SoSYM manuscript No.
(will be inserted by the editor)

Constraint Formalization for Automated Assessment of Enterprise Models

Stef Joosten ·
Ella Roubtsova ·
El Makki Haddouchi

Received: date / Accepted: date

Abstract Enterprises always do their business within some restrictions. In a team of enterprise architects, the restrictions are transformed into the modelling conventions and the corresponding modelling constraints that should be consistently applied across all enterprise models. This paper presents an approach for refining and formalizing modeling conventions into modelling constraints and using them for assessment of enterprise models by a software component called ArchiChecker. The specifics of the proposed approach is that the modeling conventions are first visualized and formalized using the types of elements and relationships of the ArchiMate modeling language, that is also used for modelling of enterprise views. The ArchiMate elements and relationships serve as types to formulate constraints. The elements and relationships in an ArchiMate model are instances of the ArchiMate elements and relationships. Using these types and instances the ArchiChecker automatically generates the lists of violations of modeling conventions in the enterprise models. Each violation shows how a specific enterprise view deviates from a given modeling convention. The paper reports a case study of application of the proposed approach to enterprise modelling views and modelling conventions used in a medical center. The case study shows the added value of formalization and automated assessment of modelling constraints in practice.

Keywords Enterprise Architecture Model · ArchiMate · Modelling Conventions · Modelling Constraints · · Ampersand

Stef Joosten
Open University of the Netherlands
E-mail: Stef.Joosten@ou.nl

Ella Roubtsova
Open University of the Netherlands
E-mail: Ella.Roubtsova@ou.nl

El Makki Haddouchi
University Medical Center Utrecht

1 Introduction

The Open Group enterprise modeling standard language ArchiMate [22] is available to enterprise architects for visualizing and sharing ideas. Enterprise architects create ArchiMate models, which consist of views. Each view in ArchiMate corresponds to a diagram, which is used by an enterprise architect to visualize an enterprise layer, a subsystem or a pattern [17]. A view is a collection of concepts, like, for example, in Figure 4. A concept is either an element or a relationship [22].

ArchiMate gives its user maximal freedom in modeling. Almost anything can be expressed by an enterprise architect in terms of the ArchiMate language. This is nice for an individual who just wants to visualize an idea in his own way. However, the business of an enterprise often defines own restrictions. In a team of enterprise architects, these restrictions are transformed into the corresponding modelling conventions. Collaboration of enterprise architects often requires shared modeling conventions. For a team of enterprise architects, who use ArchiMate, this also initiates the work towards consistency in application of modelling conventions across all the views of their enterprise model.

The visualization freedom makes it hard for enterprise architects to adopt shared modeling conventions. When enterprise architects propose, discuss, and agree upon modelling conventions, ArchiMate provides little assistance in expression of modelling conventions and in assessment of enterprise models on consistency with modelling conventions. One of the ArchiMate motivation elements is called “constraint”, however, this element is a free format field that “represents a factor that limits the realization of goals”[22] and cannot be used for modeling conventions that have to be consistently applied in enterprise models and, sometimes, even enforced.

We propose to add a rule engine to ArchiMate to verify modelling conventions. For the sake of assessment of enterprise models on consistency with modelling conventions, we propose to formalize them as modelling constraints using the element types and relationship types of the enterprise models. This allows us to assess an ArchiMate model with respect to a modelling constraint. Each assessment of an ArchiMate model yields a list of violations of the chosen modelling constraint in terms of instances of elements and relationships of the ArchiMate model.

To illustrate the need and the possibility of formalization and application of modelling constraints for assessment of enterprise models in ArchiMate we have conducted several related studies.

- Section 2 presents the results of a literature study answering the question how the constraints from different areas are expressed in ArchiMate.
- Section 3 discusses the existing attempts of automated assessment of an ArchiMate model with respect to modelling constraints.
- Section 4 presents our tool ArchiChecker designed to analyze an ArchiMate enterprise model and to identify violations of a given modelling constraint.
- Section 5 proposes a practical approach to formalise a modelling convention for assessment of an enterprise model with the ArchiChecker.
- Section 6 reports a case study applying the proposed approach to enterprise modelling views and modelling conventions used in a medical center.

– Section 7 discusses the reasons of difficulties in formalization and automated assessment of modelling constraints on enterprise models.
– Section 8 concludes the paper and draws on future work.

2 Constraints in ArchiMate

To understand how constraints can be imposed on ArchiMate models, let us briefly summarize the ArchiMate language.

The ArchiMate metamodel offers a set of elements and relationships. An element is "used to define and describe the constituent parts of Enterprise Architectures and their unique set of characteristics" [22]. A relationship presents a connection between a source and a target elements. Each element and each relationship has own shape that has been picked by the modeler from a limited set of shapes to reflect the type of the element or relationship.

Throughout this document we use the word "model" in the meaning used by ArchiMate [22], which contains a collection of views (diagrams), as opposed to the more conventional meaning in which a model refers to one diagram only. In each single view, ArchiMate shows a set of boxes presenting instances of elements and lines between those boxes, presenting instances of relationships. So the user perceives every view as a graphical diagram that shows details of an entire enterprise model. An example can be seen in Figure 4.

The model itself has (1) a collection of instances of elements, (2) a collection of instances of relationships and (3) a collection of views in which the instances of elements and relationships are shown. An attractive characteristic of ArchiMate is that different views can share the same instances of elements and relationships. One element instance can even have different shapes in different views, but still be the same instance[1].

Every element of the ArchiMate metamodel is categorized as a strategic, business, application, technologic, motivation, or implementation element.

In order to investigate the exiting ways to specify modelling constraints in ArchiMate, we have analysed research literature and found numerous attempts to present constraints related to access, security, and privacy.

