PATETA: PATterns for EnTerprise reference Architectures

A method to find EA patterns

Yesika Reinolds
Instituto Superior Técnico and INESC-ID
Lisbon, Portugal
reinolds.yesika@gmail.com

André Vasconcelos
Instituto Superior Técnico and INESC-ID
Lisbon, Portugal
andre.vasconcelos@ist.utl.pt

***Abstract* — The goal of this article is to find patterns and anti-patterns for Enterprise Architectures (EA) in order to help developing projects. In this paper we address several scientific and practical problems, including: the nonexistence of patterns covering all architectures bellowing to EA, the nonexistence of a catalog with patterns and anti-patterns and the nonexistence of methods to discover patterns in architectures.**

We developed a method named RCGD (Revision, Comparison, Generation and Documentation) which solves the problems mentioned above. We apply our approach in two university's architectures. To do the analysis we design these architectures in a XML file, in order to automate the analysis.

The RCGD method to discover patterns has four phases: revision, comparison, generation and documentation. In the Revision phase we choose the architectures and perform the analysis. The comparison phase involves a set of steps to detect the candidates' patterns using the similarity's results. To calculate the similarity between two objects we divide it into two groups: out-of-the-box and inside-the-box. In the out-of-the-box we analyze external things like layer type and name. In the inside–the-box we analyze the functionalities. For the Generation phase we generate the architectural pattern. Finally, in the documentation phase we document the pattern using a previously defined structure.

Keywords— Enterprise architecture; Method; Pattern; Anti-patterns;Patterns detection.

I. INTRODUCTION

The goal of this article is to find patterns and anti-patterns for Enterprise Architectures (EA) to help in the development of projects in specific industries. Having this aim in mind we propose a method to find patterns. In this paper we address several scientific and practical problems including: the nonexistence of patterns covering all architectures bellowing to EA, the nonexistence of a catalog with patterns and anti-patterns and the nonexistence of methods to discover patterns in architectures.

The reason why we define a method to search for patterns is because existing patterns, like client-server [1], don't cover all existing architectures in EA (Organizational Architecture, Business Architecture and Information Systems Architectures) [1]. We develop a method named RCGD (Revision, Comparison, Generation and Documentation) which can help finding patterns in many EA. With this method, we can obtain patterns covering all EA, create a catalog with patterns and anti-patterns and also we can define a method for patterns' discovery. There is a big need to design patterns for any type of architecture and use these patterns with accumulated knowledgement to get more time, to reduce effort in the development of architectural solutions and to obtain positive results in future projects [2] [3].

The RCGD method to discover patterns has four phases: revision, comparison, generation and documentation. In the Revision phase we choose architecture and perform the analysis.

We apply our approach in two university's architectures, the goal for these two architectures is to update their system using smartcards and recording all data about the students and staff. The architectures used, belong to the same university but they have different lines of business, although they have the same problems and context [4]. We manage to identify three patterns and zero anti-patterns. We didn´t find anti-patterns because the architecture's documentation doesn't give us too much information about its consequences. To do the analysis we design these architectures in a XML file to automate the analysis.

Now we will do an overview about the next sections. In the RCGD method's section we identify and explain all the steps that are a part of it. This method has four phases and we define all of them. In the implementing RCGD section we apply the RCGD method in two university's architectures. We described the decisions made every time we faced a new problem. In this section we also display some results. Finally, the conclusion's section is where we present the conclusions about the work developed.

II. RCGD METHOD

The method to define the steps to discover pattern is renamed RCGD (Revision, Comparison, Generation and Documentation) has four phases In the Revision phase we choose architecture and perform the analysis.

The comparison phase involves a set of steps to detect the candidates patterns, which can be classified as patterns out of the box. In this phase we have three sub-phases: out of the box comparison, out of the box pattern and inside the box comparison. In the sub-phase out of the box comparison we perform a lexical analysis with words used in the object of the architecture and we analyze the relationship between objects that are a part of the architecture. In the sub-phase out of the box pattern the pattern is classified with the denomination stipulated. The sub-phase inside the box comparison consists in identify the functionalities of the objects where the similarity is analyzed and can also generate an architectural pattern and consequently create the reference architecture (RA).

