

Semi-automated Model-Based Generation of Enterprise Architecture Deliverables

*Original*

Semi-automated Model-Based Generation of Enterprise Architecture Deliverables / Saenz, JUAN PABLO; Cárdenas, Steve; Sánchez, Mario; Villalobos, Jorge. - STAMPA. - (2017), pp. 59-73. (Intervento presentato al convegno 20th International Conference on Business Information Systems tenutosi a Pozna, Poland nel 28-30 June 2017) [10.1007/978-3-319-59336-4\_5].

*Availability:*

This version is available at: 11583/2676280 since: 2017-07-28T21:28:07Z

*Publisher:*

Springer International Publishing

*Published*

DOI:10.1007/978-3-319-59336-4\_5

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*

(Article begins on next page)

# Semi-Automated Model-Based Generation of Enterprise Architecture Deliverables

Juan Pablo Sáenz<sup>\*\*</sup>, Steve Cárdenas, Mario Sánchez, and Jorge Villalobos

Systems and Computing Engineering Department  
Universidad de los Andes, Bogotá, Colombia  
`{jp.saenz79,sx.cardenas10,mar-san1,jvillalo}@uniandes.edu.co`

**Abstract.** As part of Enterprise Architecture projects, models are built using different languages and tools to document and analyze the state of business and IT. However, models are just intermediate assets: deliverables are the actual outputs, but they are typically hand built using information from the models, and following the structures specified in EA methods. This requires manual effort, is error-prone, and results in artifacts that might be out-of-date very quickly. This paper addresses this by making a proposal to support the semi-automated generation of EA deliverables using a scripting Domain Specific Language for the creation of deliverable templates.

**Key words:** Enterprise Architecture, Enterprise Modeling, Domain Specific Modeling Languages, Architecture Deliverable

## 1 Introduction

A fundamental element to Enterprise Architecture (EA) are Enterprise Models: they provide a method to structure, abstract, and analyze the complexity inherent to each organization. Besides, they largely satisfy visualization and communication needs and contribute to the effective understanding of the organization in terms of its domains (typically they are categorized as business, application, technology and information [1]). By means of Enterprise Models, the enterprise as a whole is represented through various models [2], each one structured according to some meta-model. Additionally, each model intends to describe, as accurately and completely as possible, some particular aspect of the organization that is of concern to some stakeholder.

Enterprise Modeling refers to the use of a *modeling language* to coherently specify and describe components of an organization along with their relationships [3]. Each Enterprise Model has its own audience, purpose, scope and level of detail. Also, there may be various modeling tools available to build them, ranging from mere drawings to sophisticated web-based Enterprise Modeling tools [4].

However, these models are not completely disjoint. There are always *common concepts* shared between them, and having integrated or global Enterprise

---

<sup>\*\*</sup> The author is currently a Ph.D. student at the Politecnico di Torino

Models would represent a definitive advantage by means of offering a unifying perspective. This would lead to improved cross-cutting analysis offering more valuable findings. Unfortunately, in today’s state of the art, this is not typically possible because the different notations, meta-models, and tools used to create the models keep them from being integrated.

In fact, models are just an intermediate asset in Enterprise Architecture projects: *deliverables* are the actual outputs containing architects’ findings, observations, and analyses, and typically serve to present the current, intermediate, or desired state of the enterprise as a whole or in part. Deliverables are thus supposed to be concrete but partial views of the models, containing the same information but presented in more manageable ways that may vary according to their purpose, scope, and their level of granularity. However, the construction of deliverables normally requires *extensive human intervention* to extract the necessary information from the models and to build the corresponding artifacts. On top of that, the fact that *models are fragmented* make this work even harder and limits the possibility of doing cross-cutting analyses.

This situation motivated the development of the proposal described in this paper, in which the information of several Enterprise Models (built across different modeling tools) is gathered, integrated and analyzed, in a cross-cutting manner, to *automatically generate EA deliverables*. To face the problem of tool heterogeneity, in this proposal the holistic view of the Enterprise Models is achieved through *meta-model mapping* instead of using a deep weaving. To face the problem of the lack of automation, we propose a set of functions to describe *deliverable templates*, and a set of functions for using analysis methods whose results are embedded in the deliverables. All those functions are invoked by means of a DSL.

The rest of the paper is structured as follows. Section 2 discusses Enterprise Architecture deliverables. Section 3 describes the strategy employed in our proposal, including the template language and the analysis language. This section also presents the mapping between the three meta-models that we used for our experiments. Then section 4 presents a case study to demonstrate the applicability of the proposal, section 5 describes the related works, and section 6 concludes the paper.