For example, Korman et. al.[16] report the results of the graphical specification of five existing access policies in ArchiMate. "Generic metamodels for expressing configurations of existing models of access control" (page 6) are mapped to ArchiMate. The use of models of access policies is illustrated with a selection of example scenarios and two business cases. The constraints of the specified access policy are graphically modelled, but not subjected to automated verification.

Tepandi et. al.[21] present architectural patterns for the EU Once-Only Principle to ensure that citizens and businesses supply the same information only once to internally re-use this data. The authors propose a reference TOOPRA architecture based

[1] The notion of "the same" is not defined in the ArchiMate reference document [22], but ArchiMate tools typically use an internal key to identify elements. All views are scoped by an ArchiMate tool in the namespace of the model.

on the Once-Only Principle and its scenario-based evaluation. The architecture, constraints, and policies are graphically modelled and not verified.

Security policies have got an ArchiMate metamodel extension proposed by Mayer et.al. [19]. Assurance security cases graphically modeled within the enterprise model by Zhi et.al. [24]. A developed security policy extends the enterprise model as a meta model of architecture and security case, but the models are not used for the verification of enterprise architecture.

Privacy policies have been modeled in ArchiMate by Blanco-Lainé et.al. [6]. The authors have a global look on privacy policies and "addresses the modeling of a given regulation (GDPR) as an EAM fragment that needs to be integrated into a more global EAM"([6], page 14). They have identified several business services related to the GDPR and modeled them in ArchiMate. The authors do not see their ArchiMate models as a means for verification of enterprise models of organizations.

A common denominator in these studies is that ArchiMate modelers use graphical views to specify the access, security, and privacy constraints imposed on enterprise by regulations. It is because, the ArchiMate does not distinguish between a constraint as an informal motivation element and a modelling constraint that should be respected by all enterprise architects in their views. The studies do not go further for formalization and automated verification of constraints, but show the need of formalization of constraints and even their enforcement on enterprise models. This need grows with the growing amount of enterprise views and the modelling conventions imposed on enterprises by regulations.

In this research we focus on modelling constraints representing modelling conventions, violations of which should be computed. This definition allows us to use the term constraint both for its visual presentation and its formula or an algorithm that yields true or false. But it also caters for a condition in natural language, so long as it is mathematically precise. Modelling conventions that may start as informal texts must be reformulated into formulas if enterprise architects want to do automated checks of consistency between an enterprise model and a modeling convention. Formulating a modelling constraint requires the knowledge of the ArchMate elements and the skill to formalize logical expressions, so that is typically done by enterprise architects. Examples of modelling constraints are presented further in this paper.

3 Automated Constraint Verification

Automated constraint verification in models has a long history, especially in the context of systems specification and enterprise architecture [7]. Many tools for verification are available as query systems and analyzers. Marosin et.al. [18] report their experience in the ontological specification of enterprise architecture and use queries for verification of architectural specifications. The semantic web inspires Kharlamov et.al. [14] to represent constraints as OWL2 RL Axioms. The authors also conclude that "the main challenge that we encountered was to capture the constraints of the models using ontological axioms" ([14], page 7). Arriola and Markham [1] propose to use Z-notation to formulate design decisions and control them on the enterprise architecture level. This is related in that Z is a formal specification language (akin to

relation algebra) which is used for architecture specification. But automated verification is not the intention of this work.

In this work we are interested specifically in verification of enterprise architecture models in ArchiMate. Babkin and Ponomarev [2] take an approach based on relation algebra. They have made a metamodel of the ArchiMate language in Alloy [11] and used it to analyse the ArchiSurance [12] model, which is the leading example of ArchiMate and has been published alongside with the ArchiMate reference document. The MIT Alloy Analyzer searches for contradictions in the enterprise architecture models. They conclude that their approach to analysing ArchiMate models, i.e. model verification specified in relation algebra, works.

In order to support attempts of formalisation of constraints and their checks, the tool Archi [3] has recently been enriched with a JavaScript-based scripting plug-in called jArchi [4]. It is built on the Oracle Nashorn engine. With jArchi, an enterprise architect can write JavaScript to encode his own ArchiMate checker.

The common ground in these publications is that identification and formalization of constraints is the major challenge. It is an observation that we share. Once the constraints are identified and formlized, there are various ways to get them verified. But finding constraints and formalizing them turns out to be the real challenge that enterprise architects face.

4 Assessment of Models from an ArchiMate Repository with ArchiChecker

The current paper builds on earlier work with the Ampersand tool [13]. Ampersand uses relational algebra (similar to Alloy [11]) as a language to represent constraints, which allows for verification of enterprise architecture against constraints. Ampersand is used for constraint checking and it also has an established way of constraint (rule) elicitation [23].

The Ampersand compiler has been extended with a parser that reads ArchiMate model repositories [9]. Building on these earlier results, we have used Ampersand to complement ArchiMate with the tool ArchiChecker for automated verification of constraints.

Unlike the approach in Alloy taken by Babkin and Ponomarev [2], we did not make an ArchiMate metamodel. Instead, the ArchiChecker derives the metamodel from the ArchiMate metamodel. This ensures that there can be no discrepancy between the metamodel and the ArchiMate data. This also prevents programming mistakes when matching the data from an ArchiMate model to the metamodel.

Two tools serving the enterprise architect are shown in Figure 1: an ArchiMate modeling tool and the ArchiChecker. For modeling we have used the open source tool Archi (Figure 1), which stores an ArchiMate repository of models in the form of an XML-file with extension ".archimate". In Figure 1 this file is called "repo.archimate". Constraints are stored in the the file called "constraints.adl". The component "ArchiChecker" compiles the ArchiMate repository ("repo.archimate") together with the constraints ("constraints.adl") into a set of violations ("violations.log").