In the next phase Generation we generate the possible architectural patterns and these ones can or cannot be accepted for the organization members responsible for the implementation of the architecture solution. Also in this phase, we have to generalize and to classify the pattern and anti-pattern.

In the documentation phase we document the relevant information that is useful for the transmission of the knowledgement to use patterns and not to use anti-patterns.

A. Analysis Architectures – 1st Phase

In this phase we analyse every single architecture that we need to gather documentation about. This documentation is important to understand the architecture and to gather functionalities for RCGD steps. For any architecture analysis we have to get the problem and the context for the next phases. The gather of functionalities can occur in any time but we need to guarantee that in the phase "Comparison inside-the-box" it has already been done. And we can do it by gathering all architectures or only the objects that compose patterns out-of-the-box.

B. Comparison Architectures – 2nd Phase

In order to compare architectures it's necessary that the two architectures have been designed using the same meta-model.

The architecture needs to be design in the same meta-model and also it needs to have a similar context and a similar problem. If this is not the case, the architectures cannot be compared.

The comparison in this method refers to two characteristics: external and internal. The external characteristics correspond to the objects in the architecture, while the internal refers to the internal things (functionalities). The external characteristics are associated to the "out-of-the-box" steps and the internal ones with the "inside-the-box" steps.

1) Comparison out-of-the-box

In the comparison out-of-the-box we perform the first similarity detection. In this phase it's only compared the layer type, object type, name and the relationship. The name "out-of-the-box" comes from the external analysis made because we don't analyze the functionalities, instead we use the information given from the architecture's structure.

a) Lexical Analysis

In the architecture's scanning there is a need to define what information is required to save for similarity analysis. The information gathered previously is important because we need to have all data to compare to the architectures. For example, if the architectures is designed in ArchiMate, the information about layer type and object type are essential for the comparison. The type of relationship between objects and the object's name in the relation is important to be collected independently of the design language.

b) Discovering relation between objects

In this phase, the similarity between objects' names is calculated. The objects' name is composed by many words and we need to analyze and compare every word.

However before calculating any similarity we need to pay attention to check if the words are synonymous, antonymous, preposition or conjunction. If the words are synonymous, antonymous, preposition or conjunction we don't calculate the similarity because if there are synonymous the similarity is equal to 100% and if the words are antonymous, preposition or conjunction we don't need to calculate nothing.

To verify if two strings are similar we need to choose a similarity algorithm(for example Levenshtein Metric, Smith Waterman, JaroMetric, etc.) . Ahead in the document we will specify our choice.

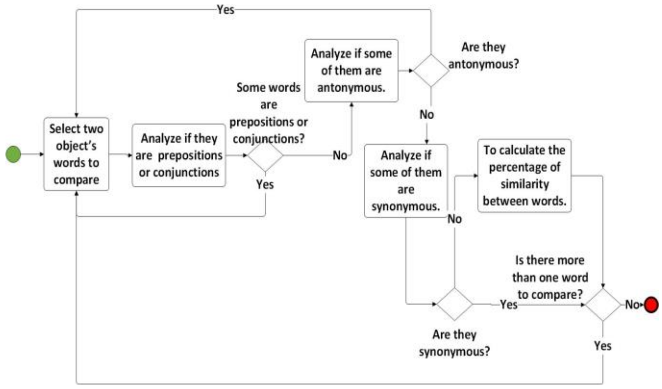

Fig. 1. Workflow to find the similarity between two objects

In order to discover the relationship between two objects we follow the next process (see figure 1). After we select two objects, the flow starts through choosing one word from each object and verifies if any word is a preposition or a conjunction. If some of these words are prepositions or conjunctions they are discarded and it's chosen other combination. If both words are prepositions or conjunctions this combination is discarded and other combination is selected. If it isn't a preposition or a conjunction we need to verify if these words are synonymous or antonymous. If they are synonymous we classify these words with 100% of similarity or if they are antonymous we classified them with 0% of similarity. If it isn't synonymous or antonymous then we calculate the similarity using similarity calculation algorithms.

We are free to choose the algorithm to calculate the similarity like specified previously. We also need to guarantee that every word combination is taken into account. We consider that two object are similar if they have a similarity equal or greater than 50%.

