Evolution Style Mining in Software Architecture

Kadidiatou Djibo[1,2], Mourad Oussalah[1] and Jacqueline Konate[2]
[1]*LS2N - UMR CNRS 6004, Nantes University, 110 Bd Michelet, Nantes, France*
[2]*Faculty of Sciences and Techniques, USTTB University, Bamako, Mali*

Keywords: Software Architecture, Evolution Style, Mining, Pattern, Sequence, Process, Data Mining.

Abstract: Sequential pattern extraction techniques are applied to the evolution styles of an evolving software architecture in order to plan and predict future evolution paths for the architecture. We present in this paper, a formalism to express the evolution styles in a more practical way. Then, we analyze these collected styles from the formalism introduced by the techniques of sequential patterns extraction to discover the sequential patterns of software architecture evolution. Finaly, from the analysis results, we develop a learning base and prediction rules to predict future evolution paths.

1 INTRODUCTION

Software systems become more complex day after day, and integrate many components. Thus, some research has focused on planning and predicting software evolution (Bhattacharya et al., 2012), (Goulão et al., 2012). However, software architectures go hand in hand with the software products they document, they evolve together and constantly. Although a lot of works have been directed towards the problem of reusing the evolution of software architectures ((Cuesta et al., 2013), (Ahmad et al., 2012)), little work has focused on the problem of planning and predicting the future evolution of software architectures. The majority of research efforts focused on the specification, development, deployment of software architectures (Smeda et al., 2005) and the analysis, design and reuse of the software architectures evolution ((Sadou, 2007), (Hassan and Oussalah, 2018)). But little works, to our knowledge, are devoted to planning and predicting future evolution in software architectures.

From previous evolution data of an evolving architecture over time A_1 to A_n, the problem is to determine the recurrent evolution sequences, the architectural elements most or least affected in order to identify and propose the possibilities and skills required to move towards A_{n+1}. To achieve this goal, the evolution style approach introduced in order to make the software architectures evolution process reusable is reused. the aim of this paper, is to extract software architectures evolution sequential patterns, to plan and predict future evolution paths. Thus, following previous software architectures evolutions, libraries of evolution styles are built, from which software architectures evolution sequential patterns are extracted. A meta-model of evolution style is proposed, it will be endowed with a simple formalism to express the evolution styles with more convenience. Data mining techniques are applied to the evolution styles expressed to extract the sequential patterns, then a learning base is developed to predict the future architecture evolution paths and evaluate the proposed paths. Indeed, the principles of sequential patterns extraction is used on software architectures evolution styles expressed through the defined formalism in order to determine the recurrent evolutions, the most or least affected architectural elements and the actors participation rate in operations during these architectural evolutions. By analogy with the approach of Agrawal et al. in (Agrawal and Srikant, 1995), data (the evolution styles expressed) is reorganized into a seven-field table where, is associated to an architectural element the date of evolution operation undergone, the name, the evolution style header. From this table, an algorithm to define evolution sequences by architectural element is defined, then the sequential patterns that define the recurrent evolution sequences are determined. Another algorithm was defined, based on a database of evolution sequences, it determines the architectural elements evolution rate and the actors participation rate in evolution operations.

The sequential patterns extraction is a very important area of data mining. It has been used in sev-

Djibo, K., Oussalah, M. and Konate, J.
Evolution Style Mining in Software Architecture.
DOI: 10.5220/0009349203130322
In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)*, pages 313-322
ISBN: 978-989-758-421-3

eral studies on specific types of data including the web (Wu and Chen, 2002), music (Hsu et al., 2001), software engineering (Amaral et al., 2014), ontology (Javed et al., 2011), medicine (Wright et al., 2015). However, the main challenge in extracting sequential patterns is related to the high costs of processing due to the large amount of data. Thus different algorithms have been proposed in previous studies to optimize data processing costs to determine sequential patterns (Agrawal and Srikant, 1995), (Srikant and Agrawal, 1996) (Mooney and Roddick, 2013).

This paper is organized as follows: Section 2 presents the state of the art of software architecture evolution styles and knowledge extraction from data (data mining). In Section 3, we present the evolution style meta-model and the new formalism that we introduced to express evolution styles with more convenience. Section 4 presents the data analysis, we apply the mining sequential patterns techniques to the data expressed according to the formalism introduced through a case study. Finally in section 5, we conclude and give the perspectives of our work.

2 STATE OF THE ART

Some existing work related to software architecture evolution styles and data mining are presented below.

