobs**. It allows the execution of queries (defined in the configuration) and validates the result of them.

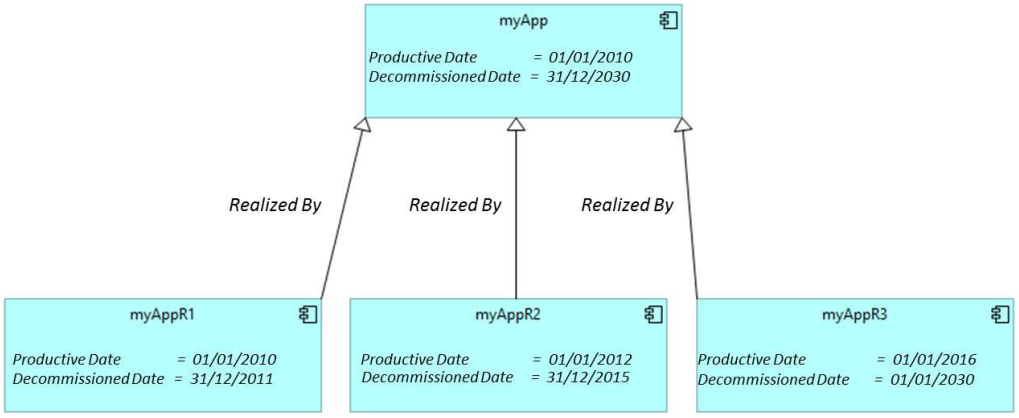
2.4.3 Define and Configure State Propagation Rules

In many situations, properties of some objects are dependent of properties another objects and should be updated accordingly. We call such jobs *State propagation jobs*. To configure a state propagation job one needs to:

- Define an *export job* with the objects that need to be updated together with the objects necessary for such update.
- Define a *transformation job* to actually perform the transformation.
- Define an import job to import the changed objects

For example, consider the case wants to update the productive and decommissioned dates of applications based on the corresponding dates of their releases. This is

illustrated iin the next figure, where application (myApp) is realized by tree releases (myAppR1,… myAppR3).




```
Transformation Job Rule:
For all  object of class Application Component; {
   Productive Date  = min(realizedBy  -> Productive Date);
   Decommissioned = max(realizedBy -> Decommissioned);
}
```

In this example, the export job exports all application components holding a reference of type Realized By. The transformation Job hold the following rules:

For objects of type Application component referenced by *realized by* reference type set:

- *Productive Date* is set to the first Productive Date amongst its Releases.
- *Decommissioned Date* is set to the latest Decommissioned Date amongst its Releases.

Finally, the import job should import all application components all application that are realized by other application components.

2.5 Phase 5 - Populate the Repository with an initial baseline

The initial loading of the Repository is usually done by importing other systems or even existing excel sheets that the client may have with relevant information.

However, loading AS-IS can be a very challenging step in organisations with a low maturity level on these subjects. As a matter of fact, if organisations had their AS-IS

ready to be loaded, they probably would not need an Enterprise Architecture/Cartography project to start with.

The main problem is usually in the quality of data of the information to be loaded, as well as the lack of homogenization at the conceptual level. Concepts such as Information System, Application, Service and Platform are among the most complex to grasp.

The effort of loading information into Repository is tiny. On the contrary, the work needed to prepare the information to be loaded (data cleansing) can be very high. In practice, there are only a few catalogues that are to be loaded, in a Pareto-law approach.

Obviously, the more complete and rigorous the Repository is, the more useful and savings one gets from the Enterprise Architecture solution. Therefore, this initial loading is often perceived as an essential step to the success of the project. In our view, this initial upload is actually optional and should not be placed as a critical success factor for the project. We argue that the true critical success factor is the ability to uploading and updating the Repository on a project by project basis. The more projects, the more rapidly the Repository will be populated and kept updated. So, in the approach presented we will consider that the Repository is loaded incrementally, project by project. This task can be done in parallel with the Architectural Maps configuration.

3 Updating the EAMS Repository

In our view, the architectural repository is the truth that every organization shares. It is on the basis of this truth that decisions are made. In this sense, and regardless of all the logs that do exist, the information must persist. That means that there can be no "deletion" of objects in the repository. Instead, object are by placed in a state of its life cycle corresponding to the dead state. This rule is fundamental to sustain the principle that the architecture repository contains information that is disclosed to the entire organisation, and can be used to take decisions. In this sense, the content of the repository is like the journal where the laws are published or as an accounting record book. In both cases, there is no way to erasing of the published laws, nor ways to delete records from the accounting book. There are, however, operations of revocation or compensation of existing records.

