An Approach for Creating and Managing Enterprise Blueprints: A Case for IT Blueprints

Pedro Sousa[1,2,3], José Lima[1], André Sampaio[1], and Carla Pereira[2,3]

[1] Link Consulting, SA
[2] Instituto Superior Técnico (IST), Technical University of Lisbon
[3] Inov-Inesc, Lisbon, Portugal
{pedro.sousa,jose.lima,andre.sampaio}@link.pt,
carla.pereira@inov.pt

Abstract. One important role of Enterprise Architecture aims at modeling enterprise artifacts and their relationships, ranging from the high-level concepts to physical ones such as communication networks and enterprise premises. As it is well known, these artifacts evolve over time, as well as their relationships. The dynamic nature of such artifacts has been a difficulty not only in modeling but also in keeping enterprise blueprints updated. This paper presents our approach to handle blueprints of the Enterprise Architecture, based on several years and projects in large organizations, both in the financial and telecommunication industry.

We started by considering "projects" as the changing elements of Enterprise artifacts and achieve a scenario where blueprints are automatically generated and updated, and a time bar allows traveling from the past (AS-WAS), to the present (AS-IS) and to the future scenarios (TO-BE). The paper also presents an overview of the underlying model, the applied methodology and the blueprints that we found to be a valuable instrument amongst elements of different communities: Project Management, IT Governance and IT Architecture. In spite that the cases studies are targeted to the IT domain, the lessons are valid for other architectural areas.

Keywords: IT Blueprints, Enterprise Architecture, IT Governance.

1 Introduction

As in any complex system, enterprises would be better understood if one could have a blueprint (schematic representation) of the artifacts that constitute the organization and their relations. In the IT domain, blueprints have always been perceived as an important asset, especially by the IT Architecture teams or departments. In fact, many companies have been trying to make blueprints of the IT landscape, from high level maps to detailed ones. But the truth is that companies fail to have such maps, claiming that update costs are simply too high given the rate of changes of the organization artifacts.

Blueprints come in many shapes and detail levels. In order to clarify what we mean by a "blueprint", we present two examples of different level of detail and scope. On the left side of figure 1, we present one example of a very high level business view of retail banking, following the classification schema proposes in [1]. On the right side we present a typical application landscape with interactions between applications.

We have been doing professional consultancy in the domain of Enterprise Architecture for over a decade and have found many other reasons, other than costs, that are preventing companies to have blueprints up-to-date. Probably the most common and simple reason is that, quite often, IT professionals assume that the adoption of a given modeling notation (i.e. UML) is enough to start producing blueprints. But it is well known that, in fact, behind each blueprint there is a *theory* that defines the governing rules, a *model* that identifies the properties and semantics of artifacts, and a *notation* to graphically express such artifacts, and also a *problem*, to provide a purpose of each blueprint and thus, making it possible to decide what artifacts should appear in each one [2].

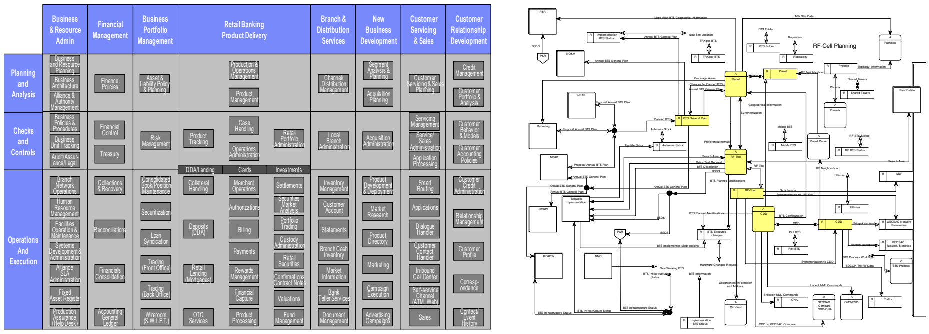
Business & Resource Admin
Financial Management
Business Portfolio Management
Retail Banking Product Delivery
Branch & Distribution Services
New Business Development
Customer Servicing & Sales
Customer Relationship Development
Planning and Analysis
Checks and Controls
Operations And Execution
DDA/Lending
Cards
Investments
RF-Cell Planning

Fig. 1. Blueprint Examples

This paper is not about the right or the best *theory*, *model* or *notation* for enterprise architecture. It is about what needs to be added to known theories, models and notations to allow blueprints to be systematically produced and maintained in real organizations.

This paper is structured as follows: in the next section we present a more detailed view of the problem, trying to narrow down the key issues behind the difficulty of creating and maintaining blueprints; in section 3 we present the related work in the scope of Enterprise Architecture; in section 4 we present some aspects of the theory and model necessary to support our approach; in section 5 we present the methodology used in real cases; in section 6 we present the BMS, our software solution for of blueprint management, and finally, we conclude in section 7.

