applied sciences

MDPI

Article

A Method for Enterprise Architecture Model Slicing

Hong Guo [1,*], Jingyue Li [2], Shang Gao [3] and Darja Smite [4]

1 School of Business, Anhui University, Hefei 230093, China
2 Department of Computer Science, Norwegian University of Science and Technology, 7034 Trondheim, Norway
3 School of Business, Örebro University, 701 82 Örebro, Sweden
4 Department of Software Engineering, Blekinge Institute of Technology, 371 41 Karlskrona, Sweden
* Correspondence: 15079@ahu.edu.cn

Abstract: Enterprise Architecture (EA) has been applied widely in industry as it brings substantial benefits to ease communication and improve business-IT alignment. However, due to its high complexity and cost, EA still plays a limited role in many organizations. Existing research recommends realizing more of the EA potential. EA can be developed for specific purposes, accumulated in a digital repository, and reused when needed later. Due to the diversity and inconsistency of the repository, it is challenging to find relevant EA data and reuse it. In the present research, we propose using slicing techniques to extract EA models for reuse. We validate the method with an official EA repository hosted by The Open Group. The result shows that the method could facilitate extracting existing EA model components for developing new EA artifacts to save cost, alleviate maintenance effort, and help keep the repository consistent for future (re)use.

Keywords: enterprise architecture (EA); agile; lean; repository; program slicing; model slicing

check for updates

Citation: Guo, H.; Li, J.; Gao, S.; Smite, D. A Method for Enterprise Architecture Model Slicing. *Appl. Sci.* **2022**, *12*, 9604. https://doi.org/10.3390/app12199604

Academic Editor: Christos Bouras

Received: 8 August 2022
Accepted: 14 September 2022
Published: 24 September 2022

1. Introduction

Enterprise Architecture (EA) is defined as "fundamental concepts or properties of an enterprise in its environment and governing principles for the realization and evolution of this entity and its related life cycle processes" [1]. EA has been applied widely to address communication issues and align business and IT. However, traditional EA practices follow formal processes with pre-defined frameworks and pursue extensive and rigid upfront planning [2], which might bring a heavy workload and limit EA's application in many organizations [3].

Theories and agile EA practices were proposed to reduce the workload and improve the cost-effectiveness of EA implementation [4,5]. Compared with traditional EA practices, agile EA is more lightweight, suggesting following an informal or no specific process and developing and using EA for particular purposes when needed (e.g., to catch rapid business changes) [2]. Studies, e.g., [6], following this direction, indicate that reusing existing artifacts could reduce the cost and alleviate the effort of maintaining an EA Repository (EAR). However, specific methods/techniques to find relevant EA models in a repository for reuse are still missing.

In this research, we propose to "borrow" (adapt) the methods and techniques from Program Slicing (PS) [7] to support the model reuse in the EA field. PS was originally introduced as a method to extract a minimal form of interest information from a software system [7] and has been widely used in various software engineering activities, such as understanding, debugging, reusing, and maintaining programs. To the best of our knowledge, no study has addressed how to leverage slicing methods in an EAR for reusing EA models and save costs of developing new EA artifacts.

We pursue to answer the Research Question (RQ): *How to find the relevant set of models in an EAR to develop new EA artifacts*? We propose a method of EA Model Slicing or EA Slicing

(EAS) for short by adapting existing Program Slicing (PS). We validate the EAS method with a case study. The result shows that the proposed method helped us find the expected set of EA components and relations in an extensive repository effectively and flexibly.

The remains of this article are organized as follows. Section 2 briefly introduces background information about EA and PS. Section 3 introduces the research method. In Section 4, we formalize our EAS method. We validate the method with a case study in Section 5. Then we discuss the method application in Section 6 and conclude the paper in Section 7.

2. Background and Related Work

2.1. EA Basics

EA is often referred to as a blueprint for enterprise composition and enterprise operating systems. EA has brought many benefits [8], including facilitating communication and aligning business and IT [9]. Despite EA's many benefits [8], one of EA's main roles is to provide the service to understand and communicate enterprise interaction patterns through abstract and graphical expressions. Another role is to facilitate the alignment of business and Information Systems (IS) in an enterprise [9].

EA usually covers the high-level content of an organization across areas, including strategy, business, information, and technology. EA describes some concepts and relations among them in such areas and is presented in the form of a set of abstract graphics. We call abstractions presented in one single document an **EA artifact**.

EA is traditionally developed based on one or more **EA Frameworks** (**EAFs**), which provided a common foundation for EA practitioners. For example, **TOGAF** [10] is maintained by the international standardization organization, namely **The Open Group,** and is one of the most widely used EAFs. An EAF usually consists of two parts. One part is a **content framework**, which mainly describes what concepts can be included in an EA and the relations among them. The other part is a **development method**, which provides guidelines for developing EA artifacts.

A **metamodel** is often used for a content framework to accurately define the concepts (both syntax and semantics) and the relations between them. The metamodel is usually one of the most critical components of an EAF. Since the primary form of EA is typically a set of graphical models, content frameworks/metamodels are often associated with a set of graphic **notations**. For example, the **ArchiMate** standard [11] hosted by The Open Group includes a set of symbols fully compatible with the TOGAF metamodel.

When a well-defined metamodel is applied, it is possible to organize and maintain a set of EA artifacts as **EA data** in a digital repository. Such an EAR, therefore, comprises a set of EA **components** and **relations** among them. An EA artifact can be considered a visualization of some subsets of a repository. Modern **EA Management (EAM) tools** [12] usually leverage such an EAR to capture and connect context and information across different domains (e.g., business, information, solution, and technology), along with other relevant architectural viewpoints, and to support strategic and tactical decision-making.

Traditionally, EAM practices follow a pre-defined framework (e.g., TOGAF), which means: (1) EA development process is pre-defined (e.g., TOGAF ADM), formal, separated from other projects, and pursues rigid and extensive upfront planning [2], (2) developing EA models according to a content framework (usually a well-defined metamodel and a set of notations, such as ArchiMate) so that the models are well organized and connected. The benefits of traditional EA practices are that EA models are rigorous and aligned across the organization. However, one major challenge is EA development is not cost-effective, or the development cost is difficult to estimate. On the one hand, the value of developing each separate EA model is unclear. On the other hand, development is challenging, and the workload is heavy. The low cost-effectiveness of EA development leads to participants' low motivation or disorientation. It also might lead to over-development of EA, turn the overall EA development into an impossible task, and even fail the final EA application. In addition, as changes are happening increasingly rapidly nowadays, such prescribed and proactive ways of using EA [2] could not meet the flexible and changing requirements well.

Agile EAM practices were proposed in academia and industry [4,5] to overcome the challenge of low cost-effectiveness of EA development and embrace changes. According to Agile EAM principles, all artifacts/models to be developed should deliver a clear business value. The quality and quantity of such models are limited to just enough to provide value. Doing so can significantly improve the cost-effectiveness of individual EA artifacts/model development. However, different artifacts might not be connected or aligned in an ideal way, which can be alleviated by maintaining a single digital repository based on a common metamodel. That means **Agile EAM practices** have the following characteristics. (1) The development process is not pre-defined and rigid, but to follow an informal one and develop and use EA when needed (e.g., to respond to dramatical business changes) [2] and separately, each for a specific purpose. One example is the Business Outcome Driven EA proposed by Gartner [13]. (2) Based on the only EAR, EAs are developed for specific business purposes and are accumulated gradually. It was advocated to have "just enough, just in time" architecture and not design EA until needed. All the independently developed models can be accumulated in a single repository, so that can be used collaboratively, like a puzzle.