## 2 Enterprise Architecture deliverables

The creation of Enterprise Architecture deliverables relies on Enterprise Models and typically spans infrastructure components, business applications, business processes, information models, and the many relationships among them [5]. Enterprise Models’ purpose is to accurately capture and represent the current, intermediate, or desired state of the Enterprise, in order to support analysis and decision-making processes. However, building these models is not a trivial task. On the contrary, it faces major challenges due to the high complexity of the today’s organizations.

Figure 1 illustrates the typical process for creating Enterprise Architecture deliverables. It stems from the selection of specific perspectives of the enterprise that are of interest and should be included in the deliverables. For each one, some meta-models are selected (e.g., BPMN, ArchiMate, SysML) and it is possible for these to share concepts and relationships. Then, some tools that support the selected meta-models are used to build the corresponding models. Once these are built, they are expressed in the form of artifacts (catalogs, diagrams, text or matrices), that are finally integrated into Enterprise Architecture deliverables. The format and structure of said deliverables depend on the needs of stakeholders, the specific concerns of the enterprise, and the specifics of the Enterprise Architecture Framework in use.

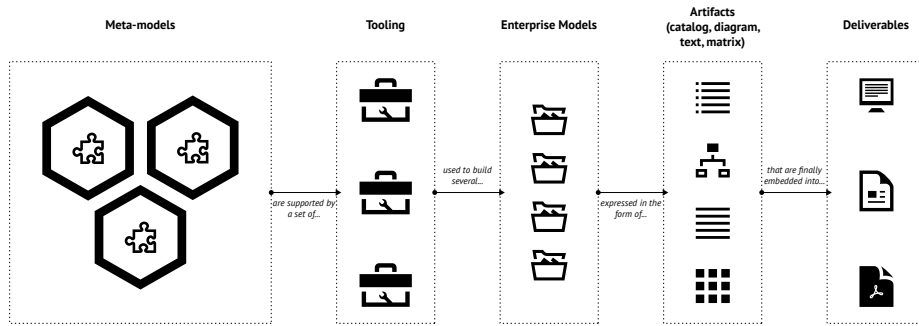


Fig. 1: Enterprise Architecture documentation process

Existing Enterprise Architecture documentation approaches struggle with the information volume and rapidly changing requirements within organizations [6]. Moreover, they rely on a high degree of manual work with very little automation during the documentation and maintenance of Enterprise Models [7]. As a result, Enterprise Architecture documentation endeavors are regarded as time-consuming, cost intensive, and error-prone [8]. For example, analysts may make mistakes when copying information from the models to the deliverables, may omit information, or may not update the deliverables as fast as models are updated.

The context outlined above has motivated various research efforts oriented to automation mechanisms that would improve Enterprise Architecture documentation process [9]. In [10], based on the application of a practitioner survey and a literature review, challenges regarding EA documentation automation were identified and grouped into four high-level categories: data challenges, transformation challenges, business and organizational challenges, and tooling challenges.

The transformation challenges that they identified are the following: the need to consolidate ambiguous concepts imported from the productive systems in the organization; the need to ensure actuality and consistency of collected data from the productive systems; and the need to avoid duplication of EA elements im-

ported from different productive systems of the organization. Meanwhile, tooling challenges are mainly related to the fact that available tools do not support importing, editing, and validating model data for automated EA documentation.

Moreover, attention should be drawn to the fact that Enterprise Architecture documentation occurs within an EA framework. On these frameworks, a set of deliverables (the contractual or formal work products of an architecture project) is specified, in accordance with the methodology proposed.

The TOGAF Architecture Development Method (ADM) [11] for instance, is a generic method that can be used by enterprises in a wide variety of industry types for developing and managing the life-cycle of an EA. When following the ADM, the first step is to modify or extend the proposed generic methodology in order to suit the specific needs of each organization, considering its architecture discipline maturity, its architecture principles, and the previously adopted enterprise-frameworks, among other factors.

The ADM establishes a Preliminary Phase whose purposes include doing any necessary work to initiate and adapt the generic method by defining an organization-specific framework that could be using either the TOGAF deliverables or the deliverables of another framework, depending on the needs of a specific Enterprise. In other words, the Preliminary Phase is about defining “where, what, why, who, and how we do architecture” of the enterprise concerned. Moreover, the level of granularity addressed in this phase depends on the scope and goals of each organization.