To discover the relationship between object we need to follow the next steps:

- Verify the similarity among objects;
- Verify if the objects belong to the same layer;
- Verify if the objects are of the same type;
- Verify the relationships an object has with others objects and analyze if these relations are the same in both architectures.

To verify if the relationships are the same in both architectures we follow the next rule:

Knowing A and B correspond to an object of different architectures and R is the relationship between A and B, then:

If A1 has R1 with B1 and A2 has R2 with B2 and A1.ObjectType= A2. ObjectType and A1.LayerType= A2. LayerType and R1.type=R2.type and B1. ObjectType = B2. ObjectType and B1. LayerType = B2. LayerType and B1.name=B2.name and A1.name = A2.name then they have a similar relationship.

This rule is used to find relationships between similar objects with other similar objects. If this rule is respected then the discover patterns are classified as patterns out-of-the-box.

2) Patterns out-of-the-box

In this phase we classify the patterns out-of-the-box with rules. This classification is important to catalog the patterns because it becomes much more easier to organize them.

The denomination rules are as follow:

- <u>Denomination rules 1:</u> Is named complex pattern when one or more object are part of the generated pattern.
- <u>Denomination rules 2:</u> Is named simple pattern when only one object is part of the generated pattern.
- <u>Denomination rules 3:</u> The patterns are specific when all objects are similar and are a part of to pattern generated. All generated patterns are specific.

Taking into consideration the previous denomination rules we need to classify the generated patterns (see figure 2).

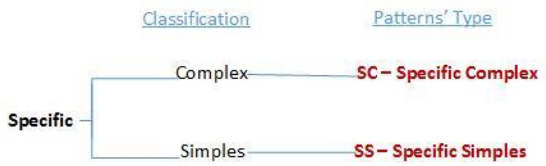

Fig. 2. *Pattern's Classification*

3) Comparison inside-the-box

In this phase are analyzed the internal aspects that compose patterns out-of-the-box, functionalities.

This phase needs the human intervention to indicate what functionalities among similar objects are actually similar. Here is used the gathering functionalities for all objects that make up the pattern used to analysis and detect similar functionalities.

a) Matching between object in patterns out-of-the-box

We can only analyze the functionalities among objects if these objects are previously classified similar. We need to guarantee that all functionalities are compared to the similar object. For each similar object the functionalities will be compared in order to the two objects have similar' functionalities. It's not to have 100% of similarity but we need to have at least one similar functionality between objects. If some objects have similar functionalities then we classified the patterns out-of-the-box as architectural pattern but we need to verify if there is any object in the patterns out-of-the-box that doesn't have similar functionalities. If so, the object is excluded from the pattern.

b) Patterns updating

Until this phase, we didn't have any architectural pattern defined, we only had similar objects (structure and functionalities) that can generate patterns. Now we have a similar pattern pair and we need to define an architectural pattern. To do it we need to generalize names for each object to compose the patterns. In the figure 3, we can verify the creation of the generalized pattern using information from two similar pair patterns belonging to different architectures.

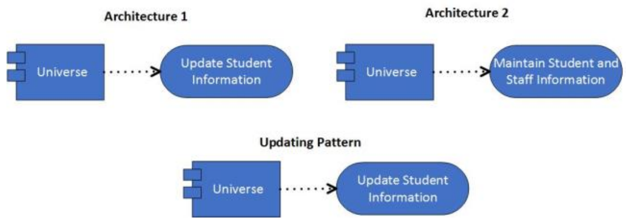

Fig. 3. Updating pattern

The main goal in the comparison inside-the-box is to indicate if a pattern out-of-the-box has similar objects when we compare their functionalities and we can also complete the specification of the architectural pattern with these functionalities.

c) Reference Architecture (RA)

After determine the architectural pattern we need to define if the pattern exists in the RA. We need to analyze the patterns discovered and in the RA verify if the architectural pattern exists. To do that, we need to compare the functionalities and the structure, like we did in the previous steps to find similar patterns. If the patterns exist in RA, we only need to verify if these patterns are in catalog, if not we need to add them in the catalog. If the patterns don't exist we need to add them in the catalog, this mean we continue towards the next steps.