2 Problem Clarification

In order to keep blueprints up-to-date, one needs two basic things:

- Information about what has changed.
- Rules to update the blueprints accordingly.

Regarding the first issue, we have found that such information normally exists in the notebooks and agendas of the IT professionals, namely of those that were involved in the changes. However, these notes were written with the purpose to refresh the memory of those who have written them. We could envisage that the problem would be solved if IT professionals use standard concepts and notations in their personal notes and published them into a common repository at the end of the day. But in medium and large size organizations, IT reality is changed mostly by IT projects, and therefore IT projects are the best entity to report back the changes in some normalized form.

But managed IT projects do have a work plan to produce the intended artifacts. For example, if a project intends to make system *A* that sends data to an existing system *B*, then both the system *A* and the data flow to *B* should be referenced in the project plan details.

Therefore, the real questions come down to:

Q1. Are the IT artifacts referred in IT project planning the ones that appear in IT architectural blueprints?
Q2. Are project plans up-to-date enough so they can be a trustful source of information?

A second concern, still related with the first issue, is the extra complexity that enterprise wide architecture blueprints may bring to IT projects. If fact, since enterprise wide blueprints tend to focus on global views of IT artifacts, rather than on the subset of artifacts that are relevant for a given project, reading and updating enterprise blueprints is more complex than it could be. For example, if a project only has to be concerned with 100 artifacts out of a total of 1000 artifacts, then the project should handle a 100 artifact´s blueprint rather than a 1000 artifact´s blueprint. Given that IT projects are mostly stressed for time and budget, such additional complexity is also a critical aspect, in particular if they have to keep blueprints updated.

Therefore, other relevant questions are:

Q3. Can we provide to each IT project a blueprint no more complex than it should be?
Q4. Can changes in such blueprints be automatically propagated back to enterprise-wide and more complex blueprints?

Regarding the second issue, the fact is that today´s blueprints are mostly a hand-made piece of art, made with a mouse; likewise a painter uses a brush. A blueprint is mostly a personal achievement, not an industrialized result. Most concepts depicted in IT blueprints such as Information Systems, Applications, Platforms, Application Components, Nodes, amongst many others, are not at all clear amongst IT professionals, as one could expect. Unlike with professionals from other engineering domains, when faced with a given reality, different IT professionals name and classify the same artifacts differently. The use of a common and well known notation such as UML does not help at all on the fundamental aspects, because the semantic meaning of symbols is not defined, and must be a personal decision. Furthermore, there is no clear definition to what artifacts should be represented in each blueprint. Once again, it is up to the designer to decide what is relevant and what is not relevant to represent/model in each blueprint.

Therefore, regarding the second issue, the relevant questions are:

Q5. To what level of detail/semantic should one define artifacts and concepts?
Q6. How can one decide which architectural blueprints to use and what artifacts should each blueprint represent?

3 Related Work

Blueprints and schematic representation are common ways of communication between people, namely to express an architectural description of things, like a system, an object, a model or, in our case, an Enterprise.

As clearly described in the IEEE1741 [2], as well as in other works [1, 3, 4], behind a architectural description, there is always a set of other concepts that must be defined to ensure that the architectural description is properly understood, namely a notation, a concern, a model, a view and viewpoint. A considerable amount of effort has been put in the development of these concepts, and in fact most EA frameworks do propose a set of concerns and corresponding views and viewpoints, a model, concepts and in some cases even a notation [2, 3, 5-9]. It also commonly assumed that, architects are able to produce blueprints and models based on proposed EA frameworks, which in turn, sustain developments in many other areas, such as strategic alignment [10], IT Governance [11], Application and Project Portfolio Management [12-14] to name a few. In [15], one my find an overview of uses and purposes of EA.

But the assumption that an architect (or an army of them) is able to model the organization is a valid one only for the domains where the change rate is in fact low, as for example the Enterprise Ontology [16], the enterprise organic structure or the enterprise vision and mission. For the domains with a high rate of changes, such as business processes or IT, one cannot assume to have valid blueprints, simple because the effort to keep them up-to-date is too high. In other words, the current EA frameworks and models do not capture the dynamic nature of enterprises.

To our knowledge, the problem of creating and keeping blueprints up-to-date in an automatic manner has not been an issue in the EA community. As referred in chapter 2, two main issues need to be addressed.

The first - information about what has changed – concerns mostly with establishing an information flow, from the ones making the changes in the IT, to the ones updating the blueprints. We found related work in the area of IT portfolio and project management [12-14], where EA is used as a source of information to feed decisions in IT portfolio and project management, but they do not established a detailed update from IT projects to EA.

The second - the rules to update the blueprints accordingly – concerns mostly on how to overcome is the lack of semantic behind common notations (as UML, SysML, IDEF, amongst many others). As Mark Lankorst refers, they are mostly symbolic models, not semantic ones [3]. Archimate [3] moves one step forward by providing a stronger model, but true semantics requires a theory, as the Ψ-theory [17], to sustain models and methodologies and well understood blueprints. In what concerns the IT, we have knowledge of a semantically sound model, although some progress has been made [18].

