Formalizing Enterprise Architecture Decision Models using Integrity Constraints

Marc van Zee[*], **Dirk van der Linden**[†‡], **Diana Marosin**[†‡] and **Georgios Plataniotis**[†‡]
[*] University of Luxembourg, Luxembourg
[†] Public Research Centre Henri Tudor, Luxembourg
[‡] Radboud University Nijmegen, the Netherlands

Contact: marc.vanzee@uni.lu, {diana.marosin, dirk.vanderlinden, georgios.plataniotis}@tudor.lu

***Abstract*—In creating an enterprise architecture (EA) several design decisions have to be made. The aim of this paper is to provide a logic-based formalism for capturing architectural design decisions in order to make the rationalization of these decisions explicit as well as traceable. Our working hypothesis is that capturing of design knowledge in terms of a logic-based framework will enable consistency checks of the underlying rationales and advanced impact/what-if analysis when confronted with changes (e.g. decisions are changed, issues are solved). We formalize a set of integrity constraints, which allow guidance of decision capturing during model creation and provide means to perform consistency checks. We apply our formal framework to a practical case study from the insurance sector.**

***Keywords*—Enterprise architecture, Decision making, Consistency check support, Decision graphs**

I. Introduction

Large and complex enterprises are a common occurrence in today's business environment. Such enterprises usually involve complex and interdependent business processes and IT systems. Enterprise Architecture (EA) is used to model such enterprises in a holistic fashion by connecting their IT infrastructure and applications to the business processes they support. In turn this links them also to the products and services that are realized by those business processes [1], [2]. When creating an EA, several design decisions have to be made. These decisions are to a large extent based on assumptions about the situation at hand. Such assumptions may relate to the goals the (individual) stakeholders have, strategic directions of the enterprise, architecture principles, requirements, and so on. In practice, enterprises are confronted with frequent changes and challenges to these assumptions.

Our long-term research goal is to explore the possibility of explicitly linking architecture-level design decisions with their underlying assumptions. The aim of doing so is to make the rationalization of these decisions explicit and traceable, so that we can formally reason about them in terms of a logic-based framework. This will enable explicit reasoning about the connections between the enterprise's architecture, the associated design decisions, and their underlying assumptions. Formalizing the elements in an architectural decision model has been shown to be useful for the structuring of knowledge, and the measuring of the quality of existing decisions [3]. Architects and designers who are not the original developers often have to control the quality of, and maintain the enterprise architecture. These people need a good understanding of the architecture in order to work effectively. It is not typical in EA for design rationales to be obtained from design specifications, because there is no systematic practice for capturing them. Even when some of these decisions are captured, they are not organized in such a way that they can be retrieved and tracked easily. Remedying this situation becomes critical and challenging when system requirements and operating environments continue to evolve [4]. Having a framework to formally reason about decisions and their underlying assumptions also allows for decision types and dependency patterns to be defined, which helps to detect the incompleteness or inconsistency of a decision model. Finally, knowledge engineers working in other decision capturing domains (i.e., not EA), can reuse the model structure to organize their knowledge [3].

In this paper, we contribute to our long-term research goal by formalizing a recently proposed framework for decision making in EA by Plataniotis *et al.* [5] using set and graph theory concepts. The framework of Plataniotis *et al.* consists of a metamodel that serves as a basis for decision design graphs composed of EA decisions, issues, observation impact, and several types of dependency relations. We analyze the correspondence between the metamodel and the decision design graphs, and propose a formal framework that captures the decision design graphs more precisely. Moreover, motivated by providing a better guidance on the use of the framework for a priori decision analysis and support, we extend the framework to cater for a more expressive notation of decision state, and we make precise several informally introduced concepts of Plataniotis *et al.* using integrity constraints. We apply our framework to a case study and show the benefits of our formal approach by demonstrating the possibility for *a priori* decision analysis through consistency checks on the integrity constraints.

The rest of this paper is structured as follows: In Section II we discuss the framework of Plataniotis *et al.*; In Section III we use this discussion as a motivation to present our formal framework; In Section IV we validate our framework for *a priori* decision analysis by applying it to our ArchiSurance use case; Finally, in Section V we position our work in state-of-the-art research.

II. Preliminaries

In this section we briefly review the key components of the metamodel of Plataniotis *et al.* [5], followed by a discussion.

We use these observations as a basis for the formal framework that we will introduce in the next section.

A. EA Metamodel and Decision Design Graphs

Plataniotis *et al.* [5] recently presented an approach for relating EA decisions. Using a metamodel and a decision design graph, they explain how decisions from different enterprise domains (business, application, and technology) relate to each other. For example, how decisions taken on a business level affect IT decisions and vice versa. Their approach is inspired by well-known mechanisms for capturing architectural rationales in software architecture. The metamodel that was presented by Plataniotis *et al.* is depicted in Figure 1. This metamodel serves as an underlying model for design decisions graphs, of which an example is depicted in Figure 6. From now on, we will refer to the metamodel in Figure 1 simply as "the metamodel", and the decision design graph in Figure 6 as "the decision graph". We will explain the details of the decision graph in more detail when presenting the case study, but in this section we will already use it to explain the main concepts of the framework, which consists of the following elements:

EA Decision represents a decision that has been made or rejected in order to resolve an issue. An EA decision shows decisions that are captured in the context of an Enterprise Transformation [6]. The decision graph contains a total of 13 decisions, from EA Decision D01 to EA Decision D13.

EA Issue represents an architectural design problem that enterprise architects have to address during the Enterprise transformation process. In this way, they can be regarded as a motivation for the design decisions. The decision graph contains 6 issues, from EAIssue IS01 to EA Issue IS06.