Taking into consideration the changes in the life's cycle of the enterprise, the RA may become obsolete. In order to satisfy the new needs we must update the RA and this update has to be accept for the community inside the enterprise.

C. *Generation Patterns – 3rd Phase*

This phase can be more subjective because it depends on the perspectives that a person has about the system and the solutions for a problem. In this phase it is important to reflect about the discovered patterns and realize if the patterns provide us relevant or additional information.

We need to verify practical aspects about the usage of these patterns and to list the consequences to use them in the architecture. To list the consequences we can use the documentation or/and the people's experience.

After that, we need to classify the consequences of the patterns. The classifications consist in three levels: positive, neutral and negative. To do the classification we must use the information about the consequences previously collected.

Here we defined patterns and anti-patterns. If the patterns are considered positive or neutral then they are classified like pattern. If they are classified as negative we classify them as anti-patterns.

D. *Patterns Documentation – 4th Phase*

To document the patterns and the anti-patterns we need to transmit the knowledge collected in the experience. The patterns and anti-patterns catalog allow the usage of knowledgement to obtain positive and rapid results. The documentation needs to follow some well-define structure in order to not miss information. We can choose the structure but we suggest the following structure with the following fields: patterns classification, Patterns/Anti-Pattern, name, Problem, Context, Consequences, Describe Solution, Generic Architecture, Type of enterprise architecture and architectural principles.

III. IMPLEMENTING RCGD

In the Figure 4 it's possible to see the solution's functional architecture.

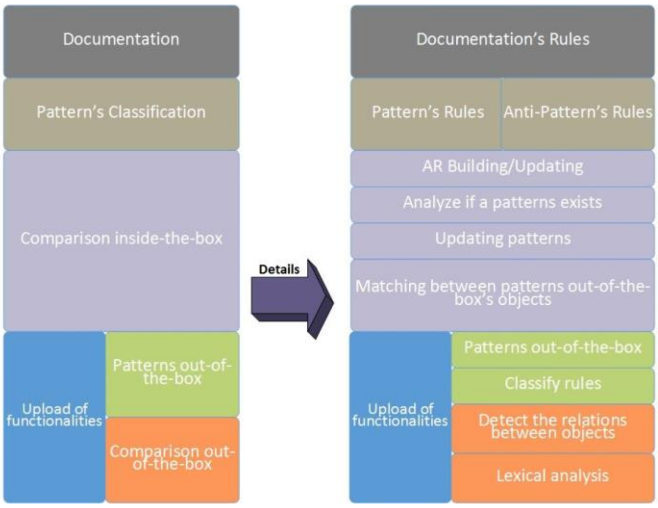

Fig. 4. RCGD Architecture

In the figure 4 it's possible to visualize the six modules composing the RCGD architecture:

- The module "upload functionalities" consists in performing the gathering of the objects' functionalities for the detection of patterns inside-the-box.
- The module "comparison out-of-the-box" does the lexical analysis of the architecture when some aspects of the architectures' objects are analyzed, like: layer, type of object, name, etc. We also find the relationships between objects in this module.
- The module "Patterns out-of-the-box" analyze the candidate patterns and we classify them with the denomination rules defined previously in the sub-phase "Patterns out-of-the-box". We need to classify them as "specific complex" or "specific simple".
- The module "comparison inside-the-box" consists in a module where the analyses of the functionalities in the objects are performed. In this module the construction/updating of the reference architecture is made from architectural patterns found.
- The module "pattern classification" analyze the consequences of the object belonging to an architectural pattern and classifies it as a pattern or anti-pattern.
- The module "documentation" consists in documenting the patterns found with the pre-defined structure.

A. *Analysis Architectures – 1st Phase*

Two architectures have been selected to analyze their problems and context. This analysis is made to verify if similarity exists between both architectures. The entity used in this laboratory test is the university. The main goal of this two architectures is to update the system of the university using the smartcard with focus in recording all registering of students and staff. The architectures used belong to the same university but with different line of businesses with the same problem and the same context.