4 Fundamentals of Our Approach

The following description intends to give the reader the basic grounds for the practical work presented in this paper. It does not intend to be a full explanation of the model, and many aspects of it are not referred.

Let an Enterprise E be modeled as a graph G of artifacts and their relationships, the vertices and the edges of the graph accordingly. Let A and R be the set of all artifacts and relationships[1], accordingly. Since this graph changes over time, let $G_t\ (A\ ,R)$ represent the value of G at time t, where t is a discrete variable corresponding to the succession of states $\{G_0 ,\ldots, G_n\}$ of $G_t\ (A\ ,R)$.

Let each artifact $a \in A$ have a type $y \in \Gamma$, where Γ is the set of all types. The statements "*a IsA y*" and "*y= type (a)*" state that *y* is the type of artifact *a.* A type defines the properties and possible values for artifacts of that type. Relationships are typed after the types of connected vertices[2][19]. For clarity sake, we´ll represent types in italic and with the first letter in capital (e.g. *Type*) and instances in italic.

We start by introducing two fundamental types of Γ:

- *Blueprint*, whose instances contain references to others artifacts. A given *artifact* is represented on a given *blueprint* if graph *G* holds as a relation between them.
- *Project,* whose instances contain references to artifacts related with the project.

We further define the state_of_existence of all artifacts other than *Blueprint* as one of the following states:

- **Conceived:** If it is only related with *blueprints.*
- **Gestation:** If it is related with alive *projects* and is not related with any other artifacts other than *blueprints.*
- **Alive:** If it is related with other artifacts in the alive state. This means that it may act upon other artifacts in conceived, gestation or alive states.
- **Dead:** If it is no longer in the alive state[3].

Let *Project.aliveList* and *Project.deadList* be the list of artifacts to become alive and dead during the project.

We now come back to the set of states $\{G_0 ,\ldots, G_n\}$ of $G_t\ (A\ ,R)$ that represent the sequence of states of the organization, and consider the sequence of alive state of the organization $\{G_{A0} ,\ldots, G_{An}\}$. Whenever a *project* ends, the set of alive artifacts in the enterprise changes from a state G_{An} to the a state G_{An+1} where

$$G_{A_{n+1}} = G_{A_n} \cup project.aliveList \setminus project.deadList$$

[1] Notice that between two given artifacts there may exist relationships of different types.

[2] The simple fact that relationships have a type means that: (i) the set Γ includes also the types of all relationships, and (ii) the relationships are in fact "processors" in the context of General System Theory [19], as are the artifacts.

[3] In the context of General System Theory [19], artifacts in the dead state are necessary passive objects/systems, regardless of the level they had while alive (from 1-passive- to 9 - finalizing).

This allows us to move back and forth in time from the past to the present and from the present to the future. In particular, it allow us to define the *ToBe(t)* as the set of alive artifacts at time *t* based on the *AsIs* state, namely:

$$AsIs = G_{A_s}(A,R);$$
$$ToBe(t) = G_{A_s}(A,R) \cup \{ p.aliveList \mid \forall p{:}\, p\ IsA\ Project \wedge s \leq p.endTime \leq t \} \setminus \{ q.deadList \mid \forall q{:}\, q\ IsA\ Project \wedge s \leq q.endTime \leq t \};$$

Where, s is the time of *AsIs* state (presumably the current date) to and *p.endTime* is planned time for the project *p* to end. It is also clear that the number of possible states of G_A between any given points in time corresponds to the number of projects that end between those two points.

We now focus on a particular type of artifacts: the artifacts of type *System.* We start by clarifying the relations[4] "IS_PART_OF" and "ACTS_UPON", according to notation used in [16]:

- IS_PART_OF represented as " $\prec$", is a relation between two artifacts such that: for any two artifacts (x, y) , $x \prec y$ if and only if, in order for y to be alive *x* must also be alive and *x* cannot be part of another artifact that *y* is not also part of.
- ACTS_UPON represented as "$\rightarrow$", is a relation between two artifacts such that: for any two artifacts(x, y) , $x \rightarrow y$ if and only if, *x* causes changes in state/behavior of artifact *y*. This implies that *y* is not in the dead state.

Following the description presented in [16], a system σ is defined by its *Composition*, *Environment* and *Structure*:

- The *Composition* C of a system σ is the set of artifacts that:

$$C(\sigma) = \{x{:}\, x\ IsA\ Component \wedge x \prec \sigma\}$$

- The Environment E of a system σ is the set of artifacts that:

$$E(\sigma) = \{x{:}\, x \notin C(\sigma) \wedge \exists y{:}\, y \in C(\sigma) \wedge (x \rightarrow y \vee y \rightarrow x)\}$$

- The Structure S of a system σ is the set of related artifacts defined as:

$$S(\sigma) = \Big\{ < x,y > \mid (x \rightarrow y \vee y \rightarrow x) \wedge \big(x,y \in C(\sigma) \vee (x \in C(\sigma) \wedge y \in E(\sigma))\big) \Big\}$$

Notice that, in spite that the above expressions are time invariant, the set of artifacts that belong to $C(\sigma), E(\sigma)$ and $S(\sigma)$ do change over time, since the IS_PART_OF and ACTS_UPON relations are defined over the alive state of artifacts, which change with the projects alive and dead lists.