2.1 Software Architecture Evolution Style

An evolution style captures a characteristic way of evolving all or part of a software architecture. It serves as a guide for an architect who must conform to the style (Le Goaer, 2009). Evolution styles aim to make the evolution activity reusable to prevent architects from starting from scratch with each evolution activity. They promote knowledge sharing but also learning and knowledge extraction. We present some team approaches. According to (Garlan, 2008), an evolution style expresses the evolution of software architecture as a set of potential evolution paths from the initial architecture to the target architecture. Each path defines a sequence of evolution transitions, each of which is specified by evolution operators. The team (Cuesta et al., 2013) has defined evolution styles based on architectural knowledge (AKdES), which are also based on architecture design decisions each time an evolution step is made. Each stage of evolution is preformed because a decision of evolution is taken following the verification of a decision of evolution. According to (Oussalah et al., 2008), the main idea of an evolution style is to model software architecture evolution activity in order to provide reusable expertise of domain-specific evolution. They consider an architectural evolution as consisting of modifications (addition, update, deletion) of architectural elements (component, connector, interface).

2.2 Extracting Knowledge from Data (Data Mining)

Extracting knowledge from data (data mining) has been used in many areas to find patterns to solve decision-making or future projection problems in companies. We then present some work done in this direction. Agrawal et al. (Agrawal and Srikant, 1995), introduced the sequential pattern discovery problem. From a database of client transactions, they define a sequence database where each sequence represents all the items purchased during a transaction. It was a question of discovering all the sequential patterns with a specified minimum support. They defined three algorithms to solve the problem including the AprioriAll, AprioriSome and DynamicSome algorithms. In 1996, the same team proposed an improvement of the previous result, they present the GSP algorithm for the discovery of generalized sequential patterns. A performance evaluation performed in (Srikant and Agrawal, 1996) indicates that GSP is performing better than AprioriAll presented in (Agrawal and Srikant, 1995). (Javed et al., 2011) study the change of ontology. They analyze ontology change logs represented as graphs to determine frequent and recurring changes. These frequent and recurring changes are identified as patterns of change that can be reused. For this, they introduced two algorithms to determine ontology change patterns, which are the algorithm for searching complete and ordered change patterns (OCP) and the search algorithm for complete and unordered change patterns (PCU). They then performed a performance study of the two algorithms to determine the different limitations.

The new model that we introduce to express and analyze software architectures evolution styles in order to predict and plan future evolutions of theses is presented below.

3 META-MODEL AND SIMPLIFIED EXPRESSION OF EVOLUTION STYLE

Our goal is to apply mining sequential patterns techniques to software architectures evolution styles in order to discover sequential patterns of architecture evolution, determine the evolution rate of architectural elements and the actors participation rate in the evolution operations in order to predict and plan the future evolution of architectures. For this, we must first have a database of evolution styles. Previously, we have defined a meta-model of evolution style. We extend it to define an evolution style as a process (Fig. 1) by specifying the role, the architectural element and the operation. Thus, the extended meta-model (Fig. 1) answers the following questions : what ? (what is evolving ?) through the ArchitectureElement package, who ? (who did it ?) through the Actor concept, when ? (when to evolve ?) from the TimeEvolution concept and how ? (how to make it evolve ?) through the concepts Header, Competence, Action ands Impact. All the concepts Header, Competence, Action, Impact define the operation. Through the class diagram in Figure 1, we highlight the concepts and relations of our model.

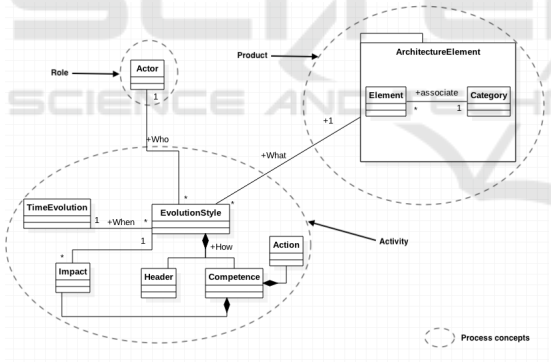

Figure 1: Evolution Style Meta-model.

The evolution style meta-model presented in Figure 1 is described below.

3.1 Description of Evolution Style Meta-model Concepts

The concept **EvolutionStyle** is the core of our model, it encapsulates what allows to describe and apply an operation of evolution to an architectural element. It consists of two complementary parts: a header and a competence. The **Header** class describes the signature of the operation. The **competence** class is split into Action and Impact. **Action** is a procedure or function of evolution that focuses on the evolving architectural element. It describes an implementation unit corresponding to the header. The **Impact** class specifies the evolution styles that will be impacted by the execution of the currently defined style. It allows to establish a relationship between the evolution styles. The **Actor** concept defines the actor, either a natural person or a program that triggers the operation. The **ArchitectureElement** package including the evolving element and its category allows to model and reify any significant element of an evolving architecture. If an architectural concept is an instance of this class (in the object-oriented sense), then it becomes possible to associate evolution styles with it. In addition, the **Category** concept allow to share architectural elements of the same nature in a class. The **TimeEvolution** class indicates the date on which the evolution operation is performed on the evolving element.