The problem of the two architectures is about the access control system in the university context. When the context is in the university and the actor try to access one service. The architecture 1 is a sport center and the architecture 2 is the library. For this reason this two architectures were chosen for the analysis, because they have a similar problem and a similar context. The architecture used can be found in reference [4].

B. *Comparison Architectures – 2nd Phase*

1) *Comparison out-of-the-box*

We developed a program in C# to implement the RCGD method. To upload the architecture we used a XML file with an architecture's design. The XML file has only one architecture's design but before designing the architecture we need to define the meta-model. All of information is stored in a specific XML file.

a) Lexical Analysis

In the architecture's scanning we need to save the following information: object id, layer type, object type, object name, the relationship type an object has with other objects and the respective name of the object's relations. This information was save in a XML file with the structure presented in figure 5.

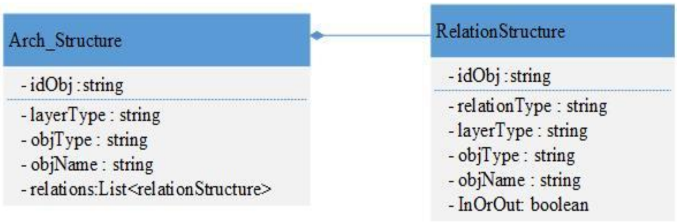

Fig. 5. Structure XML file

b) Discovering relation between objects

In this sub-phase we use the edit distance algorithm [5] to calculate the similarity between two objects. We will follow the workflow shown in figure 1 to calculate the similarity but we personalize this workflow using the edit distance algorithm [5] when the two words don't are synonymous, antonymous, preposition or conjunctions.

In the figure 6 we can see the comparison between two objects to detect the similarity. In the figure 6 we have two objects, object 1 is "Maintain Staff Student Information" and object 2 is "Update staff and student information". In this example we can verify all the combinations to detect the similarity.

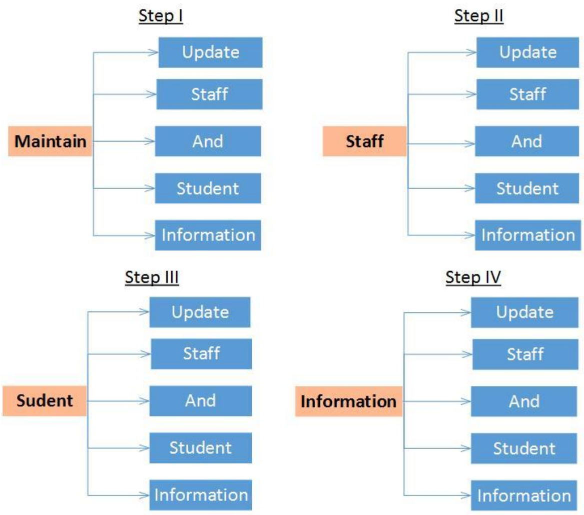

Fig. 6. Combination *between two object to compare the similarity*

b.1) Verification Realized before calculate the similarity

Before we initiate the comparison we created three files. One file with a list of synonymous, other file with a list of antonymous and the last one with a list of prepositions and conjunctions. It's from these files that we can verify if some words are a preposition or a conjunction. If that's the case for somes of the words we proceed like specify previously, we need to choose other words' combinations. If a few words are synonymous or antonymous then we don't calculate the similarity because the value of similarity is explicit in each case. We can't forget to update this list because we need to improve it.

b.2) Calculation of the similarities

For any combination of words we need to calculate the similarity. To calculate the similarity we compare all combinations of objects' names between the two architecture in analysis. And don't forget, a name can be composed by many words and any word can be a proposition, conjunction, synonymous, antonymous or neither of these. Now we will exemplify the process to calculate the similarities when none of them is verified.

To calculate the similarity between two words we used the edit distance algorithm [5] and after this calculation we divided the result for the biggest word. This is the definition of the formula 1. The biggest word means, the word has the biggest number of characters.

Formula 1: $editDistance(st1,st2)/bigCountChar()$.

In the table I we can see the application of the formula 1. It is relevant to refer that if the value of the formula is less or equal than 0.5, the words are similar. This formula is only used to know if two words are similar, so we don't use it to analyze if two objects are similar. To verify if two objects are similar we use formula 2.