After the definition of artifacts of type *System*, we are able to define its basic views (Organic, Environment, Composition and Structure), used by our blueprint engine to produce the blueprints:

[4] Both relations are transitive, but we will not explore such properties in this paper. We consider only the direct dependencies.

- The Organic View V_{Org} depicts the artifacts of subtype of *System* in a hierarchically manner according to the value of a given property P_{org}.

  $$V_{ORG}(T_{Org}, OrgHierarchySet, t) = \{Depiction(\sigma) \mid \forall\, \sigma{:}\ type(\sigma) \in T_{Org} \ \wedge\ P_{org}(\sigma) \in_t OrgHierarchySet\}$$

  Where is the set of subtypes that will be depicted, *OrgHierarchySet* is the hierarchy values according to which artifacts will be graphically arranged, and $P_{org}(\sigma) \in_t OrgHierarchySet$ states that the value of P_{org} of property σ that must be one of possible state defined in *OrgHierarchySet* at time *t*.
- The Environment View V_{ENV} of a system depicts the artifacts that belong to the Environment of that system.

  $$V_{ENV}(\sigma, T_{ENV}, t) = \{Depiction(x) \mid \forall\, x{:}\ type(x) \in T_{ENV} \ \wedge\ x \in_t E(\sigma)\}$$

  Where T_{ENV} is the set of types that will be depicted, and $x \in_t E(\sigma)$ states that artifacts must belong to environment of σ at time *t*.
- The Composition View of a system depicts the artifacts that belong to the system *Composition* and their relationships.

  $$V_{COMP}(\sigma, T_{STR}, t) = \{Depiction(x) \mid \forall\, x{:}\ type(x) \in T_{COMP} \ \wedge\ x \in_t C(\sigma)\}$$

  Where T_{COMP} is the set of types that will be depicted, and $x \in_t C(\sigma)$ states that artifacts must belong to Composition of σ at time *t*
- The Structure View of a system depicts all the relationships between any two artifacts related with the system.

  $$V_{STR}(\sigma, T_{STR}, t) = \{Depiction(x,y) \mid \forall\, x,y{:}\ type(x), type(y) \in T_{STR} \ \wedge\ (x,y \in_t S(\sigma) \vee (x \in_t S(\sigma) \ \wedge\ y \in_t E(\sigma))\}$$

  Where $T_{STR}$is the set of types that will be depicted, and $x,y \in_{ti} S(\sigma)$ states that artifacts *x, y* belong to Structure of σ at time *t.*

4.1 The Case for IT Artifacts

The application of the above model implies the definition of type hierarchy Γ, making clear the relevant concepts and their relationships. As in most Enterprise Architecture frameworks, IT related types include concepts, such as: *BusinessProcess*, *InformationEntity*, *Stakeholder*, *Repository*, *DataFlow*, *Service*, *Domain*, *Solution*, *Application*, to name a few. In most cases, these concepts are defined too loosely, to ensure a full and unique understanding amongst different persons, especially if they come from different communities. Therefore, the further we close artifact concepts the easier the communication gets. We present the example of applications and their components.

Let application to be a *System*, whose *Composition* is a set of artifacts of type *AppComponent*, being the later a subtype of *Component,* which is in turn a subtype of *System*[5].

[5] Therefore *AppComponent* is also a *System.* The recursive aspects will not explored in this paper. Suffice is to say that *Domain*, *Solution*, *Application* and *AppComponent* are all subtypes of *System* and are implemented as instances of the same class.

According to definition of *System,* many sub-systems can be considered and defined for a particular system. Therefore, the identification of *AppComponent* of a given application is normally a personal decision, with unclear rules and assumptions. Such degree of freedom makes blueprints unclear because one does not know the reasons why a particular set of *AppComponents* were considered, instead of another set. Thus, in order to produce blueprints more clear, we were forced to establish additional rules guiding the finding of the proper set of *AppComponents*.

We present a definition that has proven to be useful and simple. Let *Application* to have a layered structure[6]. For the sake of simplicity, let us assume the traditional layers: *UserInterface*, *BusinessLogic*, *Integration* and *Data*. Let *ITPlatform* artifact to be also of type a System.