EA Artifact serves as a bridging concept towards the EA modeling language ArchiMate, whereby an EA artifact links EA decisions to ArchiMate concepts. For instance, EA Decision D01 in the decision graph is related to EA artifact "Customer profile registration Business processes". EA issues are not related to artifacts.

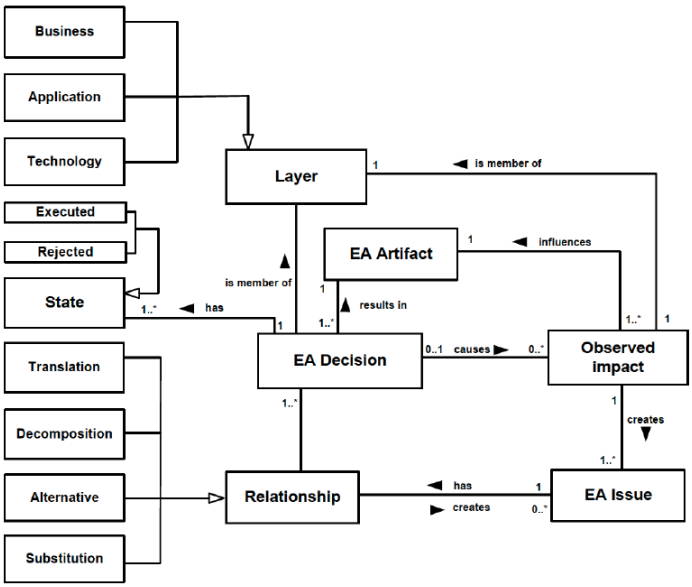

Fig. 1: EA decisions relationship metamodel [5]

Layer is in line with the ArchiMate language [7]: An enterprise is specified in three *layers*: *Business, Application* and *Technology*. Using these layers, an enterprise architect is able to model an enterprise *holistically*, showing not only applications and physical IT infrastructure (which are contained in the application and technology layers), but also how the IT impacts/is impacted by the products, services and business strategy and processes. EA Decisions are related to layers, for instance in the decision graph EA Decision D01 is related to the Business Layer, while EA Decision D06 is related to the Application Layer.

State represents the state of an EA Decision, which is either *Executed* or *Rejected*. In an executed state, an EA decision has already been made and was accepted. A rejected decision, on the other hand, is a decision that was considered as an alternative during the decision making process but was rejected because another decision was more appropriate. In the decision graph, the state of a decision is not explicitly represented but it can be inferred from the relationships. A decision that has an *alternative* relation with an issue is rejected, while all other decisions are executed.

Relationship makes the different types of relationships between EA decisions explicit. Based on ontologies for software architecture design decisions, Plataniotis *et al.* define four relationships. The *Translation* relationship illustrates relationships between decisions and issues that belong to different EA artifacts. During the enterprise transformation process architects translate the requirements that new EA artifacts impose (EA issues) to decisions that will support these requirements by means of another EA artifact. *Decomposition* relationships signify how generic EA decisions decompose into more detailed design decisions within an EA artifact. Alternative relationships illustrate the EA decisions that were rejected (alternatives) in order to address a specific EA issue. *Substitution* relationships illustrates how one EA decision replaces another EA decision. An EA decision can be replaced when it creates a negative observed impact in the enterprise architecture.

Observed Impact signifies an *unanticipated* positive/negative consequence of an already made decision to an EA artifact. This is opposed to anticipated consequences, as indicated by the *Translation* and *Decomposition* relationships. In current everyday practice, architects model anticipated consequences using what-if-scenarios [8]. Unfortunately, not every possible impact of made EA decisions can be predicted. The main usefulness of capturing observed impacts is that they can be used by architects to avoid decisions with negative consequences in future designs of the architecture [5].

For instance, in the decision graph Decision D10 decomposes to decision D11 through issue IS06. D11 turns out to have a negative observed impact OI1, which is translated to a decision D13 through issue IS07 (alternative D12 for IS07 is rejected). D13 addresses the negative observed impact of D11 by substituting D11.

B. Discussion

The metamodel serves as the underlying formalism for the decision graph, but in this subsection we motivate why this is not sufficient by discussing the differences between the metamodel and the decision graph. We will take these remarks into account when formalizing the decision graph in the next section.

According to the decision graph, the creation of a translation/decomposition relationship between two EA Decisions implies the creation of two separate relationships of the same type: one for the EA Decision to EA Issue and another one for the EA Issue to EA Decision. This creates information redundancy issues because this is not captured in the metamodel. The definition of at least one relationship of a specific type should imply that the other relationship should be of he same type. For example, in the decision graph EA Decision 01 is related with EA Issue 03 through a translation relationship. Similarly, EA Issue 03 is related with EA Decision 06 through a translation relationship. The definition of the relationship type between EA Issue 03 and EA Decision 06 should imply the same relationship type between EA Decision 01 and EA Issue 03, but this is currently not captured in the metamodel.

Furthermore, the metamodel provides two different types of states (executed, rejected) per EA Decision. Despite the fact that these two states adequately describe the state of an EA Decision during the *a posteriori* analysis, they don't provide enough expressivity in the *a priori* case. In the latter case, there is the need to express that an EA Decision is in "open" state while enterprise architects examine the alternatives [9].

Whereas the metamodel provides the notion of "Observed impact", it does not explicitly distinguish between "positive observed impact" and "negative observed impact". For instance, in the decision graph EA Decision D11 has Observed Impact OI1, which creates an issue IS07. Thus, it seems that this observed impact is negative, but neither the metamodel or the graph are able to distinguish positive impacts from negative ones.