What happens "under the hood" of the ArchiChecker is illustrated in Figure 2. The ArchiMate repository is parsed according to the ArchiMate metamodel. Con-

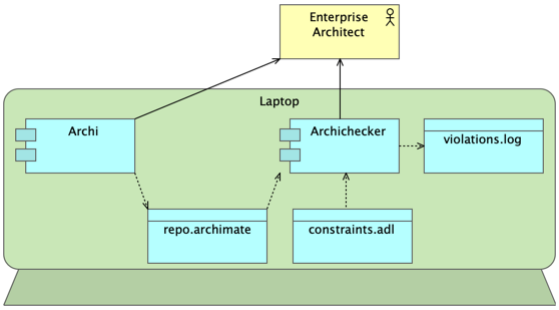

Fig. 1: A toolset for an enterprise architect

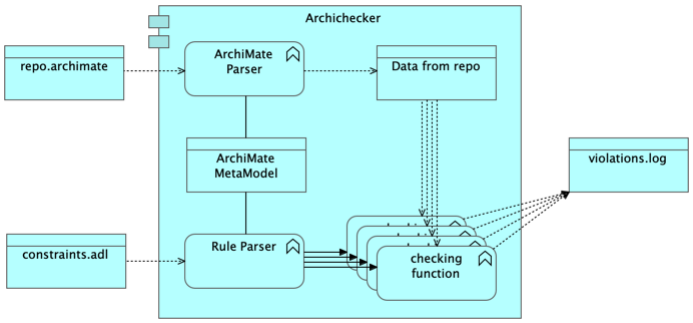

Fig. 2: Under the hood of the ArchiChecker

straints are parsed according to the same metamodel, because a constraint checker must "know" where to find the data to be checked. For each constraint a checking function is generated that can compute violations with respect to that constraint. All violations found by the ArchiChecker are emitted to the "violations.log".

To gain some understanding of the internal functioning, let us look at a small fragment of the ArchiMate metamodel (Listing 1).

Listing 1: A fragment of the ArchiMate metamodel

```
RELATION triggering [BusinessEvent*BusinessProcess]
RELATION triggering [BusinessActor*ApplicationFunction]
RELATION realization [BusinessFunction*BusinessService]
RELATION serving [BusinessFunction*ApplicationFunction]
RELATION type [Relationship*Text] [UNI]
RELATION name [ApplicationFunction*Text] [UNI]
```

The ArchiMate metamodel consists of relations[2], each of which contains pairs. For example, the relation `realization` represents a set of pairs, each of which relates one particular Business Function to one Business Service. An affix (e.g. `[UNI]`) is used to denote a set of multiplicity properties, of which there are four: `UNI` (which

[2] In this paper, the word "relation" refers to the mathematical notion of relation and to a relation in Ampersand, which are the same. The word "relationship" refers to the corresponding notion in ArchiMate.

means univalent), `TOT` (total), `INJ` (injective), and `SUR` (surjective). They can be used in any combination.

Enterprise Models. All data of any enterprise model from the repository are interpreted in terms of such pairs and each pair is contained in the appropriate relations. The fragment in Listing 2 illustrates how data from ArchiMate is allocated to relations.

Listing 2: Enterprise models in an ArchiMate repository

```
POPULATION realization [BusinessFunction*BusinessService] CONTAINS
[("37d64852 3ed1 466e 99d7 22a64c36516d","77c76f36 6b0c 4ab4 8a8b aa2ad81db69f")
,("823b1d78 3424 4ab7 b677 b4d9e4d4af5a","867688b0 b7c1 4807 b3ed 5cb0346ad19c")
,("fa69bf84 a264 4d38 bf96 04a0dfd34644","9b8bad05 1f66 48db 95a5 958ae088d96e")
,("77bb5f5f c55c 4bc4 987f 2fa63554dace","3f39d8e9 dddb 4cb0 97a9 3cbdbbf816ef")]
```

Each relation contains all pairs from all views, so the scope of this relation is the entire repository. The long numbers that constitute pairs are the internal Archi keys for ArchiMate objects.

Although the ArchiMate metamodel is internal, the ArchiChecker exports it in a readable way for the sake of documentation (export not shown in Figure 2).

Constraints. The fragment in Listing 3 shows a constraint, written in Ampersand, which resides in a file with extension .adl.

Listing 3: RULE MC3

```
I[ApplicationComponent]|-
serving ; serving[ApplicationComponent*BusinessProcess] ; serving~
```

Constraints are written in the language Ampersand [13]. An enterprise architect will have to learn this formalism in order to formulate constraints and use the ArchiChecker.

The violations produced by a constraint are pairs, so the set of violations may be interpreted as a relation too. However, for practical use we are more interested in a readable form of the violation. For this purpose the enterprise architect adds a specification to the constraint, so the ArchiChecker can produce readable sentences. Listing 4 gives an example of a violation specification “Application component “name” is not serving a Business Process”.

Listing 4: Violation specification “Application component “name” is not serving a Business Process”

```
VIOLATION (TXT "Application component \'", TGT name, TXT "\'
 is not serving a Business process.")
```

Using this specification ArchiChecker writes to the log the found violations as shown in Listing 5.:

Listing 5: Log of violations

```
There are 12 violations of RULE "MC3":
        Application Component 'Brocacef supplier (orders)'
                                    is not serving a Business Process.
        Application Component 'Central application (Chipsoft HiX)'
                                    is not serving a Business Process.
       ...
```

Summarizing, an enterprise architect makes assessment and gets an information for analysis of an ArchiMate repository by writing the modeling conventions in the form of constraints, followed by running the ArchiChecker on the repository and the constraints.

5 Six step approach to formalise a modelling convention for ArchiChecker

In business documents, one will never find constraints that are ready for automatic checks. Business policies abound, however. Some of policies informally defining modeling conventions. They are presented in natural language and can use different terms for the same things and use the terms deviated from the terms of enterprise models.