A given artifact c is a *AppComponent* of given *Application* a is and only if it complies with the following conditions:

(i) c is a subsystem of a. This means that[16]:

$$C(c) \subseteq C(a)$$

$$E(c) \subseteq C(a) \setminus C(c) \cup E(a)$$

$$S(c) \subseteq S(a)$$

(ii) c is related with one and only one layer of a. Let p,q be two application layers of a . This means that:

$$\forall x : x \in E(p), \nexists q \neq p : y \in E(q) \wedge y \in C(c)$$

(iii) c is related with one and only one *ITPlatform*, the platform where the *AppComponents* perform/execute. The formulation is the same as the previous, considering p,q to be *ITPlatforms.*

Altogether, these conditions state that an application component is a system executing on a given platform to perform one application role (*UserInterface*, *BusinessLogic*, *Integration* and *Data*). If a more fine grained rule is required, one normally add fourth rule regarding business functions, increasing further the definition of a component: an application component is a system executing on a given platform to perform one application role of a business function.

We now consider some simplifications based on two key aspects that we found to be true in some large companies, in which we found that:

- IT project management is a mature discipline, meaning that *Project* artifacts are well established and managed ones.
- IT production environment is a managed asset, meaning that placing IT artifacts into production is also a mature discipline.

Under such conditions, it is a reasonable assumption that an IT artifact:

- Is in the <u>gestation</u> state when it is being developed within a given IT project.
- It becomes in the <u>alive</u> state, when they are placed onto the production environment as a result of some IT project.

[6] A similar construction could be done for a service oriented structure.

- It becomes in the dead state, when they are removed from production environment as a result of some IT project.

Regarding IT projects, we also consider that:

- Artifacts in project alive and dead lists become alive or dead at the end of the project.
- Project alive and dead lists are up-to-date at least in two moments in time: when the project starts and when the project ends. In the first moment, these lists are a promise of what the project intends to do, and in the second moment theses lists are what the project actually did.

This means that project alive and dead lists will be considered as *ToBe* until the project has ended, and will become *AsIs* at project termination, accordingly to the expressions stated before.

5 Methodology

In order to apply the previous model to organizations we follow the following phases:

1. **Problem Statement.** We identify the key concerns of each participating communities, namely IT projects, IT Architecture and IT Governance, so that blueprints can be designed in a way such that they are useful for the above communities, and concepts may be defined in the appropriate level of detail. Thus, in this first phase, we establish desired goals and outputs of the project.
2. **Information and Processes analysis.** One analyzes the actual processes of Project Management, IT architecture and IT Governance, in the organization and the information exchanged between them. This allows us either to confirm the expectations rose in the previous phase or to adjust to them accordingly.
3. **Artifact definition.** One revises the definition of the artifacts required to address the concerns identified in the previous step, and clarifies the source information and the "master catalogue" for each concept.
4. **Blueprints definition.** We design the blueprints according to the needs of the different communities. For each concern they have we propose a blueprint and identify the artifacts types and relations that will be depicted in it. Obviously many concerns are left unanswered due to the lack of information.
5. **Notation definition.** We define a symbol for each artifact and/or relation. Whenever we have encountered a similar concept in UML, we have adopted the UML symbol for such a concept.
6. **Information and Processes Improvements.** We propose changes to IT projects, IT Architecture and IT Governance processes, and define the contents of the documents handled amongst these processes. Process optimization results mainly from a better communication and information flow amongst these different communities. Two important documents are related with the project initiation and conclusion:
   - The way project plan can be processed to identify project dead and alive lists. This can be done by making sure project plans use the proper artifacts names or IDs and uses know keywords to identify if the artifacts are being used, created, updated or deleted within that project.

  - o The description of how a project should present the architecture of the things it is intended to develop, when project starts, or the thing it has developed, when the project ends. This summarizes results of phases 3, 4 and 5.

7. **Automation.** We use our Blueprint Management System (BMS) to gather information from different sources and to produce blueprints and other documentation exchanged between the different processes. The BMS generates office documents automatically, simplifying communication between people.

6 The Blueprint Management System

We now present our Blueprint Management System (BMS), a software solution that implements our approach for generation and maintenance of blueprints.

The BMS collects information from different sources:

- Project Management systems, where information about existing projects and corresponding IDs, dates, resources involved and the artifacts CRUD lists[7], if available in project plans. For example, for each application component related to a project, the following information should be provided: "component; application; layer; platform; CRUD", where the CRUD indicates the action of the project on that component.
- Imported csv files or via a web user interface, where project teams can upload the above information in a textual form, if not existing elsewhere.
- Operational systems, such active nodes from a CMDB or the active services from a SOA service registry. This is information about the production environment that can be used to complement the information gathered from IT projects or to provide alerts. For example, if a project intends to use a service that it does not exist in the production environment (is not alive), nor in the create list of on-going projects (is not in gestation), a warning is issued. As another example, the BMS can query a development platform and find out that a given artifact was created within a given project area, and issues a warning if that artifact was not in project create list.

For each artifact type, the BMS can act as a master or a slave catalog. In the last case, the BMS collects information from existing catalogs periodically, normally in a daily basis. Based in the new information collected the BMS generates the blueprints that may have changed since the last generation.