Finally, there are a number of assumptions on the design graph that have not been made explicit in the metamodel. Firstly, all issues in the graph have been resolved. Secondly, there is always a single decision that is executed in order to solve an issue, while the others are rejected. Finally, a decision that creates a negative observed impact is assumed to be replaced by a decision that addresses this impact. These three assumptions are not formalized, and we propose to do so using integrity constraints.

III. A Formal Model for EA Decision Modeling

In the previous section we showed that the metamodel of Figure 1 is not restrictive enough to characterize the design decision graph of Figure 6 correctly. In order to resolve this issue and to obtain a consistent formalisation for the decision design graphs that allow for *a priori* decision modeling, we will introduce a formal model in this section. We define integrity constraints that are informally reflected in textual descriptions of the previous section and in the decision graph. In the next section we will validate the benefit of our formal approach by applying it to a use case.

A. Elementary Definitions for EA Decision Modeling

Basic concepts from set and graph theory are adequate to define the entities in the metamodel and the relations between them. We begin with representations for the metamodel elements *EA Decision*, *EA Issue*, and *Observed Impact*.

Definition 1 (EA Issue): Let I be a set of EA Issues, where each issue $i \in I$ is a proposition representing the issue.

Rationale and example: An EA decision issue (short: issue) represents a single design concern. For now, we follow Plataniotis *et al.* and we do not add any attributes to the issues, but we recognize that this is certainly possible and a necessary extension. For instance, Zimmerman *et al.*, attribute a total of 18 properties to issues that can be used to characterize them [3]. Because such attributes do not have a specific purpose in our formal model, we leave them out for ease of exposition. The issues in the decision graph are $I = \{IS01, \ldots, IS07\}$.

Definition 2 (EA Decision): Let D be a set of EA Decisions, where each decision $d \in D$ is a tuple (s, a, l) consisting respectively of a state $s \in \{open, executed, rejected\}$, an EA Artifact a, and a layer $l \in \{business, application, technology\}$. We also write s_d, a_d, and l_d to refer to respectively the state, the artifact, and the layer of decision d.

Rationale and example: An EA Decision presents a possible solution to the design issue that is expressed by an EA Issue. The state s represents the current state of the decision. While Plataniotis *et al.* distinguish two different states of a decision (a decision is either "executed" or "rejected"), we extend this with an additional state "open". As we mentioned in the previous section, this is motivated by the fact that we aim to capture *a priori* decision analysis, which is different from the *ex post* approach of Plataniotis *et al.* The EA artifact a of an EA Decision represents the EA artifact to which this decision is related. Finally, the layer l is the layer on which the decision is made. Similar to EA Issues, we leave out additional attributes that do not have a specific purpose in our model. In the decision graph, Decision D06 can be represented with (s, a, l), where $s = executed, a =$ "Customer administration intermediary application service", and $l = application$.

Definition 3 (Observed Impact): Let O be a set of observed impacts, where each observed impact $o = (v, a, l)$ consists of a value v that is either $positive$ or $negative$, i.e. $v \in \{positive, negative\}$, an EA Artifact a, and a Layer l. When $v = positive$ we say that o is a *positive observed impact*; when $o = negative$ we say that o is a *negative observed impact*. We also write v_o, a_o, and l_o to refer respectively to the value, the artifact, and the layer of observed impact o.

Rationale and example: An observed impact is either positive or negative, where negative observed impacts create new issues. This formalization allows for an explicit distinction between positive and negative observed impacts. In the decision graph, the only observed impact is $OI1$, which is negative, so we can formalize this as $OI1 = (v, a, l)$, where $v = negative, a =$

"Customer profile registration Business process", and $l = business$.

Definition 4 (Contains relation): Let $\prec_D \subseteq I \times D$ be a *contains* relation between issues and decisions, $\prec_I \subseteq D \times I$ be a *contains* relation between decisions and issues, $\prec_{O_{in}} \subseteq D \times O$ be a *contains* relation between decisions and observed impact, $\prec_{O_{out}} \subseteq O \times I$ a *contains* relation between observed impact and issues, and $\prec_{DD} \subseteq D \times D$ be a *contains* relation between decisions and decisions. We set the general *contains* relation $\prec = \prec_D \cup \prec_I \cup \prec_{O_{in}} \cup \prec_{O_{out}} \cup \prec_{DD}$. If $(a \prec b)$, then we say that a contains b or that b is contained in a. We sometimes abbreviate $(a \prec b) \wedge (b \prec c)$ with $a \prec b \prec c$.

Rationale and example: The *contains* relation is also used in Zimmerman *et al.* and allows us to define a single hierarchical structure, which serves as a table of content, allowing the user to locate issues and alternatives easily in the enterprise architectural knowledge and helping the knowledge engineer to avoid undesired redundancies. The *contains* relation is the underlying dependency relation that we use to build decision design graphs. We will use this relation to later define the four types of *Relationship* entities that were introduced in the metamodel. These four relationships are relatively complex, so it helps to have a simple underlying representation of the decision hierarchy. Intuitively, the *contains* relations can be obtained by treating all arcs in the decision graph as of the same type. It contains for instance the following relations: $D01 \prec IS01 \prec D02, D01 \prec IS02 \prec D03, D01 \prec IS03 \prec D04$ (see Figure 2), but also $D11 \prec OI1 \prec IS07$ and $IS07 \prec D013 \prec D11$.