It is the role (defined through the Actor concept), the architectural element and the operation (defined through the Header, Competence, Action and Impact concepts) that allow our meta-model to define an evolution style as a process (Fig. 1).

In the following, we introduce the formalism to express the evolution styles according to the extended meta-model (Fig. 1) in order to easily collect them.

3.2 Simplified Expression of Evolution Style

While waiting to collect evolution data from software architectures in order to extract sequential patterns, we propose a formalism allowing evolution actors to easily express software architectures evolution styles. We use the introduced meta-model (Fig. 1) and define a formalism that specifies the actor, the architectural element, the operation and the date of evolution. The header defined through the Header concept, identifies the signature of the evolution operation. It is unique, so we replace the operation by the concept Header and the architectural element by Element and Category. Figure 2 below represents the formalism. This

Style-name : < Actor, (Element, Category), Header, TimeEvolution >

Figure 2: Simplified Expression of Evolution Style.

final expression (Fig. 2) will allow to name and express one by one all the evolution styles of an evolving architecture A_1 to A_n. After expressing all the evolution styles of the evolving architecture with the simplified expression, a large amount of Evolution data is obtained, it can be use to extract the sequential evo-

lution patterns of software architectures, discover the architectural elements change rate and the actors participation rate in evolution operations in order to plan and predict all possible paths towards the An+1 architecture.

We present below our approach to extract the sequential patterns of architectural evolution from the data of evolution styles expressed by the formalism introduced above.

4 SEQUENTIAL PATTERNS EXTRACTION OF SOFTWARE ARCHITECTURES EVOLUTION

In artificial intelligence, there are 2 main extraction techniques, algorithmic and deep learning. We chose the algorithmic. Referring to the definitions in (Agrawal and Srikant, 1995), we define some concepts that we use for the sequential patterns extraction of software architectures evolution.

Evolution Sequence: We call an evolution sequence an ordered sequence of evolution style headers applied to a given architectural element or performed by a given actor. The order is established according to the dates of evolution.

Support: The support of an evolution sequence is the percentage of appearance of this sequence in the other evolution sequences.

Sequential Pattern: We define sequential pattern of software architectures evolution an ordered sequence of evolution operations carried out in the same order on a defined number of architectural elements. This number defined by the user represents the minimum support of an evolution sequence to be admitted as a sequential pattern.

The Sequence Length: is the total number of evolution style headers contained in the sequence.

An architectural evolution consists of creation (C), suppression (S) and modification (M) of architectural elements. An architecture can be created (C), suppressed (S), modified (M) or migrated (Mg). The latter results in the creation of a new architecture (advanced version of the previous one). An architecture change (M) consists of adding components (A_c), adding connectors (A_{con}), removing components (S_c), removing connectors (S_{con}) and changing architectural elements. A modification of architectural elements consists of adding ports (C_{pc} and C_{pcon}), deleting ports (S_{Pc} and S_{Pcon}) and changing ports (M_{PC}) for components and (M_{PCon}) for connectors. An architectural evolution is an ordered set of evolution operations carried out on the architectural elements (modification of architectural elements) in order to reach a targeted result. An evolution style is a process that describes an evolution operation carried out on a given architectural element during an architectural evolution. Thus, an architectural evolution can be represented by an ordered set of evolution styles. Based on this principle, we use the formalism introduced to extract the sequential evolution patterns of architectures in order to define evolution sequences by architectural element in a category. To do this, we reorganize the styles expressed by defining a table in which we define in column each element of the formalism.

To better explain our approach we refer, pedagogically to the following case study of the evolution of an initial architecture (defined using components and connectors) A_1 to an architecture A_3 where we apply sequential pattern extraction techniques to predict the possibles A_4 (A_{4_1}, A_{4_2}, ..., A_{4_n}).

4.1 Case Study

In this case study, we collect existing data on the evolution of the initial architecture A_1 to A_3, we express them using the meta-model and the simplified expression and then we analyze them to predict the architecture A_4. Thus, this study is carried out in three phases, including the expression phase of evolution styles, the analysis phase of expressed styles and finally the prediction phase.

4.1.1 The Expression Phase of Evolution Styles

We use the simplified evolution style expression introduced to express all evolving architecture evolution styles from the creation of the initial architectural elements until the last modification operation carried on the last version of this one in order to predict and plan future developments. We refer to the following Figure 4 of evolution of the architecture A_1 to A_3.