The blueprints are created both as images and as interactive objects. Regarding the first, users can upload an office document (a Microsoft PowerPoint, Word or Excel format) with references for the desired blueprints and BMS returns the document with the requested blueprints as images. Regarding the interactive objects, they allow both the navigation between different blueprints, since symbols are in fact links to other blueprints, and a querying mechanism based on the values of artifacts properties, that can change the depiction or color of artifacts matching the query.

[7] For the sake of simplicity, the BMS considers that projects have four artifacts lists: Create (alive), Read (used), Update (changed), Deleted (dead).

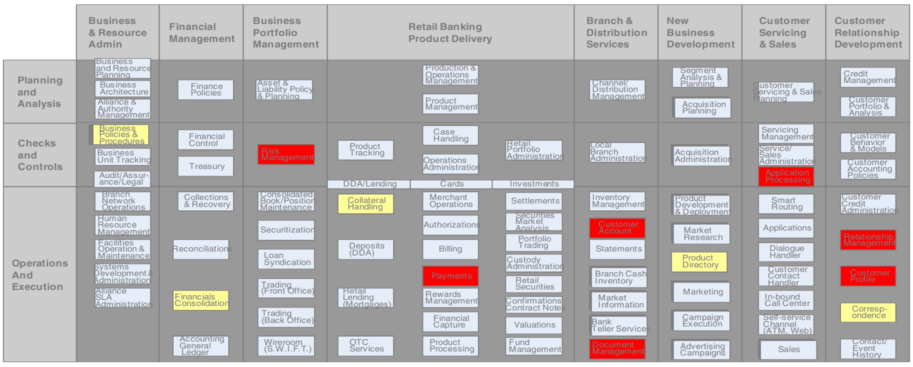

Fig. 2. Governance Blueprint

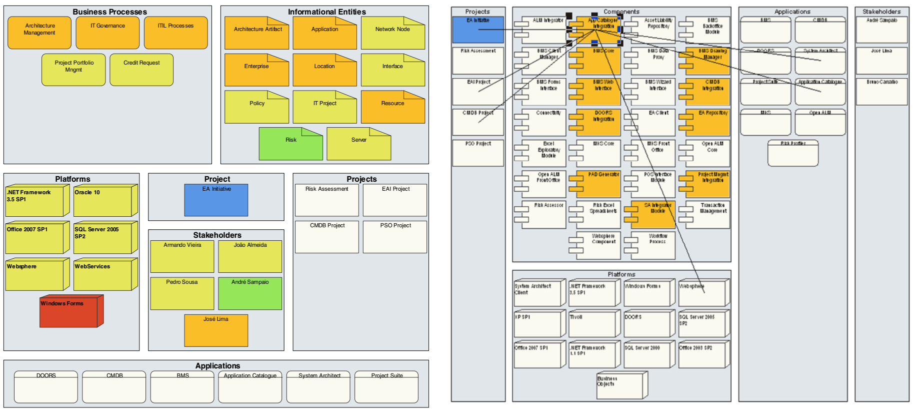

Fig. 3. Context Blueprint and Project Impact Blueprint

We now present some examples of blueprint generated. The figure 2 presents a Governance Blueprint, produced using the *Organic View* over the type hierarchy {business domain, solution, application}. In this blueprint[8], columns are broad business functions, lines are management level (plan, control, execute) and rectangles inside matrix cells are IT solutions, each aggregating several applications (not depicted). However, the solutions have a color code, according to the delays of on-going projects affecting applications within that solution. One project may affect several applications in several solutions.

The Context Blueprint presented in the left side of figure 3 has a given project as subject and answers two questions: (i) what artifacts affect that project, and (ii) what artifacts are affected by that project. The blueprint presented was produced with $V_{ENV}(\sigma, T_{ENV}, t)$ where $\sigma = Project$ and *T_ENV ={Application, ITPlatform, InformationEntity, BusinessProcess, Stakeholder, ITProject}* with a color filter according to the CRUD relation between each artifact and the project[9].

[8] This blueprint implements the Activity Based Retail Bank Component Map proposed in [1].
[9] The color code may have different semantics for different types of artifacts related.

This blueprints states that project in blue (in the center) intents to remove the platform in red, which is being used by the applications shown in the bottom , and this has an effect on the business processes and information entities, as well as on four other projects.

The details of the dependencies amongst these five projects are detailed in the Project Impact blueprint presented on the right side of the figure 3, where the application components (in UML notation) that are involved in more one project are presented in orange. The image shows the selection of one particular component, and the lines reveal the actual projects where conflict may arise.