In our view, the above must be true both for AS-IS as well as for the emerging TO-BE. In fact, once the intention to build some artefact in the future is announced, there may be decisions that were taken based on this assumption, and therefore, even when it comes known that, after all, the artefact will no longer be constructed as initially planned, we should not delete that artefact but instead simply update its state from planned to dead (or decommissioned). We call such transition as “abortion”, as in nature.

Despite such a strong remark, EAMS allows for direct deletion of objects in the repository, mainly to easy the early phases of the repository, where one is mostly concerned in the project bootstrap.

Thus, within this context, direct write operations in the repository are only necessary and recommended at project start times or in very particular situations. The normal update process during full operation goes through a workflow in which there is a change proposal, which is validated, approved, and finally published in the repository. The same goes for imports. These can be imported directly into the repository, something we only recommend when starting projects or in particular situations, or they can be imported into a scenario that follows the same process of validation, approval and publication.

We call Architectural Scenario to any change proposal to repository contents, as explained next.

3.1 Architectural Scenarios

EAMS supports the notion of Architectural Scenario, as change requests for updating the repository, whether it is a simple update of the properties of object or a new set of objects resulting from a project.

In the case of projects, our recommendation is that there be at least 2 Architectural Scenarios associated with a project:

- One after the planning phase of the project, becoming known the list of artefacts to be developed and the expected dates of their entry into production. This Architectural Scenario materializes what the project promises, and whose artefacts will be presented in the future vision of architectural blueprints. This scenario defines the TO-BE resulting from that project.
- A second Scenario when the project finishes and delivers the final artefacts to be productive (or decommissioned). This can be just a revision of the first scenario, identifying the gaps between what was promised and what was done. This scenario defines the AS-IS resulting from that project.

EAMS supports the analysis and execution of scenario rules in the context of the contents of the repository before they are actually loaded into the repository. This capability not only simplifies repository management but also validates the consistency between different scenarios before they are loaded into the repository.

EAMS also allows the generation of blueprints and analysis of the merge of various scenarios with the contents of the repository, but also with the "off-repository" scenarios, as if the scenarios were loaded in the repository, thus supporting combined analyses and validations of multiple Scenarios simultaneously. This means that in practice scenario analysis is a simple operation, which can be done at the beginning of the scenario creation process, without entailing any changes to the repository.

If we consider that each project is mapped in a scenario, and that there are N projects to load, EAMS allows you to make analyses of each project individually relative to the content of the repository, as well as make analyses and execute rules with all projects loaded and thus detect inconsistencies and dependencies between projects before loading them into the repository.

It should be noted that the analysis that is really important to analyse a project is not the impact with the AS-IS of the repository, but with the TO-BE of the repository at the expected date of production start of that project. As an example, consider that an Architect is doing (today) an impact analysis of a particular project that is planning to develop a series of artefacts to go into production in 8 months. The impact analysis between the project and repository information should be done with the 8-month TO-BE view in the repository, as it will be on that date that the project artefacts will go into production.

The ability to do analysis between any Architecture Scenario with the expected state of the repository at a particular future date is a key but distinctive feature of EAMS.

3.2 Tracking Architecture Changes with Project Plans

The integration with the project plans is one of the most relevant integrations to get the architecture to follow the real evolution of the organizations. In its very basic essence, a project plan holds two lists: (i) the list of objects to be decommissioned at the end of the project and (ii) the list of objects to be made productive at the end of the project.

By considering the project plans as the primary factor of updating the repository, cartography projects are fundamentally focused on the TO-BE vision, and enables a effortless update of the repository, since all projects have a plan that sustain its time, cost and risks. One could argue that architecture is substantially independent of cost, time, and risk issues. But our experience is just the opposite: the more rigorous the evaluation of cost, time and risk the more complete and detailed the architecture is known. In fact, this finding is perfectly aligned with the most common definition of Architecture, in which it includes knowledge that is necessary for the maintenance and transformation of organisations. Consequently, architectural knowledge is a necessary asset to estimate costs, times and risks associated with transformation initiatives. The close relationship between the Enterprise Architecture and the management key variables associated with transformation is a reliable driver to force project leaders to provide relevant architectural information early in the project lifecycle. Furthermore, assessing the impact on cost, time and risk management is a way of determining the appropriateness of an artefact in the organisation's architecture. Regardless of the theoretical discussion that this line of thought can take us, the more impact an artefact has on those three variables, the more likely it will be identified and known in the plans of a project.