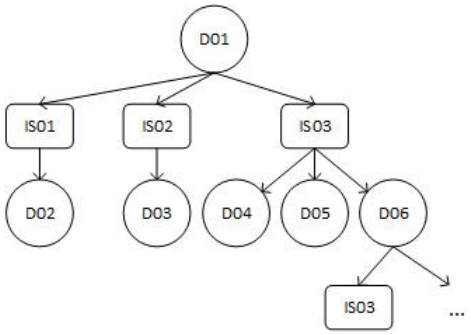

Fig. 2: The *contains* relation $\prec$ for part of the design graph.

Definition 5 (Decision Design Graph (DDG)): A *decision design graph* $\mathbb{D} = (D \cup I \cup O, \prec)$ consists of a set of decisions D, a set of issues I, a set of observed impacts O, and a *contains* relation $\prec$ that induces a graph containing issues, decisions, and observed impacts of decisions.

Rationale and example: Modeling architectural decisions in itself is not new: Ran and Kuusela also propose (but do not formalize) the notation of Design Decision Trees (DDTs) [10]. Zimmermann *et al.* propose a formalization that is comparable to ours, but our is specifically for enterprise architecture decision making and uses decision graphs instead of trees.

B. Layered EA Decision Model and Logical Relations

The metamodel from Section II and the elementary definitions from Section III-A allow knowledge engineers to capture decisions and organize the knowledge in a decision hierarchy. However, the resulting ordered architectural decision tree does not yet support the vision of an active, managed decision model taking a guiding role during architecture design. More relations between decisions, issues and observed impacts must be defined. In this section, we introduce the four relationship of Plataniotis *et al.* and formalize logical constraints by again applying concepts from graph theory.

Definition 6 (Translation relation): The *translation relationship* $R_T \subseteq D \times I \times D$ is a three-placed decision-issue-decision relationship $R_T(d_1, i, d_2)$, also denoted with $d_1 \xrightarrow{T(i)} d_2$, that connects two decisions through an issue where these decisions are related to different EA artifacts. Formally:

$$\forall_{d_1,d_2 \in D, i \in I} : (d_1 \xrightarrow{T(i)} d_2) \Rightarrow (d_1 \prec i \prec d_2) \wedge (a_{d_1} \neq a_{d_2}).$$

Rationale and example: Translation relationships indicate how a decision on one artifact translates to a decision on another artifact through an issue. Thus, having a translation relationship requires three entities: a decision, a issue, and another decision. For instance, the design graph contains the translation relationship $D01 \xrightarrow{T(IS03)} D06$. This is a valid relationship, since we have $D01 \prec IS03 \prec D06$, and we also have $a_{D01} \neq a_{D06}$ because a_{D01} ="Customer profile registration Business process" and a_{D06} ="Customer administration intermediary application service".

Definition 7 (Decomposition relation): The *decomposition relationship* $R_D \subseteq D \times I \times D$ is a three-placed decision-issue-decision relationship $R_D(d_1, i, d_2)$, also denoted with $d_1 \xrightarrow{D(i)} d_2$, that connects two decisions through an issue where these decisions are related to the same EA artifacts. Formally:

$$\forall_{d_1,d_2 \in D, i \in I} : (d_1 \xrightarrow{D(i)} d_2) \Rightarrow (d_1 \prec i \prec d_2) \wedge (a_{d_1} = a_{d_2}).$$

Rationale and example: Decomposition relationships are similar to translation relationships, with the only difference that in decomposition relationships the two artifacts belonging to the decisions in the relation should be the same. For instance, the design graph contains the decomposition relationship $D01 \xrightarrow{T(IS01)} D02$, which is valid because we have $D01 \prec IS01 \prec D02$ and $a_{D01} = a_{D02}$ =""Customer profile registration Business process".

Definition 8 (Substitution relation): The *substitution relationship* $R_S \subseteq D \times D$ is a two-placed decision-decision relationship, also denoted with $d_1 \xrightarrow{S} d_2$, that connects two decisions that are related to the same EA artifacts. Formally:

$$\forall_{d_1,d_2 \in D} : (d_1 \xrightarrow{S} d_2) \Rightarrow (d_1 \prec d_2) \wedge (a_{d_1} = a_{d_2}).$$

Rationale and example: Substitution relationships are simpler than the previous two relationships in the sense that they contain only two elements. The decision graph contains only one substitution relationship $D013 \xrightarrow{S} D11$.

Definition 9 (Alternative relation): The *alternative relationship* $R_A \subseteq I \times D$ is a two-placed issue-decision relationship, also denoted with $i \xrightarrow{A} d$, that connects an issue with a rejected decision. Formally:

$$\forall_{d \in D, i \in I} : (i \xrightarrow{A} d) \Rightarrow ((i \prec d) \wedge (s_d = rejected)).$$

Rationale and example: The alternative relationship indicates decisions that have been rejected in the decision process. For instance, in the design graph we have $IS03 \xrightarrow{A} D04$ and $IS03 \xrightarrow{A} D05$.

Definition 10 (Observed Impact relation): The *observed impact relationship* $R_O \subseteq D \times O \times I \times D$ is a four-placed decision-impact-issue-decision relationship, also denoted with $d_1 \xrightarrow{O(o,i)} d_2$, which describes the effect of a negative observed impact on a decision, which is addressed by an issue and subsequently resolved by a decision. Formally:

$$\forall_{d_1,d_2 \in D, i \in I, o \in O} : (d_1 \xrightarrow{O(o,i)} d_2) \Rightarrow (d_1 \prec o \prec i \prec d_2) \wedge (v_o = negative)$$

Rationale and example: The observed impact relation is the only relation in the design graph that has not been characterized by the metamodel. In the decision graph, EA Decision D11 causes a negative Observed Impact OI1, which is addressed by EA Issue IS07, that is subsequently resolved by EA Decision D13.