The left side of figure 4 presents a Structure Blueprint applied over a given application. This blueprint answers 3 questions: (i) the components of the application; (ii) how they are structured into application layers, and (iii) the platform each component executes. This blueprint is a combination of *Composition* and *Structure Views*. The right side of this Blueprint was produced as $V_{COMP}(\sigma, T_{COMP}, t)$, where

$$\sigma = \text{"}Aplication\ and\text{"}\ T\_COMP = \{UserInterface,\ Integration,\ BusinessLogic, Data\}$$

The left side of this blueprint was generated as $V_{STR}(\sigma, T_{STR}, t)$, where σ is an actual application and

$$T_{STR} = \{(Platform, UserInterface),\ (Platform, Integration),\ (Platform, Business),\ (Platform, Data)\}$$

The right side of figure 4 presents an Integration Blueprint, which depicts how the components are integrated within the organization IT and among themselves. The components appear in UML notation and within the execution platform, and the data flows are links to the typed objects holding the detailed description of the information

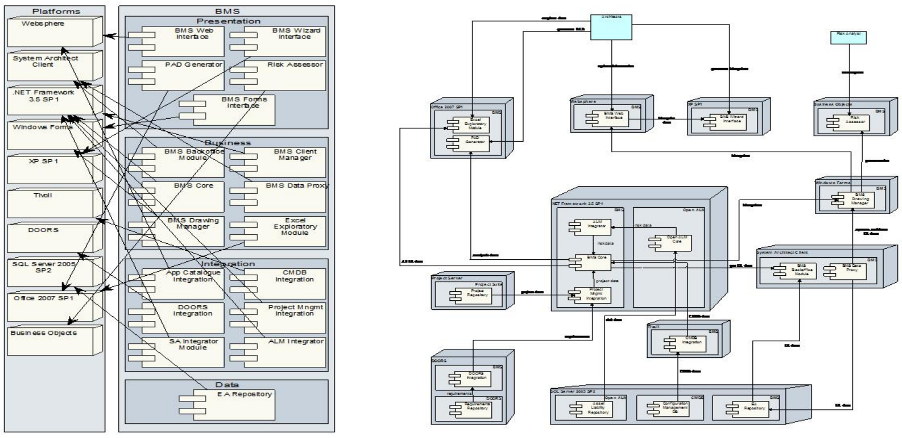

Fig. 4. Structure and Integration Blueprints

and control (push, pull, message, shared database, etc) of that flow. This Blueprint was generated as $V_{STR}(\sigma, T_{STR}, t_i)$, where

$$T_{STR} = \{ (IntegrationComponents, Component), \\ (Components,\ Platform),\ (Components, Aplication) \}$$

and σ is a given application.

The above blueprints are very a small sample of the variety of blueprints that may be produced from the basic views (Organic, Environment, Composition and Structure) described in chapter 4.

The blueprints can be generated either with a static or a dynamic contents. In the last case, a time bar similar to the one presented in figure 5 is added to the top of each blueprint. The time bar has two movable buttons that define a period in time to filter out the displayed artifacts, namely only the artifacts that are in the alive state in that period (01/01/2008 and 05/21/2008 in the example of figure 5) are displayed.

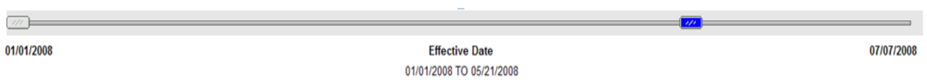

Fig. 5. The Time Bar

By moving both buttons to the same moment in time, only the artifacts that are in the alive state at that time will be presented. The default position of both buttons is set for today, so the AS_IS situation is presented by default. By moving both buttons forward in time one gets the TO_BE state, as foreseen according to the plans of the projects to be completed before that time.

7 Conclusions

We have successfully managed to have a full set of architectural blueprints being automatically generated on a weekly basis based on information retrieved from IT projects plans, and being used by users of different communities: IT project management, IT Architecture and IT Governance.

The use of blueprints as a common language had several effects in the organization. One unexpected example is the fact that IT project managers changed the way they work. Before, they did project budgeting and planning first and then they think about the architectural representations. Today they do the architecture in the first place, by sending textual data to BMS and getting back the corresponding blueprints, and only then they compile the cost and plan to make each architectural artifact into a global project cost and plan. This answers positively to question Q1 of section 2. Furthermore, it only states that IT projects are doing the same things as what IT architects are architecting and as what IT governance is governing. We believe this is one step forward in the organization self-awareness [20], which has indeed produced by itself unexpected changes.

Regarding question Q2 of section 2, the answer tends to be "yes" when the project starts and ends, and "not as it should" during project execution. This has an impact, especially in long projects, because quite often projects do different things than what they had initially planned, and those changes are not feed into the blueprint analyses and queries done before the end of the project.

Regarding questions Q3 and Q4 of section 2, the answer is definitely "yes". Projects have context, structure and integration blueprints involving only the artifacts relevant to the project, and changes to these blueprints are compiled back to the enterprise wide blueprints.

The answer to question Q5, is simple in theory: as detailed as it needs to be so that everyone can understand the same thing, but in practice one may not achieve a satisfying definition for all artifacts, especially for the non IT domains.