TABLE I. CALCULATE THE SIMILARITY OF THE WORDS

Word1 (St1)	**Word2 (St2)**	**EditDistance (st1,st2) Values**	$\frac{EditDistance(s1,s2)}{bigCountChar()}$
House	Home	2	2/5 = 0.4
Universe	Student	7	7/8 = 0.88
Information	Information	5	5/11 = 0.45
Server	Servise	3	3/7=0.43
Aleph	Server	6	6/6= 1
Aleph	Aleph	0	0
Server	Server	0	0

After calculating the similarity between words we need to calculate the similarity between objects and the formula used is:

Formula 2: $CountSimilarWords(\leq 0.5)/bigWordName()$.

$CountSimilarWords(\leq 0.5)$ is a function that counts all similar words between objects and $bigWordName()$ is a function that counts all words separately in the two object and choose the number of word in the object with the largest number. In the table 8 we can see a table with an example where we calculate the final value of the similarity using the formula 2 [6].

TABLE II. CALCULATE THE SIMILARITY OF THE OBJECTS

Object 1	**Object 2**	**CountSimilarWords (≤0.5)**	$\frac{CountSimilarWords(\leq 0.5)}{bigwordName()}$
House Student	Information Student	1	½ = 0.5
Student Server	Universe	0	0
Information Server	Information Server	2	2/2 = 100%
Server	service	1	1/1 = 100%

An object is similar if the percentage of similarity is biggest or equal than 50% threshold. In the table II if we look at the red line, the percentage of similarity is 100% but if we analyze the objects we can conclude that they aren’t similar. If we don’t want this to happen on a next time we need to update the antonymous list. This way we can construct an antonymous’ collection to the specific industries and with that information, we improve the similarity detection process. We use the antonymous list to classify words that are not synonymous. While the patterns’ discovery is in progress, the files with synonymous, antonymous, preposition and conjunctions are updated.

In order to calculate the similarity we obtain a list of similar object and we need to exclude the objects that we think that are not similar.

b.3) Discover the relation between objects

In this sub-phase we only analyze the objects that have not been excluded and for these objects we need to analyze if the object has a relation with others similar objects. To detect this type of relationship we use the XML file generated when we did the scanning for the architecture and the XML file with the architecture’s design.

For example, let’s consider the following similar object in the table III and let’s look if there is a relation between similar objects using the previous rule.

TABLE III. SIMILAR OBJECTS

Objects’ Architecture 1	Objects’ Architecture 2
Universe	Universe
Student Information	Staff Student Information
Maintain staff Student Information	Update staff and Student Information

Previously we specify the following rule to detect relationships:

If A1 has R1 with B1 and A2 has R2 with B2 e A1.ObjectType= A2.ObjectType e A1.LayerType= A2.LayerType e R1.type=R2.type e B1.ObjectType = B2.ObjectType e B1.LayerType = B2.LayerType e B1.name=B2.name e A1.name = A2.name then they have a similar relationship.

With two XML files and this rule we can detect the following relationship:

- <u>Detect relationship 1</u>

If A1 (Universe) has R1 (Accesses) with B1 (Student Information) and A2 (Universe) has R2 (Accesses) with B2 (Staff Student Information) and A1.ObjectType (Application Component) = A2.ObjectType(Application Component) and A1.layerType (Application Architecture) = A2.layerType (Application Architecture) and R1.relationType (Accesses) = R2.relationType (Accesses) and B1.ObjectType (Data Object) = B2.ObjectType(Data Object) and B1.layerType (Business Architecture) = B2.layerType (Business Architecture) and B1.name (Student Information)= B2.name (Staff Student Information) and A1.name(Universe) = A2.name (Universe) then they have a similar relationship.

- <u>Detect relationship 2</u>

If A1 (Universe) has R1 (Realizes) with B1 (Maintain staff student information) and A2 (Universe)has R2 (Realizes) with B2 (update staff and student information) and A1.ObjectType (Application Component) = A2.ObjectType (Application Component) and A1.LayerType (Application Architecture) = A2.LayerType (Application Architecture)e R1.tipo(Realizes) = R2.tipo (Realizes) and B1.ObjectType (Application Service) = B2.ObjectType (Application Service) and B1.LayerType (Infrastructure Architecture)= B2.LayerType (Infrastructure Architecture) and B1.name (Maintain staff student information)=B2.name (update staff and student information) and A1.name (Universe) = A2.name (Universe) then they have a similar relationship.