Among the deliverables produced at the Preliminary Phase, there is the Tailored Architecture Framework, whose purpose, as its name suggests, is to derive a tailored architecture method, together with a set of expected deliverables and artifacts, as well as its deployed tools, and interfaces with governance models and other frameworks. This is why each EA project follows their customized methodology with its own set of deliverables and artifacts; each one of which is built through the meta-models, the models, the viewpoints and the tools that better satisfy the stakeholder’s visualization and analysis needs.

Adherence to an EA framework adds complexity to the whole EA documentation process. It implies that the set of deliverables and their content varies among enterprises, architecture projects, and methodologies. Which also means that the set of Enterprise Models is variable too, and the artifacts to be included in the deliverables are not predefined.

### 3 Automated Generation of Enterprise Architecture Deliverables

Taking into account the challenges mentioned in the previous section, the proposal presented in this paper aims to automate the generation of EA deliverables from diverse and complementing Enterprise Models. The core of the proposal are the methods to gather information from said models, map common entities and relationships, perform cross-cutting queries and analyses, construct artifacts, and

finally output all of these results in deliverables that follow precise structures. For experimentation purposes, we selected three modeling tools with their corresponding meta-models: Bizagi Business Process modeler, which is based on BPMN and XML Process Definition Language (XPDL); the Iteraplan Enterprise Architecture Management tool, which has its own meta-model; and Archi, an open source ArchiMate modeling tool. These tools were chosen because they target aspects of the organization which are present in typical Enterprise Architecture projects, and because they share some common concepts that make cross-cutting analysis useful and interesting.

Our approach for generating EA deliverables is illustrated in Figure 2 and comprises the following steps: (a) The meta-modeler composes a set of functions intended to perform queries and analysis over the Enterprise Models, and to create artifacts based on these analyses. These functions are invoked through a DSL that we developed and named Pollux. (b) Then, the enterprise architect defines a deliverable template in terms of chapters, sections, and artifacts. This is done through a different DSL that we named Castor. The architect also embeds query and analysis functions invocations into the deliverable template. (c) The resulting deliverable template, along with its embedded functions, is then inputted into the Enterprise Architecture Deliverables Generator (EADG) engine which executes the corresponding queries and analysis over the models. (d) Subsequently, the EADG builds the artifacts (catalogs, matrices, and diagrams) that are represented as texts, tables and images. Finally, (e) based on the deliverable template and the artifacts produced by the engine, the document generator composes the deliverable and exports it in several formats, depending on the stakeholder’s visualization requirements.

We now describe the four main components of our approach: the template language (Castor), the query language (Pollux), the mapping procedure between meta-models, and the EADG.

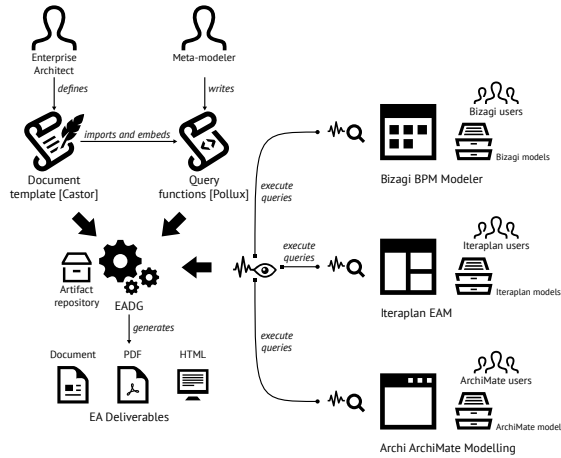


Fig. 2: EA deliverable generation process

### 3.1 Castor: the template language

Castor is a set of functions along with a Domain Specific Language intended to define the structure of deliverables based on information extracted from Enterprise Models. The actual content for the deliverable is also extracted from the models through the usage of query and analysis functions (Pollux set of functions).

The main function of Castor is to specify deliverables' structures in terms of chapters and sections. Castor also has a set of directives for: importing external files, such as text and images; invoking the query and analysis functions; building concrete EA artifacts with the data brought from the EA models; including template fragments to favor reuse; and embedding control flow directives. Table 1 presents the whole set of instructions available in the Castor DSL, along with their description.