In short, agile EA weakens the development method/process part of EAFs but strengthens the content framework/metamodel part.

2.2. EAR Management and Data Reuse

Effective **EAR management** is vital for realizing the more significant potential of EA. This is because it means developing new artifacts that can be based on existing EA data (to save cost and improve consistency). It also means the fusion of multiple EA artifacts to improve the alignment among EA artifacts. In addition, the utilization of various computing power for analysis, prediction, and visualization of the EA artifacts can be employed to enable more empowerment.

Traditional EAR management has not sparked much discussion [14], because it was assumed that EA data are developed according to a (pre-defined and fixed) metamodel, and the data are complete, structured, rigorous, and consistent. For instance, in [15], BPMN choreography diagrams can be automatically generated based on existing models stored in a web-based repository that uses the BPMN ontology to describe each component's structural properties. Furthermore, in [16], metadata about the content of UML class diagrams can be automated and archived into a pre-existing repository.

However, modern Agile EAM practices may bring new challenges for EAR management and model reuse. The pursuit of more non-professional participation [17], rapid model development and validation, and high Return on Investment (ROI) for model development around a specific purpose can lead to reduced model quality, which challenges EAR management. Researchers noticed such potential repository pollution issues and proposed to use machine learning techniques to avoid adding duplicates to the repository [18].

Reusing existing model data might be challenging because EARs are usually complex and large, consisting of many components and intricate relationships in multiple dimensions, making them difficult for a human to grasp completely. Moreover, such repositories rely on a scattered accumulation of workforce because it is often created by humans and aims for humans (to understand). As a result, the rigor and consistency of the data may be limited. The complexity, significant volume, and low consistency of EARs make reusing model data difficult for humans or computers. The vague requirements to develop new models make model reuse even more difficult. To make it clear, reusing existing model data in this article does not mean integrating data from other sources, but recommending and reusing data in the existing model repository for new views/purposes.

Many relations exist implicitly within multiple views. Thus, it is difficult to remember and find all relevant model data by humans. Due to the diversity of underlying metamodels and the inconsistency of the data, it is not easy to implement an automatic search with a pre-written (one-time) script.

2.3. Program Slicing

Weiser [7] proposed **PS** as a method or a task to make an abstract of a program to a minimal form. The reduced program is called a "slice," which guarantees representing the original program within the domain of interest. PS was widely used and accepted in various software engineering tasks, including code debugging, integration, dataflow testing, and maintenance [19].

To extract a slice, a **slicing criterion** <S, V> should be defined to specify a location/statement (S) and a variable (V) [20]. There are many types of program slices. For instance, Weiser initially used executable backward static slices [20]. PS can also be dynamic or conditional. Different **algorithms** have been applied to extract program slices. Some algorithms rely on a control flow graph following [20] and can be thought to solve a data flow problem and a graph reachability problem [20]. Basically, a directed graph G can be defined as a set of nodes N and a set of edges [20]. A control flow graph for program *P* is a graph in which each node is associated with a statement in *P*, and the edges represent the control flow in *P* [20]. More recent work tends to use the system dependence graph [21]. In general, the process of PS consists of four steps: Relation Extraction, Criteria Deciding, Slice Generation, and Slice Usage.

2.4. Model Slicing

The application of slicing in the model domain is called **Model Slicing** [22]. Model Slicing, however, usually relies on specific restrictions (e.g., model transformation rules and model constraints [23]) and is only applied to certain types of models (e.g., UML models [22] or finite-state-machine-based models [24]).

In the EA field, however, many diverse models are embraced. In addition, the consistency between models might not be guaranteed because many models are created manually. The complexity/diversity and inconsistency issues might limit the applicability of slicing techniques in the EA field. Levashova et al. [25] studied slicing EA ontologies for context retrieving, and Jacobs et al. [26] studied constructing an EA Framework (EAF) for slicing purposes by leveraging data warehouse theories.

However, to our knowledge, no study has addressed how to leverage slicing methods in an EAR for reusing EA artifacts and saving costs of developing new EA artifacts.

3. Research Method

There are two main paradigms in IS research: behavioral science and design science [27,28]. The behavioral science paradigm seeks to develop and justify theories that explain or predict human or organizational behavior. In contrast, the design science paradigm is more problem-solving and aims to create artifacts to extend the boundaries of human or organizational capabilities. For the proposed research question, we pursue a solution contributing to reusing EA models and employ the **design science** method.

According to [29], **design** can be both a process (set of activities) and a product (artifact). As identified by [30], four types of design **artifacts**/products (constructs, models, methods, and instantiations) can be produced by design science research in IS. Among them, a **method** defines processes and guides how to solve problems. We aim to construct a method as the design artifact to describe how to extract and reuse existing data in an EAR. Design **processes**/activities include building and evaluating artifacts. For design science research, it is crucial to find a proper balance between rigor and relevance [27]. While rigor is usually achieved by applying existing foundations and methodologies appropriately, framing research activities to address business needs could assure research relevance.

When **building** our proposed method, we selectively applied existing foundations and methodologies to achieve rigor [27]. We examined and adapted the existing PS techniques. We analyzed and found commonalities between the two areas (e.g., program reuse and EA reuse) concerning the problem to be solved/goal, solution, criteria pattern, and algorithm premise. The commonalities support us in applying the PS method to the EA domain. We then adapted the PS method to derive our EAS based on the EAS's unique challenges. We applied pseudo-codes, case studies, and testing to achieve rigor to **validate** our method. We used a case study to provide an in-depth observational evaluation [27] in a business environment. We used pseudo-codes and executed reference implementations to provide structural and functional testing [27]. The official case study hosted by The Open Group was selectively chosen to ensure the relevance of our approach to appropriate business problems. In this case, the EAR was modeled in ArchiMate modeling language, which conforms to the TOGAF EAF. Both TOGAF and ArchiMate are the most used international standards in the EA field. More details about the case are provided in Sections 4 and 5.

4. A Method for EA Model Slicing

We first analyzed the commonalities between the programming field and the EA field. The results indicated the potential of adapting slicing approaches in the programming field in the EA field. We adopted similar parts in the programming field in EAS. Then, according to EAS's unique feature/requirement, we developed the particular components (of the slicing method/algorithm) for the EA field. We formalize the overall method/algorithm as follows.

4.1. Adapting PS to Develop EAS

The following insights about the **commonalities** between the two areas (Program reuse and EA reuse) support us in deriving parts of EAS similar to PS.

- Goal: Both PS and our proposed EAS aim to **save costs** on software engineering tasks. For PS, such tasks are program re-development and program maintenance. While for EAS, such tasks might include new model development and maintenance.
- Solution: To save costs, both methods exclude irrelevant lines of codes/model components that are not "of interest" and try to find the **minimum subset** of the original program/repository.
- Criteria pattern: Criteria are needed to clearly define how to select the minimal subset, namely, what defines the "interest." Both PS and EAS employ a similar criteria pattern specifying **a location** (statement/view) and **a variable/component**.
- Algorithm premise: Both algorithms of PS and EAS rely on a **graph-like structure** provided by the original program/repository. PS depends on a control flow graph with nodes and edges. In comparison, an EA model repository consists of components and relations between them and is naturally organized as a graph.

  In addition, we require more **unique features** to slice EA data.