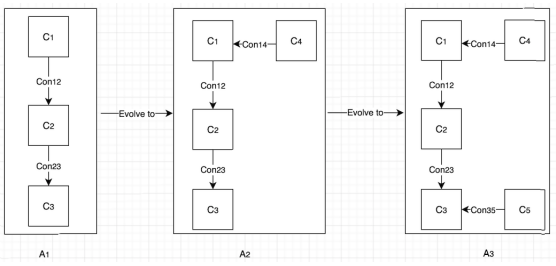

Figure 3: Evolution of the Architecture A_1 to A_3.

The architecture A_1 at the beginning has three components C_1, C_2 and C_3. Components are connected

by connectors C_{on12} and C_{on23}. The software architect decides to migrate A_1 to A_2 by creating the component C_4 and the connector C_{on14}. In this case study we start with two categories of architectural elements (connector and component). We express the evolution styles for the creation of the initial components and connectors of the initial architecture A_1 and those for the migration of A_1 to A_3. For reasons of paper size, we just give some expressions, the whole is given in table 1.
e_1 : < Act1, (C_1, Component), A_c, 01-05-2017 >. Creation of the component C_1.
e_2 : < Act1, (C_2, Component), A_c, 02-05-2017 >. Creation of the component C_2.
e_3 : < Act1, (C_{on12}, Connector), A_{con}, 03-05-2017 >. Creation of the connector C_{on12}.
e_4 : < Act2, (C_1, Component), C_{pc}, 03-05-2017 >. Creating a port on the component C_1.
e_5 : < Act2, (C_{on12}, Connector), C_{pcon}, 03-05-2017 >. Creating a port on the connector C_{on12}.
e_6 : < Act3, (C_1, Component), M_{pc}, 03-05-2017 >. Modification of the port on the component C_1 to connect C_{on12}.
We get A_3 after the evolution operations e_1 to e_{41}. We analyze below these expressed evolution styles, in order to determine the recurrent style sequences.

4.1.2 Analysis Phase of Expressed Styles

Inspired by (Agrawal and Srikant, 1995), we apply the techniques of sequential patterns extraction to the expressed evolution styles in order to determine the recurrent evolution sequences by category of architectural elements, the rate of change of architectural elements and the actors participation rate in the evolution operations. First, we reorganize the data expressed in a table (Table 1).

Indeed, we are interested in the evolution operations carried out on the architectural elements of the same category during an architectural evolution and the actors associated to these operations. The evolution date allows us to define the operations in sequence by architectural element. After reorganizing the data, we define a second table (Table 2) in which, from the first table defined (Table 1), in a given category we associate with each architectural element the evolution sequence corresponding as in Table 2. The empty sequence () is associated with the element that has not undergone any evolution operation.

An interpretation of a line from Table 2 would be: The architectural element C_1 after its creation has undergone four evolution operations of header creating component port C_{pc}, component port modification M_{pc}, creating component port C_{pc} and component port modification M_{pc} respectively. From Table 2 we can determine the architectural elements most or least affected by the length of their evolution sequence. The length of the evolution sequence associated with C_3 is four, while the length of the sequence associated with C_5 is two, we conclude that among the components C_3, C_2, C_1 have undergone more evolution operations. Thus, rules can be developed to make decisions or draw conclusions. To do this, we define an algorithm to compute and associate to each architectural element its rate of evolution. The architectural element evolution rate is computed from the length of its evolution sequence and the total number of lines in Table 1, i.e. the total number of evolution styles collected. In the same way in the Table 3 we associate with each architecture its sequence of evolution, we note that the architecture A_1 has undergone after its creation (C) ten modifications including a component addition A_c, connector addition A_{con}, etc. before migrating (Mg). From the tables Table 2 and Table 3, we determine the support of each sequence by category in the following Table 4 in order to discover the sequential patterns.

We retain as a sequential pattern all evolution sequences with a support value greater than twenty-five percent (25 %). This value is arbitrary and can be defined by the user. Table 5 represents the sequential patterns by category of architectural elements.

More than twenty-five percent (25 %) of components have undergone evolution sequences (C_{pc} M_{pc} C_{pc} M_{pc}) and (C_{pc} M_{pc}). More than twenty-five percent (25 %) of connectors have undergone the sequence (C_{pcon} M_{pcon} C_{pcon} M_{pcon}). More than twenty-five percent (25 %) of architectures have undergone the sequence (A_c A_{con} C_{pc} C_{pcon} M_{pc} M_{pcon} C_{pc} C_{pcon} M_{pc} M_{pcon} Mg).