In this paper we use the ArchiMate palette of element types and relationship types as an instrument to formulate modeling conventions as constraints from business policies. Each element mentioned in a policy can be classified as one of the element types from the ArchiMate palette. Each relation mentioned in a policy can be classified as one of the relationships from the ArchiMate palette. An incomplete relationship can be refined and completed by adding to it an element type from the ArchiMate palette. In such a way a modelling convention can be visualized in ArchiMate.

Let us elaborate an example. Let there be a policy that says:

"The use of preferred central applications is mandatory."

So we need a constraint that defines which applications are "central". We need formulate it in Ampersand, so the ArchiChecker can name all other applications as violation of this rule.

First, the enterprise architects must decide how to model the preferred central applications. Let us assume they agree to use ArchiMate elements of type *Application Component* to model applications.

Second, a decision should be made what a "central" application means. One suggestion is to put the word "Central" into the name of every preferred central application. The architect rejects this suggestion because this requires the Archichecker to parse names, which requires writing software. The architect also does not want to go through all application components to include the word "Central" in their names.

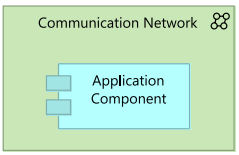

Fig. 3: Visualization of notion "Central Application"

When architects ask themselves what it means to be "central", they come up with a suggestion that an application component is central if it is inside one of the communication networks of the organization. If an application is inside of a communication

network, it is "central". After these decisions, the policy can be reformulated as a modelling constraint and it can be already visualized as shown in Figure 3. Here we have already a constraint expressed in terms of the ArchiMate metamodel:

```
RELATION inside[ApplicationComponent*CommunicationNetwork]
```

Figure 4 shows that application components Autopharma, ServiLocker, MIRA, and Chipsoft HiX that are "central" applications in this definition.

Due to their education, enterprise architects can come up with the constraint in predicate logic, knowing that the relation "inside" is in the ArchiMate metamodel:

$$\forall a \in ApplicationComponent : \exists c \in CommunicationNetwork : \ a \ inside \ c.$$

When they are acquainted with relation algebra (i.c. Ampersand) they formulate the rule with a similar amount of effort:

Listing 6: Constraint in Ampersand

```
RULE "example":
I[ApplicationComponent] |- inside ; I[CommunicationNetwork] ; inside~
VIOLATION (TXT "Application Component \'", SRC name,
TXT "\" in view \"", TGT inView;name, TXT "\" is not inside a LAN")
```

Checking the ArchiMate model (Figure 4 and Figure 5) with this rule produces the following violations:

```
There are 8 violations of RULE "example":
Application Component 'Brocacef supplier (orders)" in view "Pharmacy" is not inside a LAN
Application Component 'Tenant management Office 365" in view "Office 365" is not inside a LAN
Application Component 'App-V/ThinApp" in view "Office 365" is not inside a LAN
Application Component 'Azure AD" in view "Office 365" is not inside a LAN
Application Component 'National Exchange Point " in view "Pharmacy"is not inside a LAN
Application Component 'SCCM" in view "Office 365" is not inside a LAN
Application Component 'Active Directory local" in view "Office 365" is not inside a LAN
Application Component 'EZorg" in view "Pharmacy" is not inside a LAN
```

The application components mentioned violations are in views presented in Figure 4 and Figure 5. When confronted with the violation list, architects must decide what to do. If an application is central, then it has to be included in one of the communication networks in all views. If an application is not central, it is clearly not preferred, so they might want to trade it for a preferred application.

Summarising, we propose a

Six Steps Approach for formalising a modeling convention for ArchiChecker:

1. **Agree on a policy to be assessed by the ArchiChecker.**
2. **Agree on a set of modeling conventions for this policy.** This means to specify how to visually present this policy in terms of ArchiMate element types, properties and relationships types;
3. **Formulate a logical expression of the policy.** This step may be skipped by architects that are familiar with relational algebra and Ampersand, however, a logical expression is useful for correctness control of the visualised convention and the followed Ampersand expressions.
4. **Formulate the rule in Ampersand.** The Ampersand rule should correspond to the logical expression.

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.**
6. **Try to understand the reasons of violations and decide what to do with them.**

We have followed this six steps approach conducting a case study presented in the next section.

6 Results of a case study with the application of the Six Steps Approach

To test our Six Steps Approach, we have conducted a case study in the Utrecht Medical Center (UMC) in the Netherlands. This case study gave us a sense of the difficulties in identification and formalization of modelling conventions in practice.

The Enterprise Architecture of the UMC is partly derived from the nation-wide Hospital Reference Architecture ZiRA [25]. For this case study we have randomly selected two ArchiMate views from a large repository to experiment with. We have found the policies (business restrictions) for enterprise models in documents called Project Start Architecture (PSA) and formalized ten policies as modelling constraints. One ArchiMate view has been taken from the PSA "CS Pharmacy" (Figure 4) and the other from the PSA "Office 365" (Figure 5). Using two views we show which policies and the corresponding modeling constraints are applicable to both views, even though these views are unrelated.

Let us briefly discuss each ArchiMate view.

PSA: CS Pharmacy. In the real PSA CS Pharmacy two information systems are used: MIRA (CGM Pharmacy) and Chipsoft HiX. The systems do not communicate with each other because of the separated Infrastructures. MIRA(CGM Pharmacy) and Chipsoft HiX can be seen in Figure 4. CGM Pharmacy runs completely separate from the UMC Utrecht infrastructure. Management is carried out by the supplier.

The aim of the PSA: "CS Pharmacy" project is to change the enterprise architecture by deploying the application Chipsoft HiX within the UMC Utrecht local area network (UMC Utrecht-LAN) to make available all necessary functionality for the pharmacy.

PSA: Office 365. UMC Utrecht has developed a strategy that combines both MIRA (CGM Pharmacy) and Chipsoft HiX. The combining needs a standardized platform enabling collaboration with colleagues and people within and outside the organization. The chosen standardized platform Office 365 meets the business needs of integral working within UMC Utrecht.