With these relations introduced, we will now define three logical constraints on EA decision models. We stress that this list is by no means meant to be exhaustive; It represents a list of constraints that are suggested by the decision graph and from the discussion in Plataniotis *et al.*. These constraints are used to check the decision graph for consistency. If the graph is not consistent, we are able to locate the inconsistency by determining what constraint is violated and for which element. This is useful input for the architect in the decision making process.

Integrity Constraint 1: All issues should be resolved; For each issue, there should be a decision that is contained in this issue and that is executed:

$$\forall_{i \in I} : \exists_{d \in D} : (i \prec d) \wedge (s_d = executed)$$

Rationale and example: An issue represents an architectural design problem that enterprise architects have to address during the enterprise transformation process. Having a consistency check for the status of the issue by verifying whether a decision has been executed to resolve it can assist the architect in detecting "loose ends". This is particularly useful in large and complex graphs with many interdependent nodes [11].

Integrity Constraint 2: If a decision that is contained in an issue is executed, then all other decision that have a relation with that issue should be rejected:

$$\forall_{i \in I} : \exists_{d \in D} : ((i \prec d) \wedge (s_d = executed)) \Rightarrow (\forall_{d' \in D} : (d \neq d') \Rightarrow (s_{d'} = rejected))$$

Rationale and example: This constraint describes a dependency between decisions that are contained in the same issue. The decision graph suggests that issues are solved by a single decision. This means that when a decision is executed that is contained in an issue, all other decision that are contained in this issue should be rejected. For instance, because decision D06 is executed, both decision D04 and D05 are rejected.

Integrity Constraint 3: If a decision contains a negative observed impact, then this decision should be replaced by a decision addressing the negative impact:

$$\forall_{d \in D} : \exists_{o \in O} : ((d \prec o) \wedge (v_o = negative)) \Rightarrow \exists_{d' \in D, i \in I} : ((d \xrightarrow{O(o,i)} d') \wedge (d' \xrightarrow{R} d)).$$

Rationale and example: The goal of having negative observed impacts is to be able to reconsider decisions that have caused this impact. This constraint addresses this idea by stating that negative observed impacts should result in the substitution of the decision that has caused the impact. For instance, decision D11 contains observed impact OI1. This constraint is satisfied for this impact because we have $D11 \xrightarrow{O(OI1,IS07)} D13$ and $D13 \xrightarrow{S} D11$, indicating that decision D13 substitutes decision D11.

IV. Case study: ArchiSurance

In this section we introduce the *ArchiSurance* case study, that we will use to validate our logic-based framework for *a priori* decision analysis. We first introduce the case study, after which we apply it to our framework.

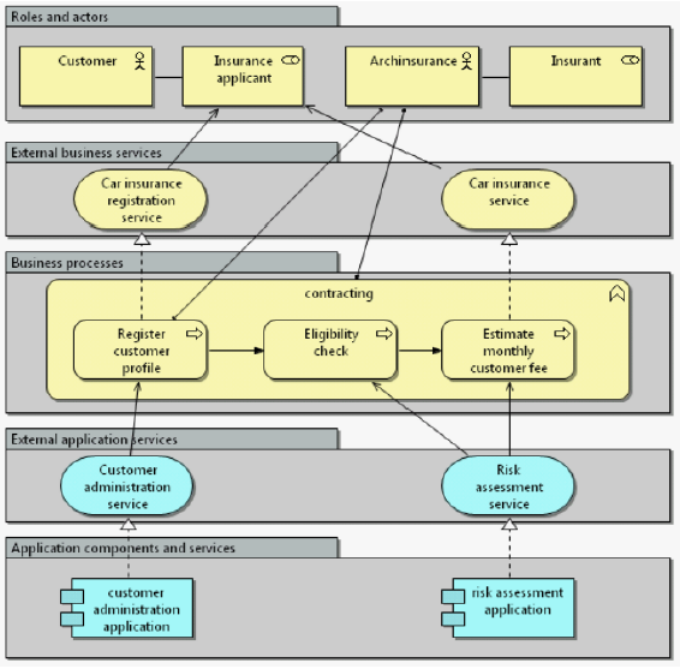

Fig. 3: ArchiSurance direct-to-customer EA model [5]

A. Introduction

This case is inspired by a paper on the economic functions of insurance intermediaries [12], and is the running case used to illustrate the ArchiMate language specifications [13]. *ArchiSurance* is the result of a recent merger of three previously independent insurance companies, that now sells car insurances products using direct-to-customer sales model. The goal of the newly created company is to reduce its operation's and product's costs.

The merger has resulted in a number of integration and alignment challenges for the new company's business processes and information systems. These challenges appear in

the *ArchiSurance* baseline business, application, data and technology architecture.

Figure 3 presents the partial (Business and Application layers) ArchiSurance's direct-to-customer sales model, modeled with the EA modeling language ArchiMate. Two business services support the sales model of ArchiSurance: "Car insurance registration service" and "Car insurance service". ArchiMate helps us to understand the dependencies between different perspectives on an enterprise. For example, in Figure 3 we see that the business service "Car insurance registration service" is realized by a business process "Register customer profile". In turn, we also see that this business process is supported by the application service "Customer administration service".