The sequential evolution patterns of software architectures correspond, in our case, to the sequences or subsequences of evolution appearing in the evolution sequences of a number of architectural elements greater than the minimum support specified by the user (k). Thus, to extract them from the Table 2, each sequence must be compared to all the other evolution sequences of the table. If a match is detected its support is incremented. At the end of the table's path its support is calculated. All the evolution sequences in Table 2 are candidate sequences, i.e. the associated support must be computed in order to extract the sequential evolution patterns of software architectures. However, during the table run if an evolution sub-sequence is read in another sequence, this sub-sequence will also be added to the candidate sequences. Its support will also be computed. Finally, all the sequences or subsequences having a calculated support greater than k are retained as sequential pat-

terns of software architectures evolution. Given the amount of evolution data and processing complexity, it is not easy to extract sequential patterns manually. Thus, we propose an algorithm allowing to extract the sequential patterns from any organized table like the Table 2 whatever the quantity of data. But the main challenge in defining sequential pattern extraction algorithms is the high cost of processing due to the high amount of data (Chiu et al., 2004). Many studies have been carried out in this context, proposing efficient and effective algorithms for the sequential patterns extraction (Agrawal and Srikant, 1995), (Srikant and Agrawal, 1996) (Chiu et al., 2004) (Mahajan et al., 2014). Thus, our objective is centered on the sequential patterns extraction of software architectures evolution. Inspired by this work already done to optimize algorithms for extracting sequential patterns, we try to propose computer algorithms to extract our sequential patterns of software architectures evolution from our data formats defined (Ex Table 2), compute the evolution rate of an architectural element and the participation rate of an actor in evolution operations.

4.1.3 Prediction Phase

To propose the possibles A_{n+1} (the possibles A_4 for this case study) to the architect, we develop a learning base (Figure 3 and Figure 5 below). We load the Table 2, Table 3 and Table 5 tables resulting from the analysis carried out in the previous phase. Figure 5 below provides an overview of the learning base.

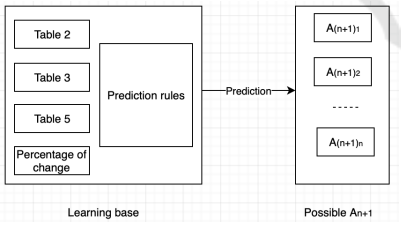

Figure 4: Learning and Prediction.

We define some rules for prediction. The rules are dynamic, they can be modified by the architect. We distinguish four categories of rules: the structural evolution rules, the behavioral evolution rules, the invariant evolution rules and the business evolution rules. Structural evolution rules refer to the evolution of the structure of architectural elements. Behavioural evolution rules refer to the way in which the behaviour of architectural elements has evolved. The invariant evolution rules guarantee properties on architecture and its architectural elements. Business evolution rules affect the costs (price, quality) and time (lifetime) of element evolution. We give some examples of rules by category.

Structural Evolution Rules

Rule 1
We define the notion of connectability of a component. A component is said to be connectable if it is possible to connect it to another component via a connector. Indeed, the components contain the property of number of ports (variable and definable by the architect), if the number of existing connections reaches the number of port of the component, it becomes not connectable. Thus its status can switch to connectable as soon as one of its ports is released following a deletion or a modification. The number of ports is specified in the component properties. An unconnectable component will not be affected during the prediction.

Behavioural Evolution Rules

Rule 1
The architect can define an architectural element that is not sensitive to evolution (Properties, structure and behaviour that make the element non-sensitive). In this case, it will not be affected by future evolution operations.
Rule 2
Architectural elements that have undergone an evolution rate greater than X % (X definable by the architect) are no longer sensitive to evolution.

Invariant Evolution Rules

Rule 1
The architecture must be for example a connected graph.

Business Evolution Rules

Rule 1
The expensive elements, whose evolution is expensive are less privileged.
Rule 2
Evolutions involving the architectural elements least affected by previous evolution operations are given priority to evolution.

For this case study, let's apply:
Structural evolution rules, all components have three ports defined, you can not go beyond three connections on a component.
Behavioural evolution rules, the components C_1, C_3 and C_4 are defined as not sensitive to change and X (minimum rate of change per architectural element) is set at eighty-five percent (85 %).