The enterprise architecture PSA: "Office 365" is shown in Figure 5. It containts the support of Personnel administration, Access facilities, Access to ICT, video conferencing, Supply chat, e-mail, address, agenda.

Polices selected for formalisation as modeling conventions. The selection consists of the following policies found in PSA: "CS Pharmacy" and PSA: "Office 365":

1. Only one information system is in use for each functionality.

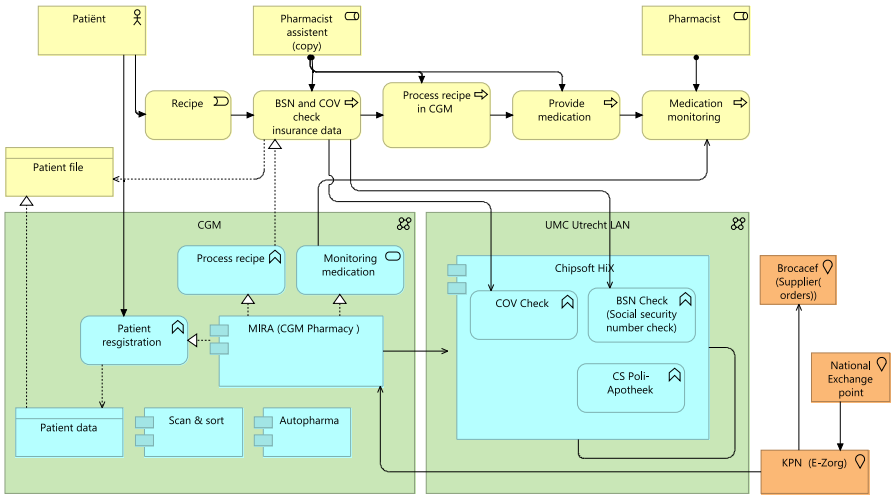

Fig. 4: Enterprise model “CS Pharmacy”.

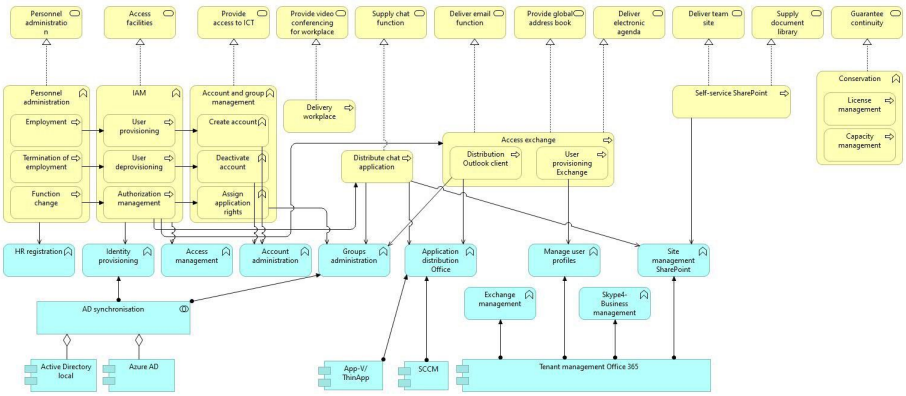

Fig. 5: Enterprise model “Office 365”.

2. Unambiguous and one-time recording of data (and multiple use).
3. Each business process should be realized by at least one application system.
4. A Business process has precisely one owner.
5. Healthcare providers and patients work with one shared file.
6. A data or data group uses one or more business objects.
7. The continuity of critical systems of the Medical Center is guaranteed.
8. The core of information provision is an Enterprise Data Warehouse (EDW).
9. Every data and data type has someone responsible.
10. Use of central applications is mandatory.

The formalization steps and the lists of constraint violation in two presented ArchiMate views are presented per policy.

6.1 Policy 1 - Only one information system is in use for each functionality.

1. **Agree on a policy to be assessed by the ArchiChecker.**
   Every functionality must be realized by precisely one application.
2. **Agree on a set of modeling conventions for this policy.**
   For every application function, there must be precisely one incoming realization relation from an application component (Figure 6).

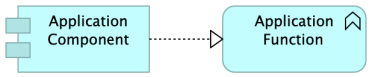

Fig. 6: For every application function, there must be precisely one incoming realization relation from an application component.

3. **Formulate a logical expression of the policy.**
   The phrase "exactly one" gives rise to two distinct expression:
   For every application function, there exists at least one incoming realization relation from an application component.

$$\forall f \in ApplicationFunction\ \exists c \in ApplicationComponent :\ c\ realization\ f.$$

   For every application function, there exists at most one incoming realization relation from an application component.
   $\forall f \in ApplicationFunctio\, \forall\, c_0, c_1 \in ApplicationComponent$:
   c_0 *realization* $f \wedge c_1$ *realization* $f\ \Rightarrow\ c_0 = c_1$.
4. **Formulate the rule in Ampersand.**

```
RULE "MC 1.a":
I[ApplicationFunction] |-
realization[ApplicationComponent*ApplicationFunction]~ ;
realization[ApplicationComponent*ApplicationFunction]
VIOLATION (TXT "ApplicationFunction \'", TGT name
          , TXT "\' is not realized by any ApplicationComponent.")
```

```
RULE "MC 1.b":
realization[ApplicationComponent*ApplicationFunction]~
; -I[ApplicationComponent]
; realization[ApplicationComponent*ApplicationFunction]
|-
-I[ApplicationFunction]
VIOLATION (TXT "ApplicationFunction \'", SRC name, TXT "\' is realized
by ApplicationComponents ", SRC
realization[ApplicationComponent*ApplicationFunction]~;name, TXT ".")
```