- Flexible parts of Criteria: Criteria define what makes a subset of the original program/repository concerning a criterion of <S, V> be "of interest." For PS, in a slice, the value of V at the position of S should remain the same as that in the original program. Such a **precise** and **unambiguous** criterion could be found for PS because programs are usually created by professional programmers, executed by computers, and therefore are formal and rigorous. However, finding a similar criterion for EAS might not be easy. Under many circumstances, models are **not accurately** defined or used because they are usually created by and used for humans who might not have specific expertise. In other words, we might not be able to define the criteria in EAS precisely due to the diversity of metamodels, inconsistent model data, and different requirements of using a slice in advance. Therefore, the EAS method might need to allow more flexibility (to allow for manual screening) for the criteria in EAS.
- Flexible parts of Algorithms: To address the flexible part of the criteria, we propose to introduce manual intervention in addition to a computerizable slicing algorithm. While the computerizable algorithm captures the fixed and accurate part of the criteria,

the manual screen examines the flexible parts and decides on inclusion/exclusion based on the actual settings.

To summarize, the original version of PS starts from a location/statement, then continuously finds new points/statements along the edge/flow in the program's control or data flow and judges whether it is relevant to the requirement according to the pre-defined criteria, and when to stop traversing. Our proposed EAS works similarly. The difference is that the traversal and judgment of PS are automated, while the judgment in our version is partially manual. As mentioned above, the data in the EAR may have some inconsistent issues, such as missing attributes or misuse of similar relations. We perform automated traversal and preliminary screening with pre-set criteria and then use manual screening to handle data inconsistencies.

4.2. Formalizing the EAS Method

According to the similarities and differences between PS and EAS, we formalize our method following a similar process as PS, as shown in Table 1. Like PS, we use **Step 1**, Extract information, to review the EAR and get a general knowledge of its metamodel and model data quality. In **Step 2**, we decide the specific criteria based on the requirements of the slice (e.g., for new EA artifacts development) and the EAR settings. After that, we conduct automatic searching to get the subset of the original repository in **Step 3**. Unlike PS, due to the complexity and inconsistency issues of EARs, we introduce manual screening in **Step 4** to exclude irrelevant/invalid components within the automatic searching results.

Table 1. The proposed method of EAS.

Steps	Activities
Step 1: Extract information	Review and get a general knowledge of the repository settings and quality.
Step 2: Decide criteria	Decide the criteria based on repository settings and requirements.
Step 3: Automatically search (with the automatic algorithm)	Automatically search related components for the given criteria.
Step 4: Manually screen (with the manual algorithm)	Manually screen the result from the automatic search.
Step 5: Integrate and use the slice	Recurse, integrate the result, and use the slice (e.g., for new EA artifact development).

After Step 4, for each component in the search results, Steps 1–4 will be recalled in a **recursive** way to find additional indirectly related components. We present this part as a simple recursion in Figure 1, and a more accurate definition of the recursion process is written in pseudo-codes and is detailed later. At last, in **Step 5**, we integrate the result based on all the intermediate results throughout the recursive process and use the slice.

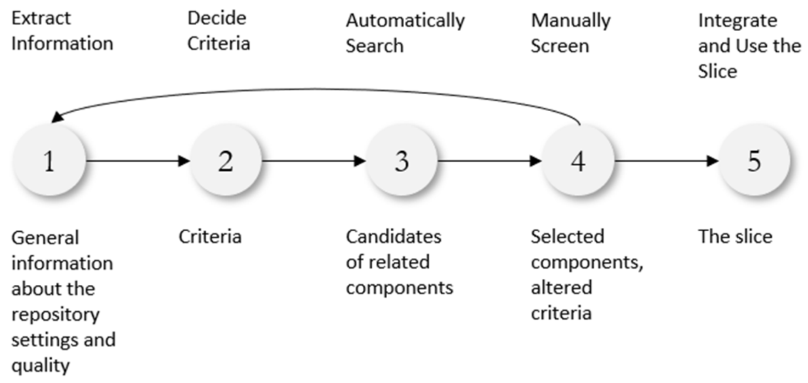
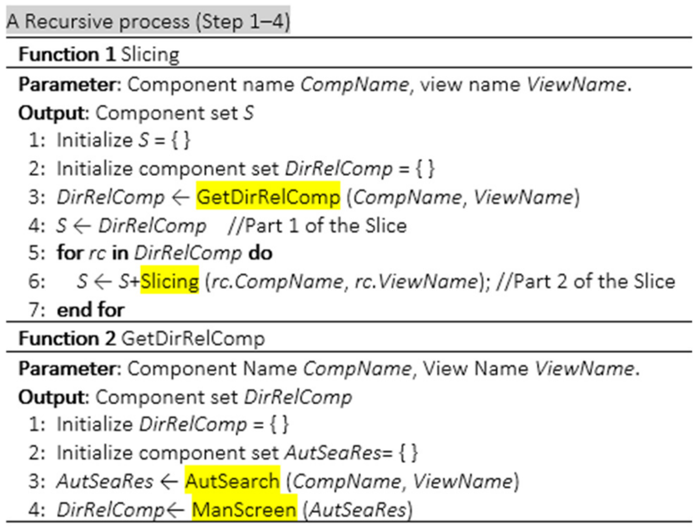

Extract Information
Decide Criteria
Automatically Search
Manually Screen
Integrate and Use the Slice
1
2
3
4
5
General information about the repository settings and quality
Criteria
Candidates of related components
Selected components, altered criteria
The slice

Figure 1. The process of the proposed EAS method.

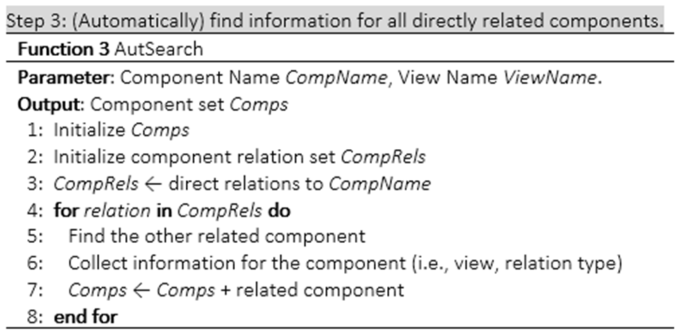
4.3. Formalizing the EAS Algorithm

With this, we formalize the overall slicing algorithm and main functions of the algorithm in pseudo-codes. In the pseudo-codes (Figures 2–4), keywords for the control flow are bolded, while comments are shadowed. We also highlight the function calls to show the overall logic.



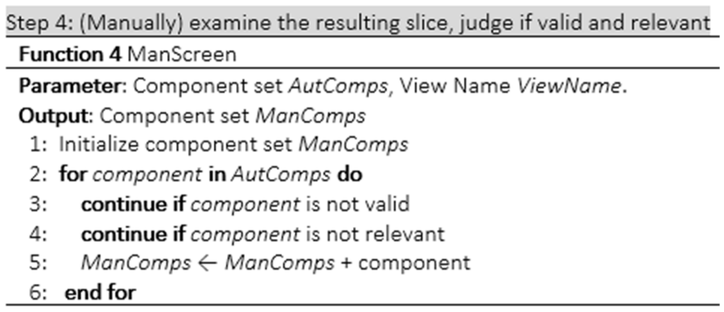
```
A Recursive process (Step 1–4)
Function 1 Slicing
Parameter: Component name CompName, view name ViewName.
Output: Component set S
 1: Initialize S = { }
 2: Initialize component set DirRelComp = { }
 3: DirRelComp ← GetDirRelComp (CompName, ViewName)
 4: S ← DirRelComp    //Part 1 of the Slice
 5: for rc in DirRelComp do
 6:     S ← S+Slicing (rc.CompName, rc.ViewName); //Part 2 of the Slice
 7: end for
Function 2 GetDirRelComp
Parameter: Component Name CompName, View Name ViewName.
Output: Component set DirRelComp
 1: Initialize DirRelComp = { }
 2: Initialize component set AutSeaRes= { }
 3: AutSeaRes ← AutSearch (CompName, ViewName)
 4: DirRelComp← ManScreen (AutSeaRes)
```