Finally the answer to question Q6 is based on the clarification of the concerns/ stockholders as recommended in [2]. Rather than having a few complex and multipurpose blueprints, one should aim at many blueprints as simple as possible, each answering to one or two simple questions that is useful for a given concern/ stockholder.

We present what we have found in actual companies in banking, telecommunication and retail industries. In some cases, we aim at zero effort blueprints, since they are automatically generated based on information that flows between different communities in the organization, mostly between IT Architecture, and IT Project Management.

So far, we have only experiment our approach in the IT domain and in large organizations. However, the usage of a similar approach in the business domain or even in the IT domain of small organizations may face other difficulties. In fact, our approach requires that artifacts become alive and dead at well known events. In case of IT of large organization, these events occur when artifacts are place into the production environment via well establish procedures. This may not be the case for small companies, where production environment is not so controlled, and for sure, is not the case for business processes domain, where each employee is in fact part of the production environment, making almost impossible to trigger well known events in the organization when business artifacts become alive.

Finally it is worth to say that this work did not evolve as presented here. Even though it had a few cycles of experimentation and formalization, the bulk of formalization presented in chapter 4 did come after the experimentation. We envisage further integration with general systems theory [19] to better address the ideas of control and changeability of organizations as envisaged in [21, 22].

References

[1] IBM, Retail Banking - Business Componet Map. White Paper IBM Global Solution Center (2007), `http://www.ibm.com`

[2] IEEE Computer Society, IEEE Std 1471-2000: IEEE Recommended Practice for Architecture Description of Software-Intensive Systems. IEEE, New York (2000)

[3] Lankhorst, M.: Enterprise Architecture at Work: Modelling, Communication and Analysis. Springer, Heidelberg (2006)

[4] Pereira, C.M., Sousa, P.: Business Process Modelling through Equivalence of Activity Properties. In: ICEIS 2008, Barcelona, Spain, pp. 137–146 (2008)

[5] Sousa, P., Caetano, A., Vasconcelos, A., et al.: Enterprise architecture modeling with the UML 2.0. In: Rittgen, P. (ed.) Enterprise Modeling and Computing with UML, pp. 67–94. Idea Group Inc. (2006)

[6] Zachman, J.: A Framework for Information Systems Architecture. IBM Systems Journal 26(3), 276–292 (1987)

[7] Open Group. The Open Group Architectural Framework (TOGAF) - Version 8.1 (June 20, 2005), `http://www.opengroup.org/togaf/`

[8] Macaulay, A., Mulholland, A.: Architecture and the Integrated Architecture Framework, Capgemini (2006)

[9] Stader, J.: Results of the Enterprise Project. In: Proceedings of Expert Systems 1996, the 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems, pp. 888–893 (1996)

[10] McFarlan, F.W.: Portfolio approach to information systems. Harvard Business Review, 142–150 (September-October 1981)

[11] Calder, A.: IT Governance: A Pocket Guide. IT Governance Publishing (2007)

[12] Makiya, G.: Integrating Enterprise Architecture and IT Portfolio Management Processes. Journal of Enterprise Architecture 4(1), 27–40 (2008)

[13] Walker, M.: Integration of Enterprise Architecture and Application Portfolio Management (17-02-2009, 2007), `http://msdn.microsoft.com/en-us/library/bb896054.aspx`

[14] Lagerstrom, R.: Analyzing System Maintainability Using Enterprise Architecture Models. Journal of Enterprise Architecture 3(4), 33–41 (2007)

[15] Op't Land, M., Proper, E., Waage, M., et al.: Enterprise Architecture:Creating value by informed Governance. Springer, Berlin (2009)

[16] Dietz, J.: Enterprise Ontology: Theory and Methodology. Springer, New York (2006)

[17] Reijswoud, V., Dietz, J.: DEMO Modelling Handbook, vol. 1, Version 2.0, Delft, Department of Information Systems, Delft University of Technology, The Netherlands (1999)

[18] Vasconcelos, A., Sousa, P., Tribolet, J.: Information System Architecture Metrics: an Enterprise Engineering Evaluation Approach. Electronic Journal of Information System Evaluation 10 (2007)

[19] Le Moigne, J.-L.: A Teoria do Sistema Geral - Teoria da Modelização (From the original la théorie du système général - Théorie de la modélisation), Instituto Piaget (1997)

[20] Magalhães, R., Zacarias, M., Tribolet, J.: Making Sense of Enterprise Architectures as Tools of Organizational Self- Awareness. Journal of Enterprise Architecture 3(4), 64–72 (2007)

[21] Santos, C., Sousa, P., Ferreira, C., et al.: Conceptual Model for Continuous Organizational Auditing with Real Time Analysis and Modern Control Theory. Journal of Emerging Technologies in Accounting 5, 37–63 (2008)

[22] Matos, M.: Organizational Engineering: An Overview of Current Perspectives, DEI, IST-UTL, Lisbon (2006)