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.** There are 10 violations of the Rule "Policy 1.1" in model Office 365 (Figure 5)

```
ApplicationFunction 'HR registration' is not realized by any
    ApplicationComponent.
ApplicationFunction 'Account administration' is not realized by any
    ApplicationComponent.
ApplicationFunction 'Application distribution Office' is not realized by any
    ApplicationComponent.
ApplicationFunction 'Groups administration' is not realized by any
```

```
    ApplicationComponent.
ApplicationFunction 'Access management' is not realized by any
    ApplicationComponent.
ApplicationFunction 'Exchange management' is not realized by any
    ApplicationComponent.
ApplicationFunction 'Manage user profiles' is not realized by any
    ApplicationComponent.
ApplicationFunction 'Site management SharePoint' is not realized by any
    ApplicationComponent.
 ApplicationFunction 'Identity provisioning' is not realized by any
    ApplicationComponent.
```

There is one violation of the Rule "MC 1.a" in view Pharmacy in Figure 4.

```
ApplicationFunction 'CS Poli-Apotheek ' is not realized by any
  ApplicationComponent.
```

There are no violations of RULE "MC 1.b".

6. **Try to understand the reasons of violations and decide what to do with them.**
   The ArchiChecker has identified the ApplicationFunctions that are not realized by any ApplicationComponent. This might indicate that the model is not complete, especially the view Office 365.

6.2 Policy 2 - Unambiguous and one-time recording of data (and multiple use).

To prevent data integrity problems the UMC wants to store a data object just once. This also makes it easier to ensure that users dont have to provide information that is already in the system. Of course multiple copies in multiple nodes are allowed for the sake of backup and high availability, but logically a data object should be present only once. The risk of multiple copies is that one copy gets changed while other copies don't, causing inconsistency of data (data pollution).

1. **Agree on a policy to be assessed by the ArchiChecker.**
   Store once and use manifold, to ensure that users get correct data.
2. **Agree on a set of modeling conventions for this policy.**
   As architects, let us agree that access to a specific data object type is channeled through a single application component, whose responsibility it is to keep that data set in order. Let us model this by a "serving" relationship between the stores and the data management applications.

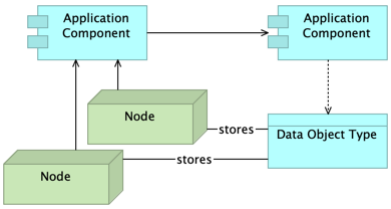

Fig. 7: Store once and use manifold, to ensure that users get correct data.

3. **Formulate a logical expression of the policy.**
   $\forall d \in DataObject,\ \forall n, n_1 \in Node,\ \forall s, s_1 \in ApplicationComponent$:
   $(n\ stores\ d \wedge n\ serving\ s) \wedge (n_1\ stores\ d \wedge n_1\ serving\ s_1) \Rightarrow s = s_1$
4. **Formulate the rule in Ampersand.**

```
RULE "MC 2":
   stores~ ; serving ; -I[ApplicationComponent] ;
   serving[Node*ApplicationComponent]~ ; stores
|- -I[DataObject]
VIOLATION ( TXT "Data objects of type \"", SRC name
          , TXT "\" are stored in "
          , SRC stores~ ; serving[Node*ApplicationComponent])
```

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.**
   This script of ArchiMate yields no violations. Given the completeness of the model this is not very surprising. For architects it is interesting to experiment with the ArchiMate model to make examples to violate this rule deliberately. In this way they can try out their modeling conventions in practice.
6. **Try to understand the reasons of violations and decide what to do with them.**
   In both EA models, the selected policy does not constitute any violation. It is just that situations in which this rule might be violated do not occur yet. It is likely that the enterprise architecture will grow in size and maturity as time goes by. In that case it is conceivable that this rule might be violated. It is up to the architects to decide how useful it is to detect those violations.

6.3 Policy 3 - Each business process should be served by at least one application system.

Similar to application components, processes are organized uniformly. It is preferable to organize the processes as uniformly as possible. Within a project it must be examined to what extent it is possible to design the processes uniformly. There is a challenge with processes outside the organization of UMC Utrecht. All master data must have an unambiguous source system to which other applications are connected.

1. **Agree on a policy to be assessed by the ArchiChecker.**
   Each business process is served by at least one application component.
2. **Agree on a set of modeling conventions for this policy.**
   For every business process, there is an application component that serves the business process by means of a serving relation (Figure 8).

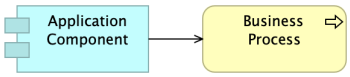

Fig. 8: Each business process is served by at least one application component.

3. **Formulate a logical expression of the policy.**

$$\forall f \in \textit{Business Process}\ \exists c \in \textit{Application Component} :\ c\ \textit{serving}\ f.$$

4. **Formulate the rule in Ampersand.**

```
RULE "MC 3":
I[ApplicationComponent]
|- serving ; serving[ApplicationComponent*BusinessProcess] ; serving~
VIOLATION (TXT "Application component \'", TGT name, TXT "\'
is not serving a Business process.")
```

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.**
   Violations of RULE “Policy 3” in the model CS Pharmacy, Figure 4:

```
Application Component 'Brocacef supplier (orders)' is not serving a
    Business Process.
Application Component 'Central application (Chipsoft HiX)' is not
    serving a Business Process.
Application Component 'Autopharma ' is not serving a Business Process.
Application Component 'MIRA (CGM Pharmacy) ' is not serving a Business
    Process.
Application Component 'ServiLocker' is not serving a Business Process.
Application Component 'National Exchange Point ' is not serving a
    Business Process.
```

   Violations of RULE “Policy 3” in the model Office365, Figure 5:

```
Application Component 'Azure AD' is not serving a Business Process.
Application Component 'Tenant management Office 365' is not serving a
    Business Process.
Application Component 'App-V/ThinApp' is not serving a Business
    Process.
Application Component 'SCCM' is not serving a Business Process.
```

6. **Try to understand the reasons of violations and decide what to do with them.** In this case all the applications do not serve any business process. This means, that the architects have not chosen to visualize the rule in both models.