Table 1: Castor DSL instructions

<b>General functions</b>	
<code>importFunction</code>	Imports the functions of a given library, which is associated to a certain Enterprise Modeling tool.
<code>importSource</code>	Defines the set of Enterprise Models to be included into the Enterprise Architecture Deliverables generation.
<code>template</code>	Defines the name of the resulting output file, which corresponds to the generated deliverable.
<b>Text functions</b>	
<code>chapter - section - text</code>	Inserts a new chapter (into the deliverable), section (into the chapter) or text fragment, into the deliverable.
<b>Artifact functions</b>	
<code>catalog - matrix - image</code>	Displays the corresponding artifact with data gathered from one or several models. Images typically correspond to Enterprise Modeling diagrams.
<b>Flow control functions</b>	
<code>forEach</code>	Allows the execution of a set of instructions in an iterative way.
<code>if - else - else if</code>	Allows the execution of a set of instructions according to conditional statements.

Listing 1 presents an example of the usage of Castor. First of all, it is necessary to load a set of models that will provide the information for the deliverable (line 1), as well as the set of query and analysis functions to pull content from the previously models (line 2). In the example, a library of functions to communicate with *Archi Modeler* is imported along with a model called *archisurance* (corresponding the ArchiSurance case study from Archi). Line 1 also defines an alias

for the model: `$mArchi`. Finally, in line 4 some basic features for the template are defined (title and output directory).

In the ensuing lines, the structure of the deliverable is specified by means of containers which can be primary, for chapters and sections, or secondary for the case of tables and lists. In Listing 1, line 5 defines a new Chapter in the template by providing its name, and line 6 defines a Section within that chapter also with a given title.

Within the containers, artifacts of different kinds can be included by means of specific directives. These serve to include text, images, and template fragments, or to embed artifacts (matrix, catalogs, and diagrams). In the listing, lines 7 and 8 demonstrate the inclusion of text content by means of the invocation of the function `getInformationView()` and `getDocumentation()` on the Archi model.

---

```

1 importModel "archisurance" as $mArchi
2 importFunction "./archi.jar" as $fArchi
3
4 template {name:"Archisurance deliverable", output_dir: "./deliverable"}
5 chapter {title:"Target Business Architecture"}
6 section {title:"Business goals and objectives"}
7 text {contents:
8   $fArchi.getInformationView($mArchi,"Goal and Principle View").
      getDocumentation()}
9
10 foreach $fArchi.GetProcess() as $process:
11   text {contents: $process.name}
12   if($fBpm.ProcessExists($process.name) == true)
13     catalog {function: $fBpm.GetActivities($process.name)}
14   else:
15     text {contents: "The process does not contains activities"}
16   end
17 /foreach
18
19 catalog {function: $fArchi.getViewElements($mArchi, "Goal and Principle View"
20 )}
```

---

Listing 1: Castor sample code

On top of the above, there is an additional kind of directives intended to provide a more dynamic control over the elements in the deliverable, based on the contents of the EA models. These directives are expressed through conditionals and cycles. Cycles allow the composition of instructions such as “*create a section in the document for each business process brought from Bizagi BPM Modeler, including its name and description*” (Listing 1, lines 10 to 17). Using conditionals, it becomes possible to express statements such as “*embed into the document the name of a business process and, in case it has activities, their names and descriptions. Otherwise, if the process does not have activities associated, display a message informing the situation*” (Listing 1, lines 12-16).



### 3.2 Pollux: the query language

Pollux is both a language and a set of functions to communicate with EA modeling tools to perform queries and analysis over the models, and to generate artifacts using their results. These functions are classified into query and analysis functions, and artifact generation functions. The first ones are responsible for extracting information from the models and applying filters based on any criteria specified by the user; the second group of functions is responsible for building artifacts. This means generating matrices, catalogs or diagrams based on the information extracted by the query and analysis functions. Lines 20 and 21 of Listing 1 illustrate this by invoking the function to create a catalog and giving this information the result of querying the Archi model using the `getViewElements()` function.

Through the use of these functions, it is possible to build artifacts by gathering and integrating information from different models that might be built across different tools. This feature enables the recognition of relationships between the elements of several models, that otherwise would go unnoticed. For instance, to determine which server supports a given business process. Therefore, this feature significantly supports and enhances the analysis tasks of the enterprise architect.

Analysis functions are generally related to quantitative analysis over the elements of a model when there is enough information into it to perform the function [12]. For instance, the response times of the business processes and applications.

It is important to mention that all the functions were developed as implementations of a common interface. This means that the current libraries can be extended to incorporate new functions or to support communication with new tools that may be incorporated in the future. These functions are packaged into a JAR (Java ARchive) that later will be imported into the deliverable template. Table 2 exemplifies the basic set of functions available to gather and deal with Archi Enterprise Models.