Although removing intermediates in the supply chain leads to a decrease of operation costs, it also increases the risk of harmful risk profiles [12]. Such profiles lead insurance companies to calculate unsuitable premiums or, even worse, to wrongfully issue insurances to customers. As a response, ArchiSurance decides to use intermediaries to sell its insurance products. After all, compiling accurate risk profiles is part of the core business of an intermediary [12]. In our scenario, an external architect call *John* is hired by ArchiSurance to help guide change to an intermediary sales model. John uses ArchiMate to capture the impacts that selling insurance via an intermediary has in terms of business processes, IT infrastructure and more.

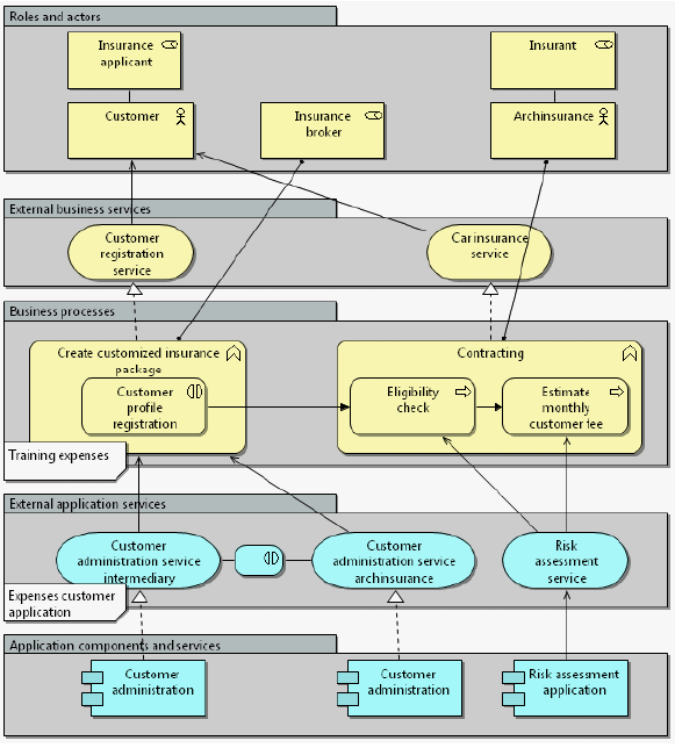

Fig. 4: ArchiSurance intermediary EA model [5]

For illustration purposes we will focus on the translation of the new business process "Customer profile registration" to EA artifacts in the application layer. The resulting ArchiMate model is depicted in Figure 4. Here we see for example how a (new) business process "customer profile registration", owned by the insurance broker (ownership being indicated by a line between the broker and the business process), is supported by the IT applications "customer administration service intermediary" and "customer administration service ArchiSurance".

B. Validation

In this section we demonstrate how the formal framework introduced in Section III supports *a priori* decision analysis of design graphs by consistency checks on the integrity constraints.

Our external architect John is in the process of transforming the ArchiMate model from Figure 3 into Figure 4. For the implementation of these EA artifacts a number of EA decisions have to be made. John, in parallel with ArchiMate modeling language, uses our approach to capture the relationships of decisions and check the consistency of the decision graph.

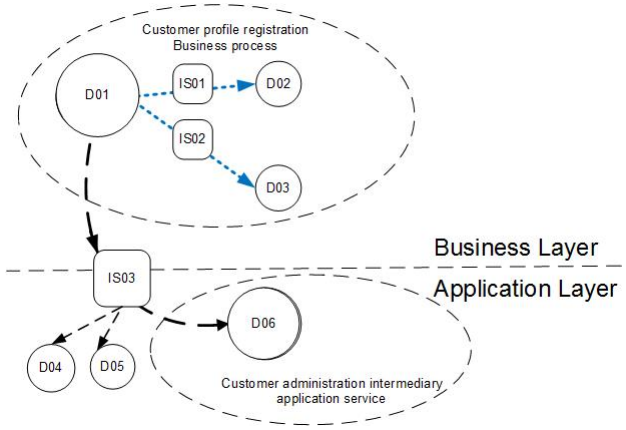

Fig. 5: ArchiSurance scenario: Integrity constraint 1 is violated because EA Issue IS03 is not resolved

John starts by adding the main decision: "Make customer profile registration via intermediary" (D01) to the decision design graph. This decision belongs to the EA artifact "'Customer profile registration Business process". After the enterprise has decided to make this decision, three new issues arise, IS01, IS02, and IS03. Both IS01 and IS02 are addressed by making a decision that related to the same artifact. For IS03, which stands for "Create an appropriate application service to support new business process", there are three different decisions that can be made in the Application Layer, namely D04, D05, and D06 (see Figure 5, the legend of the relations is in Figure 6). At this moment, none of these three decision have been made, so the status of these three decisions is still open. Thus, in Figure 5 there are two *Decomposes* relations, namely $D01 \xrightarrow{D(IS01)} D02$ and $D01 \xrightarrow{D(IS02)} D03$, and the other relations are simply *Contains* relations: $D01 \prec IS03, IS03 \prec D04, IS03 \prec D05, IS03 \prec D06$. After John has created the graph of Figure 5, he checks it for consistency. It turns out that integrity constraint 1 is violated: Not all issues are resolved because for issue IS03 there is no decision d such that $IS03 \prec d$ and $s_d = executed$. John can choose between these three decision and selects decision D06, which stands for "Introduce application service A", as the executed decision.