6.4 Policy 4 - Every business process has precisely one owner.

1. **Agree on a policy to be assessed by the ArchiChecker.**
   Every business process has precisely one owner.
2. **Agree on a set of modeling conventions for this policy.**
   Every business actor relates to precisely one business process by means of an association relation called "owner"(Figure 9).

Fig. 9: Every business actor relates to precisely one business process by means of an association relation called "owner".

3. **Formulate a logical expression of the policy.**
   MC 4.1.
   $\forall b \in BusinessProcess\ \exists a \in BusinessActor :\ a\ owner\ b.$
   MC 4.2.
   $\forall s \in BusinessProcess\ \forall a,b \in BusinessActor : a\ owner\ s\ \wedge\ b\ owner\ s \Rightarrow a = b.$
4. **Formulate the rule in Ampersand.**

```
RULE "MC 4.1":
I [BusinessProcess] |- owner[BusinessActor*BusinessProcess]~ ;
owner[BusinessActor*BusinessProcess]
VIOLATION (TXT "Business Process \'", SRC name, TXT "\' does not have an
owner.")
```

```
RULE "MC 4.2":
owner[BusinessActor*BusinessProcess] |- -(-I[BusinessActor] ; owner)
VIOLATION (TXT "Business Process \'", TGT name, TXT "\' has multiple
\item \textbf{Violations generated by the ArchiChecker.}
```

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.**
   The violations of RULE "MC 4.1" in enterprise model Office365, Figure 5:

```
Business Process 'User deprovisioning' does not have an owner.
Business Process 'Self-service SharePoint' does not have an owner.
Business Process 'Access exchange' does not have an owner.
Business Process 'License management' does not have an owner.
Business Process 'Distribution Outlook client' does not have an owner.
Business Process 'Termination of employment' does not have an owner.
Business Process 'User provisioning' does not have an owner.
Business Process 'User provisioning Exchange' does not have an owner.
Business Process 'Capacity management' does not have an owner.
...
(8 more)
```

The violations of RULE “MC 4.1“ in enterprise model CS Pharmacy, Figure 4:

```
Business Process 'BSN and COV check insurance data' does not have an
owner.
```

There are no violation of MC 4.2.

6. **Try to understand the reasons of violations and decide what to do with them.**
   The architect have chosen not to visualize owners. It is assumed that an owner has been assigned for each business process. In the case of CS Pharmacy (Figure 4) a role / owner has been assigned for a crucial process, namely “medication monitoring’

6.5 Policy 5 - Healthcare providers and patients work with one shared file.

All healthcare providers involved in the healthcare of a patient and the patients themselves work with one shared file. They have access to the healthcare file and work with the same information. Clear and findable information makes an important contribution to quality and patient safety. This contains the (core) data regarding the health status and treatment of the patient that are important for all healthcare professionals who have a treatment relationship with the patient. The implementation of CS Pharmacy creates a more unambiguous working environment and one integrated medication file

1. **Agree on a policy to be assessed by the ArchiChecker.**
   For every patient, there must be one file called “Patients File”. Each patient must have access to his or her patient file. Each health worker directly involved in medical care for a patient must have access to that patient’s file. Others have no access to this file.

2. **Agree on a set of modeling conventions for this policy.**
   The rule “A patients file is accessible only to the patient himself and all health workers directly involved in medical care for that patient.” cannot be modeled in ArchiMate in a direct fashion. Therefore it is entered textually in a constraint element. For every business actor named “Patient”, we make sure to model a data object named “Patient’s File”. The patient and health workers have an access relationship (read/write) to the patient’s file (Figure 10).

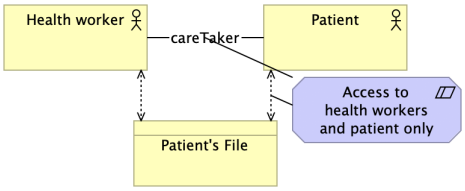

Fig. 10: For every patient, there must be one file called “Patients File”. Each patient must have access to his or her patient file. Each health worker directly involved in medical care for a patient must have access to that patient’s file. Others have no access to this file.

3. **Formulate a logical expression of the policy.**
   Let patient be a predicate on BusinessActor . Let patientfile be a predicate on BusinessObject . Let fileof be a function $BusinessActor \rightarrow BusinessObject$. Let caretaker be a relation $BusinessActor \times BusinessActor$ that relates health workers and their own patients.
   $\forall a \in BusinessActor\ \exists o \in BusinessObject :\ patient(a), fileof(a) \Rightarrow$
   $patientfile(o)\ \wedge a\ access\ o.$
   $\forall a,b \in BusinessActor\ \forall o \in BusinessObject : a\ caretaker\ b \Rightarrow\ b\ oaccess\ o\ \wedge a\ access\ o.$
4. **Formulate the rule in Ampersand.**

```
CLASSIFY PatientFile ISA BusinessObject
CLASSIFY Patient ISA BusinessActor
CLASSIFY Patient ISA Person

I[Patient] = name;"Patient";name~
I[PatientFile] = name;"Patient s File";name~
[Patient] |- access[Patient*PatientFile] ; access[Patient*PatientFile]~
careTaker;access = access[Patient*PatientFile]~

RULE "MC 5":I [BusinessObject] |- access
[BusinessObject*BusinessActor]; access [BusinessObject*BusinessActor]~
MEANING "If (a,b) is in the relation caretaker, then person a is a health
worker directly involved in medical care for patient b."
VIOLATION (TXT "PatientFile (Business Object) \'", SRC
```

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.**
   The violation of RULE "MC 5" in enterprise model CS Pharmacy, Figure 4:

   ```
   PatientFile (Business Object) 'Patient file ' is not accessed by
   a caretaker/Patient (Health worker).
   ```

6. **Try to understand the reasons of violations and decide what to do with them.**
   The visualization of the fact that the caretaker has access to the patient's file has been forgotten indeed.