The figure 7 is the results obtained using the previous rule and detect relationships. In this complex pattern we see more than one similar object.

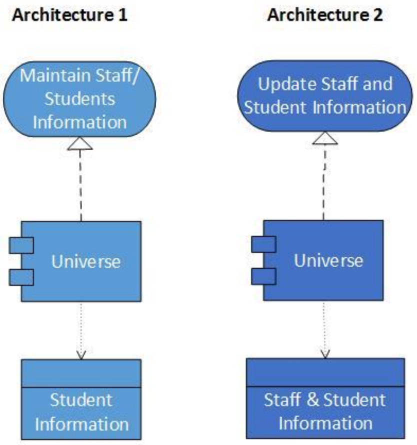

Fig. 7. Candidate patterns

The two relations detected have similar objects and they are also similar between them.

b.4) Candidate Pattern

Like we mention previously, after we get the list with the candidate patterns, we need to choose which patterns will be used to proceed with the pattern’s detection.

The person responsible for making the decision has to rely on his own analysis. In figure 7 we can see one of the candidate patterns choosen to precede with the process.

2) Patterns out-of-the-box

In this phase we analyze the candidate pattern and classify it like a pattern out-of-the-box based only in out-of-the-box aspects. We classify the candidate pattern of the figure 7 like patterns out-of-the-box because when we look at the objects from different architectures they appear to have similar functionalities, we rename these patterns as pattern out-of-the-box 1.

After we choose the patterns out-of-the-box we need to classify these patterns using the denomination rules.

We classify the pattern out-of-the-box 1 like “specific complex” because this pattern is composed by more than one object with a specific denomination.

For these six objects (three for each architecture) we gather the functionalities to be used in the comparison inside-the-box.

3) Comparison inside-the-box

To continue with this type of comparison we need to have the list of functionalities for each object that is a part if the out-of-the-box patterns.

a) Matching between object in patterns out-of-the-box

Using the pattern out-of-the-box 1 and using the functionalities gathered for the objects that compose this pattern we need to compare the functionalities between similar objects in different architectures.

Let’s now evaluate the similarity by doing a little resume about similar objects in the different architect:

Pattern out-of-the-box 1:

- Universe (Architecture 1) vs Universe (Architecture 2): both objects rely on a recorder system that saves information about the students and staff. This system gives information about the actors for other systems which can use this information.
- Student Information (Architecture 1) vs Staff Student Information (Architecture 2): is a database with actors’ information.
- Maintain Staff Student Information (Architecture 1) vs Update staff and student Information (Architecture 2): updating information about the staff and students.

When we find the similarities between functionalities we classify this pattern out-of-the-box 1 like architectural pattern 1 and after this classification we need to update the pattern.

b) Patterns updating

We need to generalize the architectural pattern 1 to create a unique architectural pattern without having two architectures. The goal of this step is to create an architectural pattern to represent the two similar fragments from different architecture, to achieve it we need to generalize the name and the functionalities.

b.1) Architectural pattern 1

In the figure 8 is possible to verify that the pattern is composed by more than one object. The names assigned to each object are generalized from the original objects that identify the objects in the architectural pattern. The data object named “student Information” comes from two object named “student information” and “staff student information”. The application service named “update student information” is created from two objects named “maintain staff student information” and “update student information”. The application component is named “Universe” because the two object had the same name “Universe”.

Fig. 8. Architectural pattern 1

- Architectural patterns’ functionalities

Object – Student Information: is a database with records of the student.

Object – Universe: Students and staff recorder system that give us information about its actors.

Object – Update student information: saving and updating personal students’ information and also information about his activities.

C. Patterns Documentation – 4^{th} Phase

Here we document the architectural patterns founded. But before that we need to specify the structure where we will save information about the pattern. To create the structure we rely in Martin Fowler‘s structure patterns [1] and Eric Gamma, Richard Helm, Ralph Johnson e John Vlissides ‘s structure pattern [7].