Let us consider an organisation with a pipe line of projects. The upper part of the Figure 5 illustrates a scheduling of projects, where one can see that project *X* execution is scheduled to begin and end at *Tm* and *Tn* respectively, and project *Y* is expected to end between those dates.

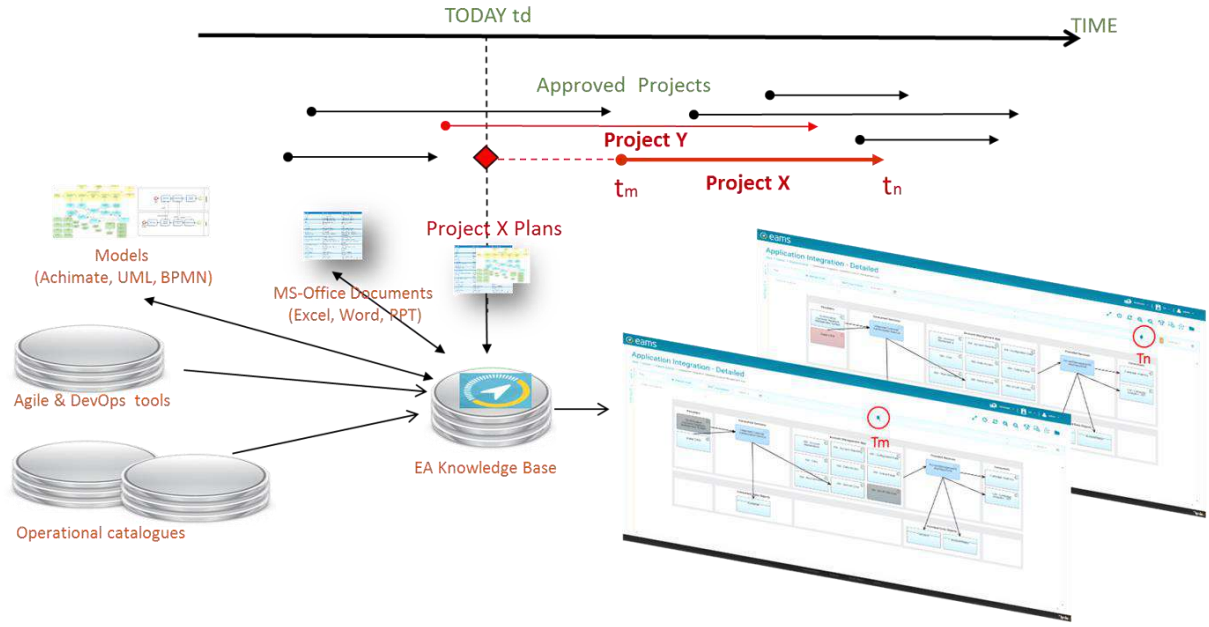


Figure 5 - Loading Project Plans to sustain TO-BE Architecture Views

The lower part of the figure shows the Enterprise Architecture Repository receiving information from various sources of information and producing architectural maps. Each map has a time slider that lets you see how its contents evolve over time. By pre-assigning a colour to each artefact lifecycle stage, one may see the lifecycle state of the artefacts appearing in the map at any point in time.

Consider now that project *X* intends to add a new artefact, and that project Y aims at replacing one artefact by another. Since project *Y* ends before project *X*, the architects of project *X* should take into account the replacement done by project *Y*.

By loading project *X* plans, architects can see the impact of this project in the organisation architecture in the generated blueprints.

The left corner of the Figure 5 shows a single blueprint with the time slider at *Tm* and *Tn* positions, corresponding to project *X* begin and end dates respectively. The blueprint contents shows the impact of all the projects that finish its execution until the date selected in the time slider.

So, when the time slider is set to Tm, the blueprint shows the component to be created by project *X* and *Y* as under-development (grey) and, when the time slider is set to *Tn*, the blueprint shows created artefacts as alive (light blue) and project *Y* removed artefact as dead (red).