Figure 2. Pseudo-codes for the overall recursive slicing process (Step 1–4).



```
Step 3: (Automatically) find information for all directly related components.
Function 3 AutSearch
Parameter: Component Name CompName, View Name ViewName.
Output: Component set Comps
 1: Initialize Comps
 2: Initialize component relation set CompRels
 3: CompRels ← direct relations to CompName
 4: for relation in CompRels do
 5:     Find the other related component
 6:     Collect information for the component (i.e., view, relation type)
 7:     Comps ← Comps + related component
 8: end for
```

Figure 3. Pseudo-codes for the automatic searching algorithm (Step 3).



```
Step 4: (Manually) examine the resulting slice, judge if valid and relevant
Function 4 ManScreen
Parameter: Component set AutComps, View Name ViewName.
Output: Component set ManComps
 1: Initialize component set ManComps
 2: for component in AutComps do
 3:     continue if component is not valid
 4:     continue if component is not relevant
 5:     ManComps ← ManComps + component
 6: end for
```

Figure 4. Pseudo-codes for the manual screening algorithm (Step 4).

In EAS, the slicing algorithm is recursive. As shown in the upper part of Figure 2, the resulting slice consists of two parts. One includes all directly related components called "direct slice" (Line 4). Another contains slices for all components in the "direct slice" (Line 6). We use *GetDirRelComp* to get directly related components and construct the recursion with the *Slicing* function call.

In the lower part of Figure 2, we show "*GetDirRelComp*" contains automatic searching and manual screening (with corresponding pseudo-codes shown in Figures 3 and 4). By combining the automatic and manual tasks in an interwoven way, the algorithm realizes the searching/filtering for all components that (directly or indirectly) are related to a given component (within a particular view) in the entire EAR, thereby achieving the purpose of slicing.

As indicated by the pseudo-code in Figure 3, the algorithm for the automatic searching task in Step 3, for the given component in a particular view, is about finding information of all (directly) related components (Line 6), such as relation names and view names. This task executes automatically so that the structured retrieving can be conducted quickly and reliably. This task also maps user-friendly information such as component name to IT (Line 3) to facilitate automatic searching, or vice versa to facilitate human understanding (Line 7). The result of this task will provide candidates for the subsequent manual screening task.

The pseudo-code in Figure 4 demonstrates the algorithm for the manual screening in Step 4, which manually selects relevant components based on the information provided in Step 3. Manual screening includes correlation judgment (Line 4) and consistency verification (Line 3).

5. Evaluation

We designed a proof-of-concept case study to demonstrate our proposed method's use and prove the concepts' validity. In this section, we introduce the settings of the case study, present the results, and reflect on the results.

5.1. Settings

Our case study is an insurance enterprise that maintains a digital EAR and needs to develop a new roadmap artifact. The existing EAR has been used to capture and connect the context and information across domains (e.g., business, information, solution, and technology) to support strategic and tactical decision-making. We employ the official repository [31] hosted by The Open Group [32]. In this case, the repository [33] is modeled with the ArchiMate modeling language [11] and stored with ArchiMate Model Exchange File Format [34]. We suppose a new **roadmap** artifact is required because the management wants a coherent and straightforward picture representing the overall strategic planning. A roadmap artifact is usually defined as "*structured graphical views of all planned IT initiatives in specific business areas having direct business value*," serving to "*achieve clearer traceability between the business strategy and future IT investments*" [3]. Therefore, we need to find out existing components and relations that are relevant to this view. Our proposed EAS method will be used to extract relevant components in the repository and lower the cost of new artifact development through reuse.

To reuse existing model data to develop the new roadmap artifact, we conducted several slicing **tasks** to get relevant components concerning several strategic components. Due to the space limitation, we demonstrate the process of using our proposed EAS method for one of the tasks. The task is to find all existing components/IT initiatives that directly or indirectly support one of the planned *Business Output*, namely, *Detailed Insights in Customer* in the view of *Business Strategy*.

Here four **requirements** for the slicing are implicitly given: (1) "support" indicates that a relation with a meaning of "support" (e.g., *Realization* and *Serving*) connects some component to others; (2) "directly support" means such relation directly connects one component and the other, while "indirectly support" happens when more than one such relation connects or other relations (e.g., *Aggregation*) are involved additionally; (3) the resulting components should be at a high level and cannot contain more details than that of general IT initiatives; (4) as a general non-functional requirement, the model data should be of acceptably high quality, which means that the data should be at least reasonable to humans and consistent with other data in the repository.

Note that although these requirements look clear and comprehensive enough for a human, they are still somewhat vague to be formalized in programs. Take the first requirement, for instance, we as humans understand the general meaning of "support".

However, we could not determine the exact types of relations to apply in a particular repository due to various metamodels that can be applied in the EA fields. Other examples include what defines an "indirect" relation in the second requirement, what components are sufficiently "high-level" in the third requirement, and what kind of data inconsistency issues are unacceptable in the fourth requirement. They are all the "undefined" or "vague" part of the requirements. Such flexible requirements are the primary reason we adopt a hybrid method to combine automatic searching and manual screening.

We used existing **tools** for our demonstration and concept-proof validation. We used Archi 4.8.1 [35], an open-source ArchiMate modeling tool, for reviewing the EAR in a visualized way (Step 1). An Extensible Markup Language (XML) viewer is also used in Step 1 to help us examine the repository serialized in the exchange file format. We employed an Integrated Development Environment (IDE) to implement the criteria/algorithm and run the automatic searching (Step 3) in Python 3.8. We used a text editor to record the recursive slicing process and all intermediate results (Step 4). At last, we used Archi [35] to present the integrated results of the slice (Step 5) to develop the new artifact based on it.

5.2. Results

According to EAS, we recursively conduct the slicing process. First, we find components that "directly support" the starting component. Then, for each component found, we conduct Steps 1–4 to find the components that "directly support" it. We perform this process recursively until finding no more components. Then we take all the components and relations found throughout the process and compile the final result. We provide more details for the first loop as an example.

In **Step 1**, we browse the entire repository to capture the general information in a visualized way in Archi (as shown in Figure 5) and in a serialized exchange file format in an XML viewer (see Figure 6). Such general information includes the overall library size, major existing views, and types of components and relations used in the repository. The result shows that, in the repository, 328 EA *Components* and 701 *Relations* are presented in 71 EA views spanning different layers of the enterprise, including strategy, business, and IS. The relations include 643 common relations, such as *Serving* and *Realization*, 39 *Grouping*, and 19 *Junction* relations.