Thus three paths of evolution open:

- Creating a connector between components C_5 and C_2, this associates with architecture A_3 the sequence (A_{con} C_{pc} C_{pcon} M_{pc} M_{pcon} C_{pc} C_{pcon} M_{pc} M_{pcon} Mg) with the following architecture A_{4_1} (Figure 6);
- Creating a component C_6 and a connector $C_{on}62$, this associates with architecture A_3 the sequence (A_c A_{con} C_{pc} C_{pcon} M_{pc} M_{pcon} C_{pc} C_{pcon} M_{pc} M_{pcon} Mg) with the following architecture A_{4_2} (Figure 6);
- Creating a component C_6 and a connector $C_{on}65$, this associates with architecture A_3 the sequence (A_c A_{con} C_{pc} C_{pcon} M_{pc} M_{pcon} C_{pc} C_{pcon} M_{pc} M_{pcon} Mg) with the following architecture A_{4_3} (Figure 6);

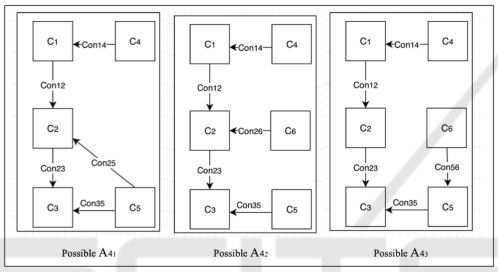

Figure 5: Possibilities.

4.1.4 Evaluation of Proposals

This evaluation is based on Table 5, the sequence associated with the architecture being migrated is compared to the sequential patterns associated with the architecture discovered (Table 5), if the sequence is identical to one of the sequential patterns discovered the weight one (1) is associated with the possibility otherwise the zero weight (0) as represented in Table 6.

If we apply the business evolution rules, the evolutions carried out on the components C_4 and C_5 will be preferred, so the proposal A_{4_1} and A_{4_2} will be excluded, A_{4_3} would be the only proposal. This (only one proposal) results from the case study chosen, depending on the case, more can be achieved. The evolution sequences by actor and the calculation of the actors participation rate in the evolution operation allow to plan the proposed future evolution operation. Indeed, for each path, we are able to propose the competences (actors) that can be involved. Through Table 7 we give an overview of the other results obtained from this case study in addition to the other tables defined.

In the following section, the principles adapted to respectively define the algorithms for extracting sequential patterns of software architecture evolution and calculating the architectural elements evolution rate and the actors participation rate in evolution operations are explained.

4.2 Principle to Extract Sequential Patterns of Software Architectures Evolution and Calculating the Architectural Elements Change Rate

(For reasons of paper size, we could not give an overview of the defined functions, but we explain the principles.)

We define a first function which, starting from the Table 1, associates with each architectural element, the corresponding evolution sequence. The function named *Sequence*, retrieves the table (Table 1) sorted on the Category, TimeEvolution and Element columns and associates with each architectural element the corresponding evolution sequence. It provides as an output an equivalent of Table 2. The second function named *SequenceSupport* allows to compute and associate to each candidate sequence its support, it takes as input the candidate sequences defined from Table 2 (output of the previous function), then computes and associates to each sequence its total number of appearance among all the other candidate sequences. It provides as an output a table that associates each candidate sequence with its total number of appearances. The *SequentialPattern* function returns sequential patterns by category of architectural elements with k support provided as a parameter. For example, if we take a k equal to twenty-five percent, the sequential patterns will correspond to all the evolution sequences or sub-sequences appearing in the evolution sequences of more than twenty-five percent of architectural elements in the same category. It takes as input the output of the previous function, computes and associates to each candidate sequence its support in percentage and compares it to the minimum support k provided in parameter. If a superiority is read, the current sequence is stored in the sequential pattern table. At the end of the process, it provides this sequential pattern table which contains all the sequences with a support higher than the k provided. The *PercentageEvolution* function takes as input any table similar to Table 2 associated with Table 1 that contains all the evolution operations performed. It associates to each architectural element, its rate of evolution by multiplying the size of the evolution se-

quence associated with the element by hundred then dividing the result by n (the size of Table 1 or the total number of evolution styles involved in the search for sequential patterns). As an output, the algorithm provides a two-column table where, to each architectural element is associated its rate of evolution or to each actor its rate of participation in evolution operations.

5 CONCLUSION

In this paper, we have presented a technique based on sequential pattern extraction to plan and predict future evolution paths of an evolving software architecture. In addition, we evaluated the proposed evolution paths and defined some algorithms to define sequences, determine sequential patterns and compute the architectural elements evolution rate and the actors participation rate in evolution operations.
In the next step, we propose to apply our approach to a evolving software system in production , then to an urban architecture evolutions in order to propose a model that is easily usable, prospective and predictive, allowing us to analyze, estimate and predict future evolutions of urban architecture.

REFERENCES

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *icde*, page 3. IEEE.

Ahmad, A., Jamshidi, P., Arshad, M., and Pahl, C. (2012). Graph-based implicit knowledge discovery from architecture change logs. In *Proceedings of the WICSA/ECSA 2012 Companion Volume*, pages 116–123. ACM.

Amaral, J. N., Jocksch, A. P., and Mitran, M. (2014). Mining sequential patterns in weighted directed graphs. US Patent 8,683,423.