Architectural pattern 1

Pattern’s Classification: SC – Specific Complex.

Pattern/Anti-Pattern: Pattern.

Pattern’s name: Information Management.

Problem: we need to update the system and save the records of the actors’ information. These actors have actions in the system.

Context: This pattern is used for: save information and documents, data updating and system updating.

Consequences: keep record of the actors in the system and updating new data and new changes.

Solution: The pattern’s objects are:

- Object – Student Information: is a database with records of the student.
- Object – Universe: Students and staff recorder system that give us information about its actors.
- Object – Update student information: save and updating personal students’ information and also information about his activities.
- “Universe” (component) is realized by “Update student information”. “Universe” create a new object, read data from object, write and change data from “Student Information” data object and their relation are access notation.

Generic Architecture:

Fig. 9. Generic Architecture

Architectural principle: none.

CONCLUSION

In this research we explore the patterns and anti-patterns' importance in EA. Firstly we propose the RCGD method to detect partners and anti-partners The RCGD method does the revision, comparison, generation and documentation of patterns. In the revision the architectures are choosen, in the comparison we compare two architectures to detect the similar objects that compose the architectures, in the generation phase the patterns are generated and classified like patterns out-of-the-box and in the documentation phase the patterns are documented with a specific structure defined.

During this work we came across some problems, like detecting the border to define if two objects are similar, because we don't have to analyze the functionalities. If we analyzed all the functionalities between all object it would be a challenge, and also time consuming and it would be impossible to achieve this in a short time.

To calculate the similarity we divide the steps to find similar objects in two groups: out-of-the-box and inside-the-box. In the out-of-the-box we analyze external things like layer type, name, object type, etc. In the inside–the-box we analyze the functionalities. When we initialize the comparison inside-the-box we already find the similar objects in the steps out-the-box, so we don't need to do that many functionalities' comparisons. We only compare the similar out-of-the-box's objects.

Gathering functionalities was very difficult because of the poor documentation. In the 3rd phase we propose to evaluate if the architectural pattern is a pattern or anti-pattern analyzing the pattern's consequences. However the documentation provides us the information needed about the consequences about implementing the architecture component.

We did tests with people to detect patterns. We give them the same analysis' architecture and they needed to find a pattern without a method. They feel lost after a while and they were only able to match straightforward patterns. Many patterns found by them are not related with the functionalities but in simple suppositions. The steps in the comparison out-of-the-box are important steps because they eliminate many possible combinations between objects. We started by explaining the problems addressed in this investigation. This method can help us finding patterns in many EA. With it we can obtain patterns covering all EA and create a catalog with patterns and anti-patterns. The method for patterns' discovery gave us a process to follow. This process serve as a guide, which is essential because it's very easy to get lost taking into account the amount of data present. Thus, one can focus in essentials aspects. For future work we need to ensure that the patterns and anti-patterns are aligned with the enterprise's business and enterprise's principles. Another point for future work would be to automate the analysis of the functionalities, for example using natural language processing. All of these points to improve the solution are aligned with entreprise's goals and the solution's automation.

ACKNOWLEDGMENT

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

REFERENCES

[1] M. Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002

[2] Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal. Pattern-Oriented Software Architecture,volume Volume 1. WESLEY, 2001.

[3] N. Harrison, P. Avgeriou, and U. Zdun. Using patterns to capture architectural decisions. Software,IEEE, 24(4):38–45, 2007. ISSN 0740-7459. doi: 10.1109/MS.2007.124.

[4] N. Czechowski, S. Padam, I. Anderson, and C. Woodcock. Enterprise architecture evaluation report. Technical report, Coventry University.

[5] E. Ristad and P. Yianilos. Learning string-edit distance. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 20(5):522–532, 1998. ISSN 0162-8828.

[6] F. Gondima. Algoritmo de comparação de strings para integração de esquemas de dados. Master's thesis, Universidade Federal de Pernambuco.

[7] E. Gamma, R. Helm, R. Johnsin, and J. Vlissides. Design Patterns: Elements of Reusable ObjectOriented Software. ADDISON-WESLEY, 1995