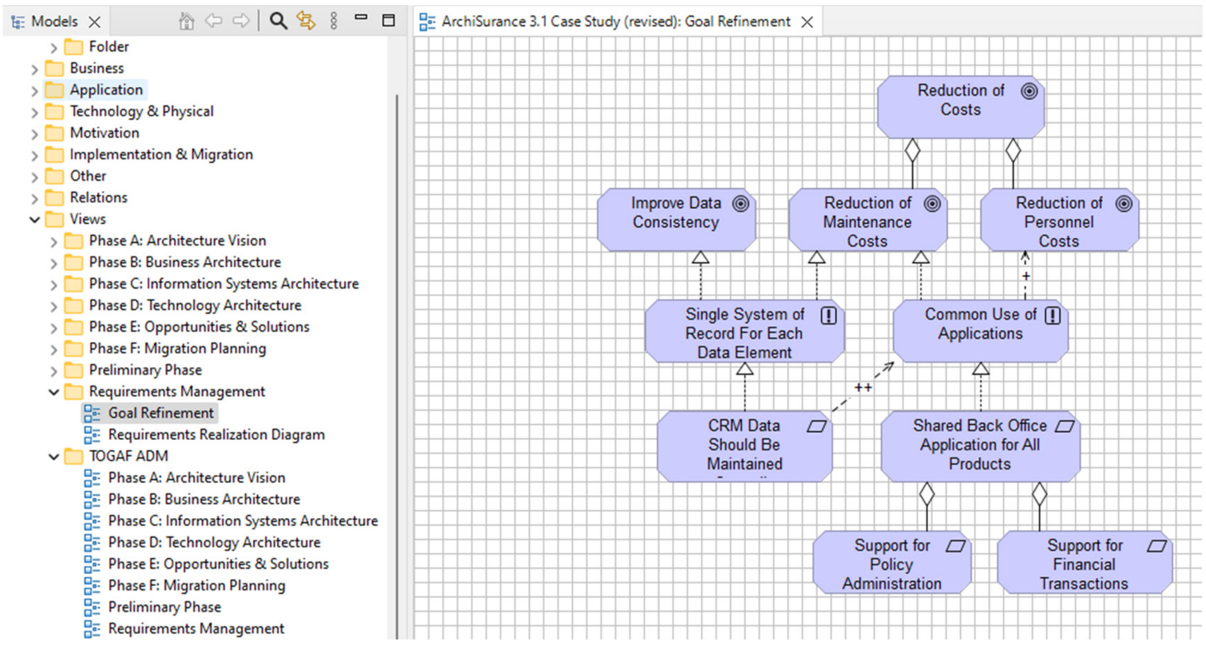
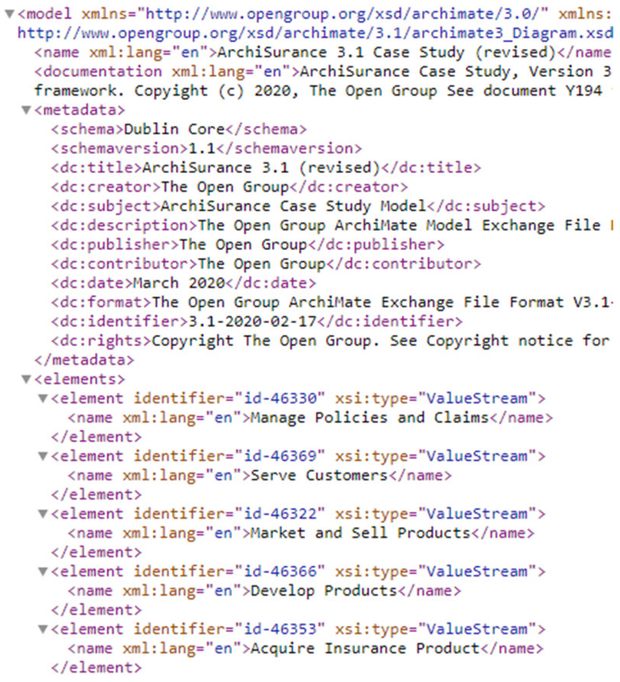

Models
Folder
Business
Application
Technology & Physical
Motivation
Implementation & Migration
Other
Relations
Views
Phase A: Architecture Vision
Phase B: Business Architecture
Phase C: Information Systems Architecture
Phase D: Technology Architecture
Phase E: Opportunities & Solutions
Phase F: Migration Planning
Preliminary Phase
Requirements Management
Goal Refinement
Requirements Realization Diagram
TOGAF ADM
Phase A: Architecture Vision
Phase B: Business Architecture
Phase C: Information Systems Architecture
Phase D: Technology Architecture
Phase E: Opportunities & Solutions
Phase F: Migration Planning
Preliminary Phase
Requirements Management
ArchiSurance 3.1 Case Study (revised): Goal Refinement
Reduction of Costs
Improve Data Consistency
Reduction of Maintenance Costs
Reduction of Personnel Costs
Single System of Record For Each Data Element
Common Use of Applications
CRM Data Should Be Maintained
Shared Back Office Application for All Products
Support for Policy Administration
Support for Financial Transactions
++
+

Figure 5. Browsing the use case repository in Archi.

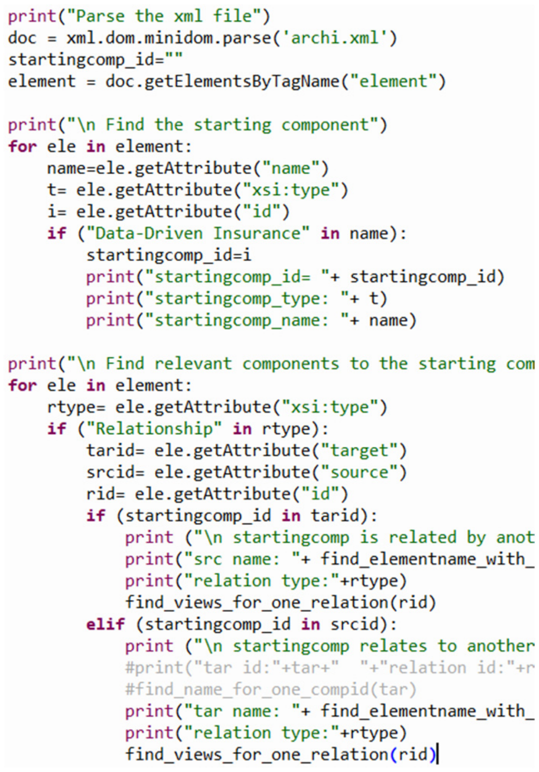


```
▼<model xmlns="http://www.opengroup.org/xsd/archimate/3.0/" xmlns:
 http://www.opengroup.org/xsd/archimate/3.1/archimate3_Diagram.xsd
   <name xml:lang="en">ArchiSurance 3.1 Case Study (revised)</name
   <documentation xml:lang="en">ArchiSurance Case Study, Version 3
   framework. Copyright (c) 2020, The Open Group See document Y194
  ▼<metadata>
     <schema>Dublin Core</schema>
     <schemaversion>1.1</schemaversion>
     <dc:title>ArchiSurance 3.1 (revised)</dc:title>
     <dc:creator>The Open Group</dc:creator>
     <dc:subject>ArchiSurance Case Study Model</dc:subject>
     <dc:description>The Open Group ArchiMate Model Exchange File
     <dc:publisher>The Open Group</dc:publisher>
     <dc:contributor>The Open Group</dc:contributor>
     <dc:date>March 2020</dc:date>
     <dc:format>The Open Group ArchiMate Exchange File Format V3.1
     <dc:identifier>3.1-2020-02-17</dc:identifier>
     <dc:rights>Copyright The Open Group. See Copyright notice for
   </metadata>
  ▼<elements>
    ▼<element identifier="id-46330" xsi:type="ValueStream">
       <name xml:lang="en">Manage Policies and Claims</name>
     </element>
    ▼<element identifier="id-46369" xsi:type="ValueStream">
       <name xml:lang="en">Serve Customers</name>
     </element>
    ▼<element identifier="id-46322" xsi:type="ValueStream">
       <name xml:lang="en">Market and Sell Products</name>
     </element>
    ▼<element identifier="id-46366" xsi:type="ValueStream">
       <name xml:lang="en">Develop Products</name>
     </element>
    ▼<element identifier="id-46353" xsi:type="ValueStream">
       <name xml:lang="en">Acquire Insurance Product</name>
     </element>
```

Figure 6. One section of the case repository exchange file in an XML viewer.