Bhattacharya, P., Iliofotou, M., Neamtiu, I., and Faloutsos, M. (2012). Graph-based analysis and prediction for software evolution. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 419–429. IEEE.

Chiu, D.-Y., Wu, Y.-H., and Chen, A. L. (2004). An efficient algorithm for mining frequent sequences by a new strategy without support counting. In *Proceedings. 20th International Conference on Data Engineering*, pages 375–386. IEEE.

Cuesta, C. E., Navarro, E., Perry, D. E., and Roda, C. (2013). Evolution styles: using architectural knowledge as an evolution driver. *Journal of Software: Evolution and Process*, 25(9):957–980.

Garlan, D. (2008). Evolution styles-formal foundations and tool support for software architecture evolution. *Computer Science Department, reports-archive.adm.cs.cmu.edu*, page 650.

Goulão, M., Fonte, N., Wermelinger, M., and e Abreu, F. B. (2012). Software evolution prediction using seasonal time analysis: a comparative study. In *2012 16th European Conference on Software Maintenance and Reengineering*, pages 213–222. IEEE.

Hassan, A. and Oussalah, M. C. (2018). Evolution styles: Multi-view/multi-level model for software architecture evolution. *JSW*, 13(3):146–154.

Hsu, J.-L., Liu, C.-C., and Chen, A. L. (2001). Discovering nontrivial repeating patterns in music data. *IEEE Transactions on Multimedia*, 3(3):311–325.

Javed, M., Abgaz, Y. M., and Pahl, C. (2011). Graph-based discovery of ontology change patterns.

Le Goaer, O. (2009). *Styles d'évolution dans les architectures logicielles*. PhD thesis, Université de Nantes; Ecole Centrale de Nantes (ECN).

Mahajan, S., Pawar, P., and Reshamwala, A. (2014). Performance analysis of sequential pattern mining algorithms on large dense datasets. *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, 3(2).

Mooney, C. and Roddick, J. F. (2013). Sequential pattern mining - approaches and algorithms. *ACM Comput. Surv.*, 45:19:1–19:39.

Oussalah, M. C., Le Goaer, O., Tamzalit, D., and Seriai, A. (2008). Evolution shelf: Exploiting evolution styles within software architectures. In *SEKE*, pages 387–392.

Sadou, N. (2007). *Evolution Structurelle dans les Architectures Logicielles à base de Composants*. PhD thesis, Université de Nantes et école centrale de Nantes, archives-ouvertes.fr.

Smeda, A., Oussalah, M., and Khammaci, T. (2005). Madl: Meta architecture description language. In *Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA'05)*, pages 152–159. IEEE.

Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, pages 1–17. Springer.

Wright, A. P., Wright, A. T., McCoy, A. B., and Sittig, D. F. (2015). The use of sequential pattern mining to predict next prescribed medications. *J. of Biomedical Informatics*, 53(C):73–80.

Wu, Y.-H. and Chen, A. L. (2002). Prediction of web page accesses by proxy server log. *World Wide Web*, 5(1):67–88.

APPENDIX

Table 1: The Evolution Styles of Architecture A_1 to A_3 Reorganised.