### 3.3 Mapping between meta-models

In order to be able to perform queries and analysis in a cross-cutting manner and generate meaningful artifacts, it is necessary to be able to find relationships between models built with different tools. To support this, it was necessary to identify common concepts between the meta-models by means of establishing a mapping between equivalent elements. In the case of the sample tools that we selected, this included a mapping between ArchiMate and BPMN, one between BPMN and Iteraplan, and one between Iteraplan and ArchiMate. Table 3 describes, at first, the mapping that was made between entities in ArchiMate and entities in BPMN. It is important to highlight that beyond those entities that are specific to the meta-models, other entities belonging to the tool, were included. That is the case of the Bizagi element called *Entity*, which does not belong to BPMN.

Table 2: Pollux DSL instructions for Archi

<b>Views</b>	
<code>getViewInformation</code>	Gets the information (properties) of a certain view.
<code>getViews</code>	Gets a list of all the views into the Archi Enterprise Model.
<code>getViewImage</code>	Gets the graphical representation (diagram) associated to a certain view.
<code>getViewElements</code>	Gets the elements present in a certain view.
<b>Layers</b>	
<code>getElement</code>	Gets an element along with its attributes, properties, and relationships.
<code>getElementsByType</code>	Gets the set of elements of a certain type.
<code>getElementsByLayer</code>	Gets the elements belonging to a certain layer. For instance, Business layer, Application layer, Technology layer.
<code>getProcess</code>	Gets the elements whose type is process.
<b>Canvas</b>	
<code>getContainersFromView</code>	Delivers a list of the containers present in a certain Enterprise model.
<code>getViewElementsByContainer</code>	Delivers a list of elements or notes that are placed inside a certain container.

Further on, the mapping between the entities from ArchiMate and Iteraplan is presented. In this regard, it is noticeable that the mapping between the entities does not always comply a one-to-one correspondence. In fact, several Iteraplan business concepts do not have any associated entity in the ArchiMate business layer meta-model.

Finally, the last section of the table presents the mapping between Iteraplan and Bizagi meta-models. There are few mappings as the Iteraplan meta-model is more focused on project management at a lower level of detail while Bizagi offers a high level of detail over the business process. In the Case Study section, an example of a cross-cutting query and analysis function that depends on this mapping proposal is presented.

### 3.4 Generation of the Enterprise Architecture deliverables

Once the deliverable template is defined along with the Pollux functions and the external files, the EADG engine generates the document which is consistent with a meta-model, proposed by us, to represent the different kind of components that might be present in a text processing tool. Entities such as document, document body, container element, content element, table, list, image, text, table row, table cell and list item are included into this meta-model. Lastly, the document generator composes the deliverable, and according to the stakeholder's visualization preferences, exports it in Microsoft Word, PDF or HTML format.

Table 3: Mapping between modeling tools meta-models

<b>ArchiMate business layer</b>	<b>Bizagi element</b>
Business process	Business process diagram, Pools, Lanes
Function	Task, Sub-Process
Business event	Event
Business Object	Entity
Business Role	Lane, Organizational role
<b>ArchiMate application layer</b>	
Application function	Service, Task, Script task
Data object	Entity
<b>ArchiMate technology layer</b>	
Device	
Artifact	
<b>ArchiMate business layer</b>	<b>Iteraplan (business)</b>
	Business domain
	Information system domains
	Architectural domains
Business Process	Business process
Business Role	Business unit
Product, Business service	Product
	Business mapping
Business function	Business function
Business object	Business object
<b>ArchiMate application layer</b>	<b>Iteraplan (application)</b>
Relation, Application Interface	Interface
Application component, Application collaboration	Information system
Application service, Application function	IT service
<b>ArchiMate technology layer</b>	<b>Iteraplan (technology)</b>
Technical component	System software, Artifact
Infrastructure element	Node, Device, Network
<b>ArchiMate (implementation/migration)</b>	
Gap	Project
<b>Iteraplan (business)</b>	<b>Bizagi element</b>
Business process	Business process diagram, Pools, Lanes
Business function	Task, Sub-process
Business object	Entity
<b>Iteraplan (application)</b>	
Business object	Entity

## 4 Case Study

To illustrate the proposal presented in this paper, a case study was developed on the basis of three inputs: an EA academic project; an EA framework specifically tailored for this project; and the set of three different Enterprise Modeling tools, mentioned in the previous section (Bizagi BPM Modeler, Iteraplan EAM, and Archi ArchiMate modeling).