In **Step 2**, we define the initial slicing criterion as <*Detailed Insights in Customer*, *Business Strategy*>, indicating that the slice should include all valid components that "directly support" the starting component of *Detailed Insights in Customer* in the view of *Business Strategy*.

This criterion and the corresponding algorithm were programmed in Python (as shown in Figure 7) in **Step 3**. After running the automatic searching (as shown in Figure 8), we achieve five relations that connect "*Detailed Insights in Customer*" with other components.

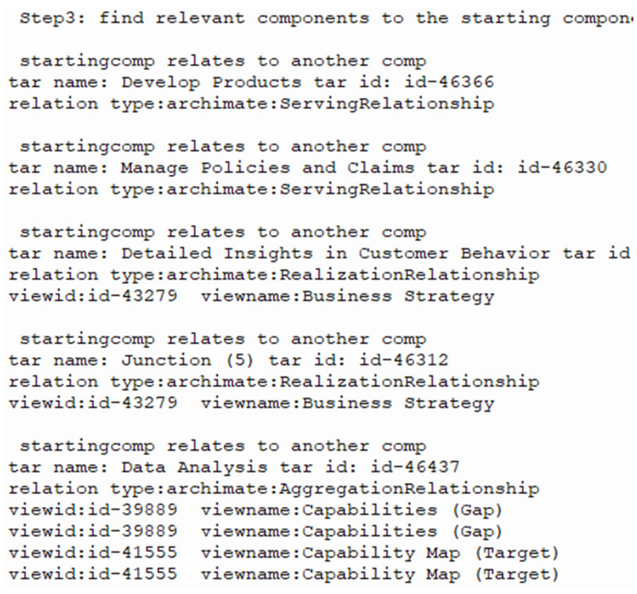


```
print("Parse the xml file")
doc = xml.dom.minidom.parse('archi.xml')
startingcomp_id=""
element = doc.getElementsByTagName("element")

print("\n Find the starting component")
for ele in element:
    name=ele.getAttribute("name")
    t= ele.getAttribute("xsi:type")
    i= ele.getAttribute("id")
    if ("Data-Driven Insurance" in name):
        startingcomp_id=i
        print("startingcomp_id= "+ startingcomp_id)
        print("startingcomp_type: "+ t)
        print("startingcomp_name: "+ name)

print("\n Find relevant components to the starting com
for ele in element:
    rtype= ele.getAttribute("xsi:type")
    if ("Relationship" in rtype):
        tarid= ele.getAttribute("target")
        srcid= ele.getAttribute("source")
        rid= ele.getAttribute("id")
        if (startingcomp_id in tarid):
            print ("\n startingcomp is related by anot
            print("src name: "+ find_elementname_with_
            print("relation type:"+rtype)
            find_views_for_one_relation(rid)
        elif (startingcomp_id in srcid):
            print ("\n startingcomp relates to another
            #print("tar id:"+tar+"  "+"relation id:"+r
            #find_name_for_one_compid(tar)
            print("tar name: "+ find_elementname_with_
            print("relation type:"+rtype)
            find_views_for_one_relation(rid)
```

Figure 7. Sample code snippets for automatic slicing in Python.



```
 Step3: find relevant components to the starting compon

 startingcomp relates to another comp
tar name: Develop Products tar id: id-46366
relation type:archimate:ServingRelationship

 startingcomp relates to another comp
tar name: Manage Policies and Claims tar id: id-46330
relation type:archimate:ServingRelationship

 startingcomp relates to another comp
tar name: Detailed Insights in Customer Behavior tar id
relation type:archimate:RealizationRelationship
viewid:id-43279  viewname:Business Strategy

 startingcomp relates to another comp
tar name: Junction (5) tar id: id-46312
relation type:archimate:RealizationRelationship
viewid:id-43279  viewname:Business Strategy

 startingcomp relates to another comp
tar name: Data Analysis tar id: id-46437
relation type:archimate:AggregationRelationship
viewid:id-39889  viewname:Capabilities (Gap)
viewid:id-39889  viewname:Capabilities (Gap)
viewid:id-41555  viewname:Capability Map (Target)
viewid:id-41555  viewname:Capability Map (Target)
```

Figure 8. Intermediate results of performing an EAS automatic searching.

Then, in **Step 4**, we conduct a manual validation by examining each relation and checking if the involved relation, related component, and view are reasonably relevant and valid according to the criteria. We found that among the five relations, only one of them is valid and relevant to our specific goal. This relation indicates that the "Data-Driven Insurance" component realizes the starting component of "Detailed Insights in Customer Behavior". Later, we decided to further track the EA components of the slicing result by starting a new iteration because they are a higher level of abstraction than the expected "IT plan".

We conducted **four iterations in total**. In the last iteration, we found that all relations in the slicing result are irrelevant or invalid according to the criteria. The observation indicates that all components of interest have been retrieved in previous iterations. Therefore, we stopped the slicing process. The resulting slice contains four components and three relations from two original views. They are reasonable and valid data concerning the slicing goal. We then presented them (by dragging them from the original to the new views) to reuse them for developing the new artifact, as shown in Figure 9. Note that we examined (manually screened) 18 relations within 22 views throughout the recursive slicing process. The numbers of components, relations, and views of the original repository, manual screening, and the resulting slice, are recorded and compared, as Table 2 shows.

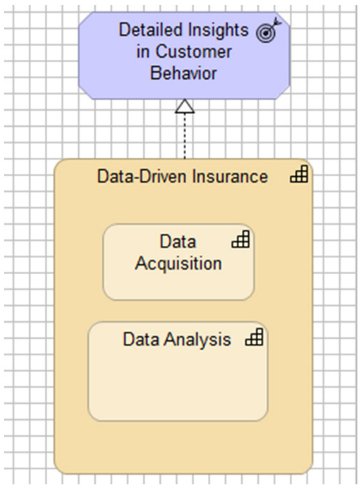


Figure 9. Using the resulting slice for a new EA artifact development.

Table 2. The number of components, relations, and views before and after slicing.

	Component Number	**Relation Number**	**View Number**
Original repository	328	701	71
Manual screening	18	18	22
Resulting slice	4	3	2

5.3. Reflections

Our experience shows that simple manual or automatic methods are indeed challenging to achieve the slicing goals in such a representative EA context for the following reasons.

First, it is difficult, if not impossible, for humans to exhaustively and reliably identify all relevant relations distributed in many views. We felt it was impossible to remember all the relevant components. We needed to focus on 18 out of 701 relations in our case (as shown in the second and third lines of Table 2). Secondly, when we need to validate and extract more detailed information, it is very costly to traverse across a large number of views. This means checking 18 relations in 22 views in our context (as shown in the third line of Table 2). Manual operation is not only time-consuming and labor-intensive but also easy to bring inconsistent and unreliable data.