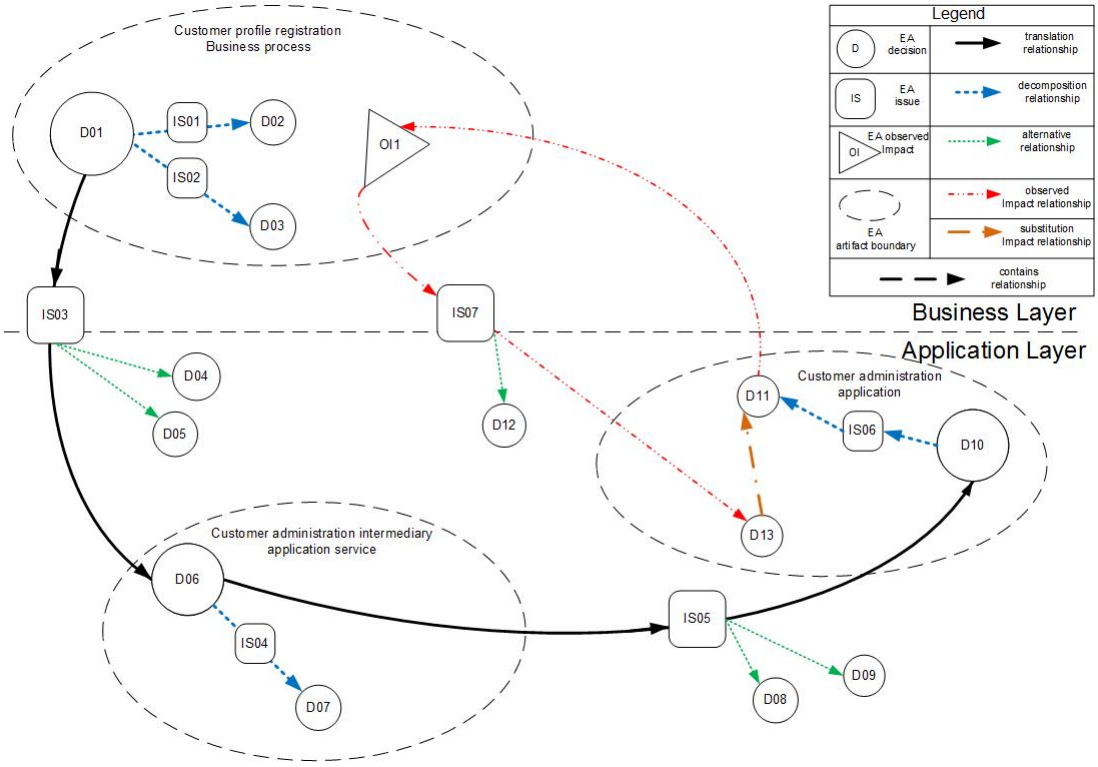

Fig. 6: EA decisions relationships visualization[1]

After having changed the status of decision D06 from "open" to "executed", John checks the consistency of the graph again. This time, another inconsistency arises, namely that integrity constraint 2 is violated. The reason for this is that since decision D06 is contained in issue IS03 (i.e., we have $IS03 \prec D06$) and D06 is executed (i.e. $s_{D06} = executed$), all other decisions that are contained in IS03 (that is, decision D04 and D05) should be rejected. Therefore, John decides to change the status of both these decision from "open" to "rejected". When John checks the graph for consistency now, he finds that the graph is consistent.

Decision D06 results in two new issues, of which "Find an appropriate application to interface with the intermediary" (IS05) is solved by "Acquisition of COTS application B" (D10), resulting in the EA artifact "Customer administration application". Decision D10 decomposes through issue IS06 in the decision "Application interface type 1" (D11).

Using the concept of an Observed impact, John formalizes that users of "Customer administration application" had difficulties using this new application interface. This is signified by the negative observed impact 01 "Degraded user experience in the application use" (OI1). As such, EA decision 11 "Application interface 1" has a negative observed impact on the business process "Customer profile registration".

According to integrity constraint 3, a negative observed impact should be addressed by a decision replacing the original decision that causes the observed impact. Therefore, John translates the observed impact "Degraded user experience in the application use" via EA issue 07 "have fitting application interface" into "replace of existing application interface with an interface similar to the old one" (EA decision 13), after having rejected the alternative decision "Training of users on the new application". The last step John has to take is to replace EA decision 11 "Application interface type 1" with EA decision 13 "Application interface type 2".

When the transformation has finished and all decisions have been made, John obtains the graph that is depicted in Figure 6. This graph is consistent according to the integrity constraints.

V. Related Work

In the domain of software architecture, which is a subset of EA, several design rational approaches have been developed: *argumentation* based approaches such as Issue-Based Information System (IBIS) [14], Design Rationale Language (DRL) [15], *template* based approaches, such as [16] and *model* based approaches, such as [17], [4]. Most of them capture textually the architecture decisions, the rationales, the issues and the implications. In addition, the model based approach provides means to relate those decisions with the software artifacts and with other decisions.

About twenty years ago, Ran *et al.* [10] proposed a systematic approach to document, refine, organize and reuse the architectural knowledge for software design in the form of a Design Decision Tree (DDT) that is a partial ordering on decisions put in the context of the problem requirements and the constraints imposed by earlier decisions. More recently, Tyree and Akerman [16] recognized that architecture decision capturing plays a keys role in what they call "demystifying architecture". They stress that architecture decisions should have a permanent place in the software architecture development

[1]Figure adapted from Plataniotis *et al.* [5]

process. Moreover, it facilitates traceability from decisions back to requirements and it provides agile documentation (which is crucial in an agile process where a team does not have the time to wait for the architect to completely develop and document the architecture).

Both Zimmerman *et al.* [3] and Tan *et al.* [4] recently proposed a comprehensive framework for decision capturing in software architecture. Zimmermann *et al.* also provide a formal framework, focusing mostly on the re-usability of decision by distinguishing between alternatives and outcomes.