The experimentation consisted in generating Enterprise Architecture deliverables for Editorial de los Alpes (EDLA), an academical exercise to simulate a publishing house that is responsible for carrying out the complete textbook production process, from the selection of the author to the distribution to the points of sale. The main organizational goals of the publishing house are to break into the digital market, lower the operational costs and increase the annual sales.

The meta-model of EDLA is composed of 13 domains: applications, business motivation model, business process architecture, business partners, financial structure, human resources, information, infrastructure, business assets, organizational structure, products, services, and technology. There are 109 entities modeled into these domains and the model built upon this meta-model, has about 1000 elements and more than 1500 relationships.

A deliverable of 48 pages was produced and EADG was able to generate and insert 26 customized artifacts (catalogs, matrices, diagrams, and texts), by executing query and analysis functions, some of them in a cross-cutting manner, over several models built on Bizagi BPM Modeler, Iteraplan EAM, and Archi ArchiMate modeling, as shown in Figures 3 and 4.

The Enterprise Models built on each tool were the following. In Archi, our Enterprise Architecture research group developed all the models regarding business motivators, business canvas, business capabilities, competitors, infrastructure, value chain, and stakeholders. In Bizagi all the business process were modeled. And in Iteraplan, the landscape diagram, cluster diagram, nesting cluster, information flow, and portfolio flow were generated based on the information corresponding to the 14 building blocks.

The sample code in Listing 2 illustrates the capability of our tool to perform cross-cutting queries and analysis by extracting certain information from several models built across different tools. In this example, at first, a query is performed over the projects registered in Iteraplan and their information is embedded as text into the deliverable (Listing 2, lines 15-21). Then, their business processes are taken from Bizagi, embedding the corresponding diagram and catalog of tasks (Listing 2, lines 24-32). Finally, the application services that support each task are gathered from Archi and embedded into the deliverable as a catalog (Listing 2, line 35).

The cross-cutting query involves elements that belong to different models and were built in different tools. By means of the meta-model mapping presented above, it is possible to move across the elements and their relationships.

---

```

1 importFunction "./archi.jar" as $fArchi
2 importModel "archisurance" as $mArchi
3
4 importFunction "./process.jar" as $fBizagi
5 importModel "process" as $mBizagi
6
7 importFunction "./iteraplan.jar" as $fIteraplan
8 importModel "http://eadg.edu.net/iteraplan/" as $mIteraplan
9
10 template { name : "EDLA deliverable", output_dir : "./ deliverable"}
11 chapter { title : "Project Management"}
12 section { title : "Key projects"}
13
14 /* Iterate over the projects in Iteraplan */
15 <forEach $mIteraplan.getElementsByType($mIteraplan, "project" as $project:$>
16
17     Project name: $<text{contents:$project.name}$>
18     Description: $<text{contents:$project.description}$>
19     Strategic drivers: $<text{contents:$project.strategicDrivers}$>
20     Total cost: $<text{contents:$project.costs}$>
21     Accountability: $<text{contents:$project.accountability}$>
22
23     /* Iterate over the business process of the current project in Bizagi */
24     <forEach $project.businessProcess as $bp:$>
25
26         Process name: $<text{contents:$bp.name}$>
27
28         /* Display the diagram of the current business process */
29         <image{id:"bpImg", uri:$bp.getDiagram($mBizagi, $bp.name)}$>
30
31         /* Embed a catalog with the tasks of the current business process */
32         <catalog{function:$bp.getTasks($mBizagi)}$>
33
34         /* Embed a catalog with the Application services (modeled in Archi) that
35            supports the business process */
36         <catalog{function:$mArchi.getRelatedElements("businessProcess", bp.name, "
37            applicationService")}$>
38     </forEach$>
39 </forEach$>

```

---

Listing 2: Cross-cutting function over the three tools code sample

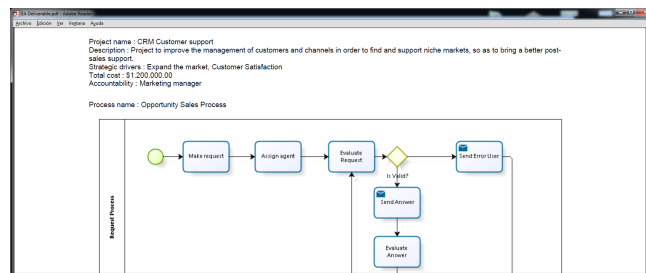
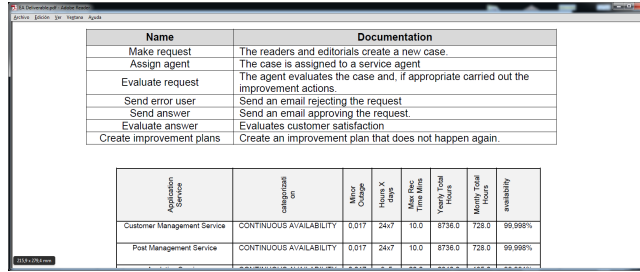