In addition, for automatic methods (pre-defined or one-time defined programming methods), it is not easy to cope with various previously mentioned undefined/vague requirements. Firstly, due to the different metamodels used in EARs and the diverse preferences of model developers, the relevant slicing principles and algorithms need not be defined or programmed precisely. For example, in our case, we only selected components that *Realize* the starting component for Realization relations. However, for the Aggregation relations, we selected components that *are Aggregated by* the starting component. Secondly, many human-made models have various inconsistency issues which need to be handled flexibly. For example, in our case, we found that there was no relevant view for some relations. For some relations, the related component did not have a name. By observing more relevant information, we think they are possibly caused by previous unregulated human operations (the models were deleted in views but not in the repositories, for instance). We, therefore, simply ignored them.

In contrast to pure manual or automatic methods, our proposed method combines the advantages of manually making intelligent decisions and the computational power of automation. The EAS method can flexibly handle the metamodel complexities and data inconsistencies. It can also traverse the database sufficiently and reliably in a cost-effective way.

6. Discussion

We summarize the **benefits** of our proposed method and discuss how to **apply** our proposed method in the EA field.

6.1. Benefits of Our Proposed EAS Method

In this paper, we propose to use slicing methods in the EA field to lower the cost of developing new EA artifacts by retrieving and reusing relevant components. Considering specific characteristics in the EA field, we propose to introduce manual intervention in addition to an automatic algorithm. Such characteristics are:

(1) There are more flexible requirements in the EA field as EA models are often developed for human understanding purposes. Accordingly, the slicing criteria can hardly be defined in a very precise way.
(2) The metamodels used in the EA field can be very diverse. The analogy is to use multiple programming languages (statically or dynamically) in PS. Thus, it is challenging to implement the algorithms in advance fully.
(3) The EA model data inconsistency issues might widely exist as people manually develop many models. This imposes diverse and flexible data validation work.

However, by weaving automatic and manual tasks together, we can automatically retrieve component candidates quickly and reliably and present them with meaningful and readable information to humans. We also benefit from manual screening to ensure the results are valid and fit flexible requirements (but reasonable enough for humans) of slicing.

6.2. Balance between the Automatic and Manual Tasks

We have considered the possibility of implementing the EAS algorithm with programs completely without manual intervention. The answer is yes if a comprehensive review of the repository could be conducted in advance. However, it can be very uneconomical and difficult to maintain. Pre-programming requires comprehensive manual browsing of the entire repository and general rule extraction. While manual screening only takes care of a small subset of the repository and conducts specific judgments. Since the repository is dynamically developed and used/maintained and all the metamodels and models can be continuously changing, maintaining the slicing algorithm might be considerably time-consuming or even become a mission impossible.

However, it is possible to balance and distribute tasks in automatic or manual parts under different situations differently or even dynamically to achieve the best slicing performance.

6.3. Tool Support

Appropriate tool support would speed up and ensure the quality of EAS, especially for Step 3 and Step 4 in the slicing method. In our case, we programmed the slicing criteria/algorithms with Python in Step 3. However, note that this type of program is quite domain-specific and contains common patterns. Therefore, tools can be developed to enable no/low code programming to configure patterns rather than writing general code to configure which types of relations are considered. By doing so, not only could the slicing process be accelerated, but more stakeholders, such as business architects without programming expertise, could also benefit and conduct the process independently [17]. Tools could be used to address inconsistency issues in Step 4 by highlighting inconsistent data and providing automatic corrections, for example. They can also provide What You See Is What You Get (WYSIWYG) functions to facilitate and accelerate the manual screening in Step 4 by highlighting related components in corresponding views, for instance.

6.4. Data Validation/Dealing with Model Inconsistency

In EAS, data validation is conducted manually, and model inconsistencies (e.g., different relations used for similar purposes) are considered/solved flexibly.

All inventory data should contain time and owner information to facilitate later verification. In our case study, we are the only user/developer of the EAR. However, in actual scenarios, multiple users/developers usually operate the repository. Therefore, information about its owner and timestamp may be required for all data in a repository. In this way, when relevant data are retrieved, whether such data are valid should be verified in Step 4 according to the timestamp information. Then, if necessary, the owner could be contacted to double-check and fix/update the data before reusing it.

6.5. Inventory Maintenance

EA inventories should be maintained while using. In the present research, we propose a cost-effective method primarily for new EA artifact development by reusing EA models/data in an existing EAR. However, reusing and developing new data is not the end. In real scenarios, newly developed artifacts should be merged back into the repository. By doing so, the repository could be continuously and sustainably accumulated/maintained. Therefore, we should make sure all relevant data can be retrieved in Steps 3–4. In addition, after using the data in Step 5, newly developed EA data should be merged back properly to keep the repository consistent.

6.6. Other Slicing Techniques/Algorithms

PS techniques/algorithms could be further explored in the EA field. Many PS algorithms exist, such as static, dynamic, or conditional PS and forward or backward PS. We have only employed the basic algorithm to reveal the feasibility and potential. Considering the similarities in the two fields, we believe it would be rewarding to explore further and apply other relevant techniques/methods.

6.7. Other Application Areas

The present method might be useful in more EAM tasks and reusing for new EA artifacts development. PS has been widely applied in understanding, testing/debugging, redesigning, reusing, maintaining/evolving, and measuring programs. We believe the proposed method could be extended/adapted with a similar algorithm and principle for further application in more tasks.

In addition, we primarily discussed how this method facilitates activities for EA models. However, this method can also be applied to general model repositories if they are based on integrated metamodels and even general data repositories. For instance, Data lakes are emerging to support extracting knowledge from a large number of highly heterogeneous and rapidly changing data sources [36,37]. New techniques are needed to extract complex knowledge patterns from data stored in data lakes. The method proposed in this article might provide some new insights into this goal.

7. Conclusions

In this paper, we proposed to use a method in terms of slicing techniques to extract and reuse existing EA models. We validated the method with an official EAR hosted by The Open Group. The results show that it is possible to reuse existing EA models to develop new EA artifacts to save cost and help keep the single repository consistent for future use.

There are some limitations of the present research. First, we focused on how to reuse data in an EAR that contains inconsistent data. We have not discussed in depth how to deal with/fix such inconsistent data in the EAR systematically. Second, we validated our method mainly by a proof-of-concept case study. Future work will include developing prototype tools to explore further how to deal with inconsistency issues, maintain the EAR, and validate the method with user tests.

Author Contributions: Writing—original draft, H.G.; Writing—review & editing, J.L., S.G. and D.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partically funded by the Research Council of Norway grant number NFR 295920 (IDUN Project). And the APC was funded by Anhui University grant number Y040418077 and Ministry of Human Resources and Social Security of the People's Republic of China grant number Z010146009.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. ISO/IEC/IEEE. ISO/IEC/IEEE 42020:2019 Software, Systems and Enterprise—Architecture Processes. 2019. Available online: https://www.iso.org/standard/68982.html (accessed on 7 August 2022).
2. Kotusev, S.; Singh, M.; Storey, I. Consolidating enterprise architecture management research. In Proceedings of the 2015 48th Hawaii International Conference on System Sciences, Kauai, HI, USA, 5–8 January 2015; pp. 4069–4078.
3. Kotusev, S. Enterprise architecture and enterprise architecture artifacts: Questioning the old concept in light of new findings. *J. Inf. Technol.* **2019**, *34*, 102–128. [CrossRef]
4. Hauder, M.; Roth, S.; Schulz, C.; Matthes, F. Agile enterprise architecture management: An analysis on the application of agile principles. In Proceedings of the 4th International Symposium on Business Modeling and Software Design, Luxembourg, 24–26 June 2014; pp. 38–46.