The time slider can also operate in gap analysis mode, in which it presents the evolution of each artefact between the dates selected in the time slider. In the following figure abaixo, we can see the same blueprint in gap mode between the time *Tm* and *Tn*. The left part of the artefacts is coloured with the state of the artefact in *Tm* and the right part with the state in *Tn*. This analysis can also be done in depth taking into account the dependency graph of each artefact.

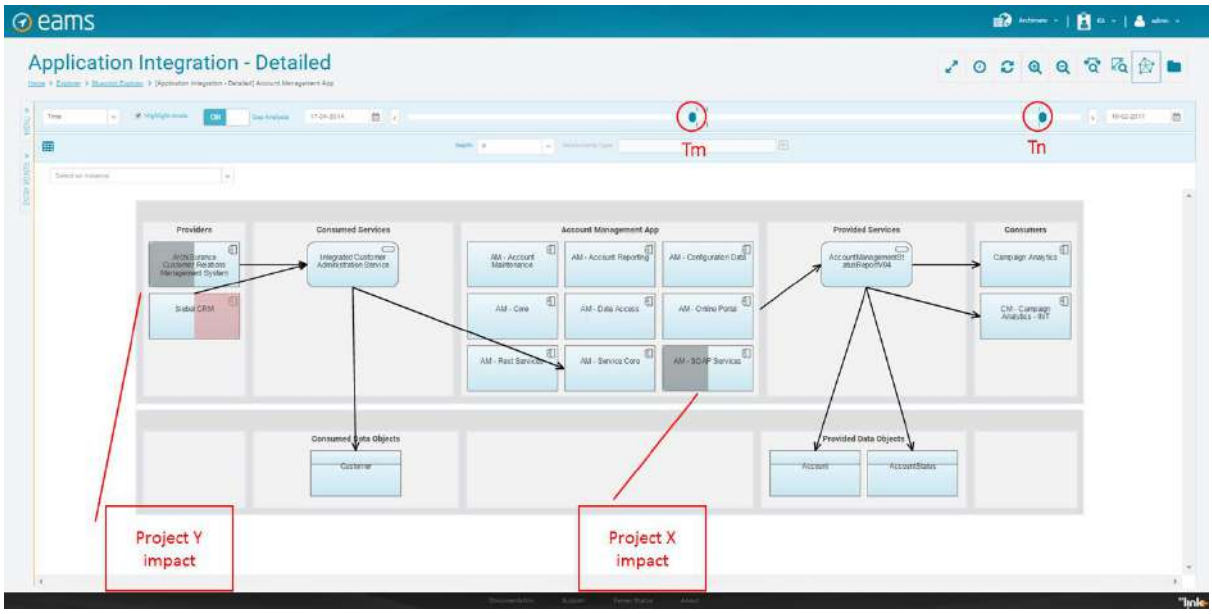


Figure 6 - Architectural views in GAP Analyses mode.

But it is clear the repository cannot be fed only from the promises (plans) of the projects. At least at the end of each project, they should upload their revised plans to the artefacts that were actually produced.

As an example, consider again project *X*, which plans include putting into production an artefact on the date *Tn*. This promise is made at the beginning of the project by setting the alive date of the artefact to be created to *Tn*. The loading of this artefact at the beginning of the project (before *Tn*) is always a promise regarding the artefact aliveness because it refers to a future event. But, loading this same artefact on the date *Tn* is no longer a promise but a statement about the reality, for it refers to the present.

Thus, for objects whose plans were fulfilled, nothing needs to be done. In fact, loading them again at the end of the project is an operation that does not affect the state of the Repository, since one is only repeating an architectural statement that was already in the Repository.

For the remaining artefacts, the revision of the project plan leads four possible situations:

- For the artefacts that were made productive without being initially planned, one needs to add the artefact with the alive date set to *Tn*, being clear that the object has a date of aliveness but does not have a date of conception or gestation.
- For the artefacts that were to be discontinued in the plan but they remain in production, one needs to clear their date of death.
- For the artefacts that fail to born at *Tn* as initially planned, one needs to set the date of death to *Tn*. This artefact has a date gestating and a date of death but no date of aliveness. Corresponds to what in nature is known as abortion.
- Finally, for the artefacts that were to be decommissioned in *Tn*, but remained in production, one just needs erasing the *Tn* value from the date of death.