In the field of enterprise architecture the literature is significantly more scarce. Even if software architecture is a subset of EA, in this field different types of decisions exist and they can have dependencies and relationship with artifacts and decisions from different layers of the architecture. Plataniotis *et al.* [5] view complements model based approaches for software architecture by providing more specialized attributes for EA decisions as well as more specific dependency and relationship types between EA Decisions.

Finally, goal-oriented modeling frameworks (e.g. i*[2], Tropos[3]) provide means to deal with the motivations of designs , being more expressive then the ArchiMate 2.0. motivation layer. Even so, their main focus is not to provide decision rationales.

VI. Conclusion

In this paper we introduced a logic-based framework for capturing relationships between Enterprise Architecture decisions. This framework is based on recent work by Plataniotis *et al.*. With this formalization, we allow for capturing decision relationship dependencies and consistency checks on additional logical dependencies that we formalized using integrity constraints.

We demonstrated how these constraints can be used to check a decision graph for consistency. However, we did not yet present a framework that will actively search for solutions to inconsistencies and in this way support the architect in its decision making process. To actually do this, a more elaborate representation of decision quality is needed, such that different decision can be compared with each other. We see this are promising future work.

The integrity constraints that we have defined in this work are not meant to be a complete list. As we discussed above, each decision in the metamodel of Plataniotis *et al.* is either *Executed* or *Rejected*. Kruchten *et al.* [9] argue that design decisions evolve in a manner that may be described by a state machine or a statechart. They distinguish between seven different states, which are *idea, tentative, decided, approved, rejected, challenged,* and *obsolete*. Having such an expressive representation of a decision allows for more complex constraints on the decision making process. This is another direction of promising future work.

Finally, one of the biggest challenges in decision capturing is the problem of return of capturing effort. The fact that it takes architects much time to capture design making strategies without having a direct benefit might be a discouraging factor. We believe that our approach simplifies the capturing effort by assisting the architect in its decision making process. Part of our future research will focus on evaluating the actual practical usefulness of our approach.

[2]http://www.cs.toronto.edu/km/istar/

[3]http://www.troposproject.org/

References

[1] M. Op 't Land, H. Proper, M. Waage, J. Cloo, and C. Steghuis, *Enterprise Architecture – Creating Value by Informed Governance*, ser. Enterprise Engineering Series. Springer, Berlin, Germany, 2008.

[2] J. Hoogervorst, "Enterprise architecture: Enabling integration, agility and change," *International Journal of Cooperative Information Systems*, vol. 13, no. 3, pp. 213–233, 2004. [Online]. Available: http://dx.doi.org/10.1142/S021884300400095X

[3] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster, "Managing architectural decision models with dependency relations, integrity constraints, and production rules." *Journal of Systems and Software*, vol. 82, no. 8, pp. 1249–1267, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/jss/jss82.html#ZimmermannKLPS09

[4] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," *J. Syst. Softw.*, vol. 80, no. 6, pp. 918–934, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2006.08.040

[5] G. Plataniotis, S. de Kinderen, and H. A. Proper, "Relating decisions in enterprise architecture using decision design graphs," in *EDOC*, 2013, pp. 139–146.

[6] H. Proper and M. Op 't Land, "Lines in the Water: The Line of Reasoning in an Enterprise Engineering Case Study from the Public Sector," in *Proceedings of the 2nd Working Conference on Practice-driven Research on Enterprise Transformation (PRET), Delft, The Netherlands*, ser. Lecture Notes in Business Information Processing, vol. 69. Springer, 2010, pp. 193–216.

[7] M.-E. Iacob, H. Jonkers, M. Lankhorst, and H. Proper, *ArchiMate 2.0 Specification*. The Open Group, 2012.

[8] M. Lankhorst, *Enterprise architecture at work: modelling, communication, and analysis*. Springer, 2005. [Online]. Available: http://books.google.lu/books?id=ZR_1gXt9vcYC

[9] P. Kruchten, P. Lago, and H. van Vliet, "Building up and reasoning about architectural knowledge," in *Proceedings of the Second International Conference on Quality of Software Architectures*, ser. QoSA'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 43–58. [Online]. Available: http://dx.doi.org/10.1007/11921998_8

[10] A. Ran and J. Kuusela, "Design decision trees," in *Proceedings of the 8th International Workshop on Software Specification and Design*, ser. IWSSD '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 172–. [Online]. Available: http://dl.acm.org/citation.cfm?id=857204.858278

[11] D. N. Kleinmuntz and D. A. Schkade, "Information displays and decision processes," *Psychological Science*, vol. 4, pp. 221 – 227, 1993.

[12] J. Cummins and N. Doherty, "The economics of insurance intermediaries," *The Journal of Risk and Insurance*, vol. 73, no. 3, pp. 359–396, 2006.

[13] H. Jonkers, I. Band, and D. Quartel, "The ArchiSurance Case Study," The Open Group, White Paper, Spring 2012.

[14] W. Kunz, H. W. J. Rittel, W. Messrs, H. Dehlinger, T. Mann, and J. J. Protzen, "Issues as elements of information systems," Tech. Rep., 1970.

[15] J. Lee, "Extending the potts and bruns model for recording design rationale," in *ICSE*, L. Belady, D. R. Barstow, and K. Torii, Eds. IEEE Computer Society / ACM Press, 1991, pp. 114–125.

[16] J. Tyree and A. Akerman, "Architecture decisions: demystifying architecture," *IEEE Software*, vol. 22, pp. 19–27, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=30525&arnumber=1407822&count=16&index=3

[17] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, ser. WICSA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 109–120. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2005.61