Architecture evolution	Style Name	Actor	Element	Category	Header	TimeEvolution
Initial architecture creation	e_1	Act1	C_1	Component	A_c	01-05-2017
	e_2	Act1	C_2	Component	A_c	02-05-2017
	e_3	Act1	C_{on12}	Connector	A_{con}	03-05-2017
	e_4	Act2	C_1	Component	C_{pc}	03-05-2017
	e_5	Act2	C_{on12}	Connector	C_{pcon}	03-05-2017
	e_6	Act3	C_1	Component	M_{pc}	03-05-2017
	e_7	Act3	C_{on12}	Connector	M_{pcon}	03-05-2017
	e_8	Act2	C_2	Component	C_{pc}	03-05-2017
	e_9	Act2	C_{on12}	connector	C_{pcon}	03-05-2017
	e_{10}	Act3	C_2	Component	M_{pc}	03-05-2017
	e_{11}	Act3	C_{on12}	connector	M_{pcon}	03-05-2017
	e_{12}	Act1	C_3	Component	A_c	04-05-2017
	e_{13}	Act1	C_{on23}	connector	A_{con}	05-05-2017
	e_{14}	Act2	C_2	Component	C_{pc}	05-05-2017
	e_{15}	Act2	C_{on23}	Connector	C_{pcon}	05-05-2017
	e_{16}	Act3	C_2	Component	M_{pc}	05-05-2017
	e_{17}	Act3	C_{on23}	Connector	M_{pcon}	05-05-2017
	e_{18}	Act2	C_3	Component	C_{pc}	05-05-2017
	e_{19}	Act2	C_{on23}	Connector	C_{pcon}	05-05-2017
	e_{20}	Act3	C_3	Component	M_{pc}	05-05-2017
	e_{21}	Act3	C_{on23}	Connector	M_{pcon}	05-05-2017
A_1	e_{22}	Act1	C_4	Component	A_c	06-05-2018
	e_{23}	Act1	C_{on14}	Connector	A_{con}	07-05-2018
	e_{24}	Act2	C_4	Component	C_{pc}	07-05-2018
	e_{25}	Act2	C_{on14}	Connector	C_{pcon}	07-05-2018
	e_{26}	Act3	C_4	Component	M_{pc}	07-05-2018
	e_{27}	Act3	C_{on14}	Connector	M_{pcon}	07-05-2018
	e_{28}	Act2	C_1	Component	C_{pc}	07-05-2018
	e_{29}	Act2	C_{on14}	Connector	C_{pcon}	07-05-2018
	e_{30}	Act3	C_1	Component	M_{pc}	07-05-2018
	e_{31}	Act3	C_{on14}	Connector	M_{pcon}	07-05-2018
A_2	e_{32}	Act1	C_5	Component	A_c	10-05-2019
	e_{33}	Act1	C_{on35}	Connector	A_{con}	11-05-2019
	e_{34}	Act2	C_5	Component	C_{pc}	11-05-2019
	e_{35}	Act2	C_{on35}	Connector	C_{pcon}	11-05-2019
	e_{36}	Act3	C_5	Component	M_{pc}	11-05-2019
	e_{37}	Act3	C_{on35}	Connector	M_{pcon}	11-05-2019
	e_{38}	Act2	C_3	Component	C_{pc}	11-05-2019
	e_{39}	Act2	C_{on35}	Connector	C_{pcon}	11-05-2019
	e_{40}	Act3	C_3	Component	M_{pc}	11-05-2019
	e_{41}	Act3	C_{on35}	Connector	M_{pcon}	11-05-2019

Table 2: Evolution Sequence by Architectural Element and Category.

Category	Element	Evolution sequence
Component	C_1	$(C_{pc}\ M_{pc}\ C_{pc}\ M_{pc})$
	C_2	$(C_{pc}\ M_{pc}\ C_{pc}\ M_{pc})$
	C_3	$(C_{pc}\ M_{pc}\ C_{pc}\ M_{pc})$
	C_4	$(C_{pc}\ M_{pc})$
	C_5	$(C_{pc}\ M_{pc})$
Connector	C_{on12}	$(C_{pcon}\ M_{pcon}\ C_{pcon}\ M_{pcon})$
	C_{on23}	$(C_{pcon}\ M_{pcon}\ C_{pcon}\ M_{pcon})$
	C_{on14}	$(C_{pcon}\ M_{pcon}\ C_{pcon}\ M_{pcon})$
	C_{on35}	$(C_{pcon}\ M_{pcon}\ C_{pcon}\ M_{pcon})$

Table 3: Evolution Sequence by Architecture.

Architecture	Evolution sequence
A_1	$(A_c\ A_{con}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}$ Mg)
A_2	$(A_c\ A_{con}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}$ Mg)
A_3	()

Table 4: Support by Architectural Element and Category.

Category	Evolution sequence	support
Architecture	$(A_c\ A_{con}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}$ Mg)	66,6 %
Component	$(C_{pc}\ M_{pc}\ C_{pc}\ M_{pc})$	60 %
	$(C_{pc}\ M_{pc})$	100 %
Connector	$(C_{pcon}\ M_{pcon}\ C_{pcon}\ M_{pcon})$	100 %

Table 5: Sequential Pattern by Category of Architectural Elements.

Category	Sequential pattern >25 %
Architecture	$(A_c\ A_{con}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}$ Mg)
Component	$(C_{pc}\ M_{pc}\ C_{pc}\ M_{pc})$
	$(C_{pc}\ M_{pc})$
Connector	$(C_{pcon}\ M_{pcon}\ C_{pcon}\ M_{pcon})$

Table 6: Evaluation.

Possibilities	Evolution sequence	Weight
A_{4_1}	$(A_{con}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}$ Mg)	0
A_{4_2}	$(A_c\ A_{con}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}$ Mg)	1
A_{4_3}	$(A_c\ A_{con}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}\ C_{pc}\ C_{pcon}\ M_{pc}\ M_{pcon}$ Mg)	1

Table 7: Other Results.

Category	Result
Architectural elements most affected	C_3, C_2, C_1, C_{on12}, C_{on23}, C_{on14}, C_{on35}
Architectural elements less affected	C_4 and C_5
The selected architecture	A_{4_3}
The most active actors	Act2 and Act3
The least active actors	Act1