Fig. 3: Project information from Iteraplan and Business Process diagram from Bizagi



Name	Documentation
Make request	The readers and editorials create a new case.
Assign agent	The case is assigned to a service agent.
Evaluate request	The agent evaluates the case and, if appropriate carried out the improvement actions.
Send error user	Send an email rejecting the request.
Send answer	Send an email approving the request.
Evaluate answer	Evaluates customer satisfaction.
Create improvement plans	Create an improvement plan that does not happen again.

Application Service	Availability	Minor Outage	Hours X Days	Hours X Days	Hours X Days	Hours X Days	Hours X Days	Hours X Days	Hours X Days
Customer Management Service	CONTINUOUS AVAILABILITY	0.017	24x7	10.0	8736.0	728.0	99.998%		
Post Management Service	CONTINUOUS AVAILABILITY	0.017	24x7	10.0	8736.0	728.0	99.998%		

Fig. 4: Catalog with tasks from Bizagi and Application services from Archi

## 5 Related Work

Jackson et al. [13] state that a review document or any other paper artifact can be interpreted as a view of the system or engineering model. Based on this premise, they document the development of a tool and a method to produce sophisticated artifacts as views and by-products of integrated models. In their proposal, any paper artifact is assumed as a serialized model that describes and narrates a set of views of the main engineering model. The document structure is completely designed and modeled as in UML, using a custom profile, while the content is provided through query and analysis functions whose inputs are pointers to the main engineering model. Then, the software produces a DocBook XML file, which may be used to generate the final document in PDF or HTML format.

A methodology for automatic software documentation generation is proposed in [14]. In this, the automatic generation is achieved through the design of a documentation model and the definition of mapping relationships between its elements and the system model. Based on this artifacts, the mapping process is able to extract graphical and textual information from UML and SmartC models, organizes them according to the documentation model and generates the software documentation into a Microsoft Word document.

Buschle et al. [15] focused in automating the collection of the data used to build the organization-wide models. The paper illustrates how a vulnerability scanner can be utilized for data collection aimed to automatically create Enterprise Architecture models, mainly covering infrastructure aspects (Application and Technology layer of the organization). Moreover, the outcomes from the vulnerability scanner are taken by an EA analysis tool, responsible for asserting certain system quality attributes. Scanners do not deliver complete EA models (especially those regarding Business Layer)

A particular Enterprise Service Bus (ESB) implementation can be used to extract EA relevant information according to [16] and [17]. This statement is motivated by the idea that an ESB can be considered “the nervous system of an enterprise interconnecting business applications and processes as an information source”. Moreover, based on the application of a survey, the paper concludes

that the SAP PI ESB seems to be suitable and a reasonable start point for an automated EA documentation endeavor.

All the proposals outlined above share concerns regarding the time consuming, cost intensive and error prone nature of EA documentation and maintenance. Their solution approaches mainly focus on the automatic generation of Enterprise Models based on the execution of scanning tools over IT systems that are already deployed. Likewise, in [10] transformation challenges and tooling challenges are related to the need to ensure actuality and consistency of collected data from the productive systems, as well as the inability of the available tools to support importing, editing and validating model data for automated EA documentation.

On the contrary, our proposal focuses on automating EA deliverables generation based on a set of Enterprise Models that are already built using different tools and according to several meta-models. Moreover, in addition to the possibility to automatically gather information from these models in order to compose the deliverables, our proposal allows the integration between them through a meta-model mapping, which enables, as well, cross-cutting analysis over different kinds of Enterprise Models. These Enterprise Models are not restricted to Application and Technology layers, they also represent business logic that may not be gathered from the productive systems of the organization.