5. Gill, A.Q. Agile enterprise architecture modelling: Evaluating the applicability and integration of six modelling standards. *Inf. Softw. Technol.* **2015**, *67*, 196–206. [CrossRef]
6. Guo, H.; Li, J.; Gao, S.; Smite, D. Boost the Potential of EA: Essential Practices. In Proceedings of the 23rd International Conference on Enterprise Information Systems, Online Streaming, 26–28 April 2021; Volume 2, pp. 735–742.
7. Weiser, M. Program slicing. *IEEE Trans. Softw. Eng.* **1984**, *4*, 352–357. [CrossRef]
8. Winter, K.; Buckl, S.; Matthes, F.; Schweda, C.M. Investigating the State-of-the-Art in Enterprise Architecture Management Methods in literature and Practice. In Proceedings of the Mediterranean Conference on Information Systems (MCIS), Tel Aviv, Israel, 12–14 September 2010; Volume 90.
9. Korhonen, J.J.; Lapalme, J.; McDavid, D.; Gill, A.Q. Adaptive enterprise architecture for the future: Towards a reconceptualization of EA. In Proceedings of the 2016 IEEE 18th Conference on Business Informatics (CBI), Paris, France, 29 August–1 September 2016; pp. 272–281.
10. The Open Group. The TOGAF®Standard. 2020. Available online: https://www.opengroup.org/togaf (accessed on 7 August 2022).
11. The Open Group. ARCHIMATE®3.1 SPECIFICATION. Available online: https://pubs.opengroup.org/architecture/archimate3-doc/ (accessed on 7 August 2022).
12. Gartner. Enterprise Architecture (EA) Tools Reviews and Ratings. Available online: https://www.gartner.com/reviews/market/enterprise-architecture-tools (accessed on 7 August 2022).
13. Gartner Research. Stage Planning a Business-Outcome-Driven Enterprise Architecture. Available online: https://www.gartner.com/en/documents/3642517/stage-planning-a-business-outcome-driven-enterprise-arch (accessed on 7 August 2022).
14. Rouhani, B.D.; Mahrin, M.N.r.; Nikpay, F.; Ahmad, R.B.; Nikfard, P. A systematic literature review on Enterprise Architecture Implementation Methodologies. *Inf. Softw. Technol.* **2015**, *62*, 1–20. [CrossRef]
15. Wiśniewski, P.; Kluza, K.; Suchenia, A.; Szała, L.; Ligęza, A. Recomposition of Process Choreographies Using a Graph-Based Model Repository. In Proceedings of the International Conference on Knowledge Science, Engineering and Management, Singapore, 6–8 August 2022; pp. 478–488.
16. Di Felice, P.; Paolone, G.; Paesani, R.; Marinelli, M. Design and Implementation of a Metadata Repository about UML Class Diagrams. A Software Tool Supporting the Automatic Feeding of the Repository. *Electronics* **2022**, *11*, 201. [CrossRef]
17. Sandkuhl, K.; Fill, H.-G.; Hoppenbrouwers, S.; Krogstie, J.; Matthes, F.; Opdahl, A.; Schwabe, G.; Uludag, Ö.; Winter, R. From expert discipline to common practice: A vision and research agenda for extending the reach of enterprise modeling. *Bus. Inf. Syst. Eng.* **2018**, *60*, 69–80. [CrossRef]
18. Borozanov, V.; Hacks, S.; Silva, N. Using machine learning techniques for evaluating the similarity of enterprise architecture models. In Proceedings of the International Conference on Advanced Information Systems Engineering, Rome, Italy, 3–7 June 2019; pp. 563–578.
19. Tip, F. *A Survey of Program Slicing Techniques*; Centrum voor Wiskunde en Informatica: Amsterdam, The Netherlands, 1994; p. 58.
20. Binkley, D.W.; Gallagher, K.B. Program slicing. *Adv. Comput.* **1996**, *43*, 1–50.
21. Horwitz, S.; Reps, T.; Binkley, D. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst. (TOPLAS)* **1990**, *12*, 26–60. [CrossRef]
22. Bae, J.H.; Lee, K.; Chae, H.S. Modularization of the UML metamodel using model slicing. In Proceedings of the Fifth International Conference on Information Technology: New Generations (ITNG 2008), Las Vegas, NV, USA, 7–9 April 2008; pp. 1253–1254.
23. Shaikh, A.; Wiil, U.K. UMLtoCSP (UOST) a tool for efficient verification of UML/OCL class diagrams through model slicing. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, Cary, NC, USA, 11–16 November 2012; pp. 1–4.
24. Androutsopoulos, K.; Clark, D.; Harman, M.; Krinke, J.; Tratt, L. State-based model slicing: A survey. *ACM Comput. Surv. (CSUR)* **2013**, *45*, 1–36. [CrossRef]
25. Levashova, T.; Lundqvist, M.; Pashkin, M. Moving towards automatic generation of information demand contexts: An approach based on enterprise models and ontology slicing. In Proceedings of the OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", Rhodes, Greece, 21–25 October 2006; pp. 1012–1019.
26. Jacobs, D.; Kotzé, P.; Van Der Merwe, A. Towards an enterprise repository framework. In Proceedings of the Joint Workshop on Advanced Technologies and Techniques for Enterprise Information Systems, Milan, Italy, 6–10 May 2009; pp. 77–89.
27. Hevner, A.R.; March, S.T.; Park, J.; Ram, S. Design science in information systems research. *MIS Q.* **2004**, *28*, 75–105. [CrossRef]
28. Tuunanen, T.; Gengler, C.E.; Rossi, M.; Hui, W.; Virtanen, V.; Bragge, J. The Design Science Research Process: A Model for Producing and Presenting Information Systems Research. Available online: https://www.researchgate.net/publication/228650671_The_design_science_research_process_A_model_for_producing_and_presenting_information_systems_research (accessed on 7 August 2022).
29. Walls, J.G.; Widmeyer, G.R.; El Sawy, O.A. Building an information system design theory for vigilant EIS. *Inf. Syst. Res.* **1992**, *3*, 36–59. [CrossRef]
30. March, S.T.; Smith, G.F. Design and natural science research on information technology. *Decis. Support Syst.* **1995**, *15*, 251–266. [CrossRef]
31. The Open Group. ArchiSurance Case Study, Version 3.1. Available online: https://publications.opengroup.org/y194 (accessed on 7 August 2022).
32. The Open Group. Available online: https://www.opengroup.org/about-us/who-we-are (accessed on 7 August 2022).

33. The Open Group. ArchiSurance Case Study, Version 3.1, ArchiMate®Model Exchange File Format. 2020. Available online: https://publications.opengroup.org/y194m (accessed on 7 August 2022).
34. The Open Group. ArchiMate®Model Exchange File Format for the ArchiMate Modeling Language, Version 3.1. Available online: https://publications.opengroup.org/c19c (accessed on 7 August 2022).
35. Beauvoir, P. Archi ArchiMate Modeling. Available online: https://www.archimatetool.com/ (accessed on 7 August 2022).
36. Giudice, P.L.; Musarella, L.; Sofo, G.; Ursino, D. An approach to extracting complex knowledge patterns among concepts belonging to structured, semi-structured and unstructured sources in a data lake. *Inf. Sci.* **2019**, *478*, 606–626. [CrossRef]
37. Diamantini, C.; Lo Giudice, P.; Potena, D.; Storti, E.; Ursino, D. An approach to extracting topic-guided views from the sources of a data lake. *Inf. Syst. Front.* **2021**, *23*, 243–262. [CrossRef]