## 6 Conclusions

This paper presents an approach for automatically generating Enterprise Architecture deliverables by integrating multiple Enterprise Models built across different tools. A Domain Specific Language was developed to invoke the query and analysis functions that are embedded into a deliverable template. Likewise, a set of query and analysis functions were developed in order to communicate and gather information from the Enterprise Models. The information extracted from these models is embedded into the deliverables in the form of artifacts, such as catalogs, matrices, and diagrams. Nevertheless, the value-added of this proposal does not completely lie on the capability to automatically generate the Enterprise Architecture deliverables by embedding information extracted from many Enterprise Models. Besides that, there is the opportunity to perform cross-cutting queries and analysis by means of a mapping proposal between common entities and relationships from several meta-models. To illustrate our approach, Bizagi Business Process modeler, Iteraplan Enterprise Architecture Management tool, and Archi ArchiMate modeler meta-models were mapped, and an Enterprise Architecture deliverable was generated. We think that the Enterprise Architecture documentation may substantially benefit from our proposal, given the fact that currently this process relies on a high degree of manual work, and is regarded as time-consuming, cost intensive, and error-prone. Moreover, the cross-cutting queries and analysis enable the recognition of relationships between the elements of several models, that otherwise would go unnoticed.

## References

1. M. Lankhorst, *Enterprise Architecture at Work*, vol. 10. 2009.
2. M. M. Lankhorst, "Enterprise architecture modelling—the issue of integration," *Advanced Engineering Informatics*, vol. 18, no. 4, pp. 205 – 216, 2004. Enterprise Modelling and System Support.
3. H. Jonkers, M. M. Lankhorst, R. van Buuren, S. Hoppenbrouwers, M. M. Bonsangue, and L. van der Torre, "Concepts for modeling enterprise architectures," *International Journal of Cooperative Information Systems*, vol. 13, pp. 257–287, 2004.
4. F. Matthes, S. Buckl, J. Leitel, and C. M. Schweda, "Enterprise Architecture Management Tool Survey 2008," tech. rep., 2008.
5. S. Buckl, F. Matthes, S. Roth, C. Schulz, and C. M. Schweda, "A Conceptual Framework for Enterprise Architecture Design," in *Interface*, vol. 3, pp. 1–4, 2008.
6. S. Roth, M. Hauder, M. Farwick, R. Breu, and F. Matthes, "Enterprise Architecture Documentation: Current Practices and Future Directions.," *11th International Conference on Wirtschaftsinformatik*, no. March 2013, pp. 1–15, 2013.
7. K. Winter, S. Buckl, F. Matthes, and C. M. Schweda, "Investigating the State-of-the-Art in Enterprise Architecture Management Methods in literature and Practice.," *Mcis*, 2010.
8. S. Kaisler, F. Armour, and M. Valivullah, "Enterprise Architecting: Critical Problems," *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, vol. 00, no. C, pp. 1–10, 2005.
9. C. Lucke, S. Krell, and U. Lechner, "Critical Issues in Enterprise Architecting – A Literature Review," in *16th Americas Conference on Information Systems (AMCIS) 2010*, pp. 1–11, 2010.
10. M. Hauder, F. Matthes, and S. Roth, "Challenges for automated enterprise architecture documentation," in *Lecture Notes in Business Information Processing*, vol. 131 LNBIP, pp. 21–39, 2012.
11. The Open Group, *TOGAF Version 9*. 2009.
12. H. Florez, M. Sánchez, and J. Villalobos, "A catalog of automated analysis methods for enterprise models," *SpringerPlus*, vol. 5, no. 1, p. 406, 2016.
13. M. Jackson, C. Delp, D. Bindschadler, M. Sarrel, R. Wollaeger, and D. Lam, "Dynamic gate product and artifact generation from system models," in *Aerospace Conference, 2011 IEEE*, pp. 1–10, March 2011.
14. C. Wang, H. Li, Z. Gao, M. Yao, and Y. Yang, "An automatic documentation generator based on model-driven techniques," in *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, vol. 4, pp. V4–175–V4–179, April 2010.
15. M. Buschle, H. Holm, T. Sommestad, M. Ekstedt, and K. Shahzad, "A Tool for Automatic Enterprise Architecture Modeling," *Is Olympics: Information Systems in a Diverse World*, vol. 107, pp. 1–15, 2012.
16. S. Grunow, F. Matthes, and S. Roth, "Towards automated enterprise architecture documentation: Data quality aspects of SAP PI," in *Advances in Intelligent Systems and Computing*, vol. 186 AISC, pp. 103–113, 2013.
17. M. Buschle, M. Ekstedt, S. Grunow, M. Hauder, F. Matthes, and S. Roth, "Automating Enterprise Architecture Documentation using an Enterprise Service Bus," *Americas conference on Information Systems*, vol. 6, pp. 4213–4226, 2012.