6.6 Policy 6 - A data or data group uses one or more business objects.

Data is always recorded, presented and exchanged in their "context". The place where the user is located (clinic or emergency room), the situation, the time, the device used and the user account. The "circumstances" under which data was obtained and recorded. A date and time is set for healthcare logistics data, so that control information about the progress of healthcare logistics processes can be made available. To ensure the relevance of data, every data object in an ArchiMate model must be linked to a business object. The purpose is to spot redundant data or incomplete architecture models.

1. **Agree on a policy to be assessed by the ArchiChecker.**
   Every Data Object realizes one or more Business Objects.

2. **Agree on a set of modeling conventions for this policy.**

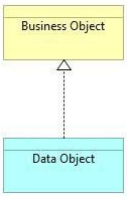

Fig. 11: Every Data Object realizes one or more Business Objects.

3. **Formulate a logical expression of the policy.**
   $\forall b \in DataObject\ \exists o \in BusinessObject : c\ realization\ b.$
4. **Formulate the rule in Ampersand.**

```
RULE "Policy 6":
I[DataObject] |- realization[DataObject*BusinessObject];
realization[DataObject*BusinessObject]~
VIOLATION (TXT "Data Object \'", SRC name, TXT "\'
does not realize any Business Object")
```

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.**
   This script of ArchiMate containts no violations.
6. **Try to understand the reasons of violations and decide what to do with them.**
   There is just one data object, 'Patient data', which resides in model CS Pharmacy Figure 4. Since it is connected to Business object Patient file by a realization relation, there are no violations of this policy.

6.7 Policy 7 - The continuity of critical systems is guaranteed.

The availability of the data must be and remain guaranteed. The data must also remain available for updates or a switch to another system. The data must therefore also be understandable without the Office365 logic.

1. **Agree on a policy to be assessed by the ArchiChecker.**
   Each IT system has an owner responsible for ensuring continuity and taking adequate measures.

2. **Agree on a set of modeling conventions for this policy.**
   The architects can choose to model an owner for every application component, by making an association relationship called "owner" from every application component to its owner (a Business Actor). We let the checker test whether the owner is unique, because we dont want multiple owners for one application component (Figure 12).

Fig. 12: Each IT system has an (unique) owner responsible for ensuring continuity and taking adequate measures.

3. **Formulate a logical expression of the policy.**
   $\forall a \in ApplicationComponent\ \exists b \in BusinessActor:\ b\ owner\ a.$
   $\forall b,d \in BusinessActor, \forall a \in ApplicationComponent:\ b\ owner\ a \wedge d\ owner\ a \Rightarrow b = d.$
4. **Formulate the rule in Ampersand.**

```
RULE "MC 7.1":
  I [ApplicationComponent] |-
  owner[BusinessActor*ApplicationComponent]~ ;
  owner[BusinessActor*ApplicationComponent]
VIOLATION (TXT "Application Component \'", SRC name,
TXT "\' does not have an owner.")

RULE "MC 7.2":
  owner[BusinessActor*ApplicationComponent] |-
  -(-I[BusinessActor] ; owner)
VIOLATION (TXT "Application Component \'", TGT name,
TXT "\' has ", SRC name, TXT " owners.")
```

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.**
   Violations of RULE "MC 7.1" in model CS Pharmacy, Figure 4:

```
Application Component 'Brocacef supplier (orders)' does not have an owner.
Application Component 'Central application (Chipsoft HiX)' does not have
an owner.
Application Component 'Autopharma ' does not have an owner.
Application Component 'MIRA (CGM Pharmacy)' does not have an owner.
Application Component 'ServiLocker' does not have an owner.
```

   Violations of RULE "MC 7.1" in model Office365, Figure 5:

```
Application Component 'Tenant management Office 365' does not have an owner.
Application Component 'App-V/ThinApp' does not have an owner.
Application Component 'Azure AD' does not have an owner.
Application Component 'National Exchange Point' does not have an owner.
Application Component 'SCCM' does not have an owner.
```

There are no violations of MC 7.2.

6. **Try to understand the reasons of violations and decide what to do with them.** The enterprise models CS Pharmacy (Figure 4) and Office365 (Figure 5) show 13 application components, none of which has an owner. So whatever the ArchiMate practice and whatever the formalization, there will always be 13 violations of this policy as long as owners are not being modeled.

6.8 Policy 8 - The core of information provision is an Enterprise Data Warehouse (EDW).

1. **Agree on a policy to be assessed by the ArchiChecker.**
   Enable integrated data and information provision on various domains.

2. **Agree on a set of modeling conventions for this policy.** Each application component serves the Enterprise Data Warehouse (EDW) (Figure 13).

Fig. 13: Each application component serves the Enterprise Data Warehouse (EDW).

3. **Formulate a logical expression of the policy.**

$$\forall b \in ApplicationComponent\ \exists c \in Node : \ c\ serving\ b.$$

4. **Formulate the rule in Ampersand.**

```
RULE "MC 8":
I[ApplicationComponent] |- serving [ApplicationComponent*Node];
serving [ApplicationComponent*Node]~
VIOLATION (TXT "Application MIRA\'", SRC name,
TXT "\'has no access with the Data Warehouse")
```

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.**
   Violations of RULE "MC 8" in model CS Pharmacy, Figure 4:

```
Application 'MIRA (CGM Pharmacy)' has no access with the Data Warehouse.
Application 'ServiLocker' has no access with the Data Warehouse.
Application 'Azure AD' has no access with the Data Warehouse.
Application 'National Exchange Point' has no access with the Data Warehouse.
```

   Violations of RULE "Policy 8" in model Office365, Figure 5:

```
Application 'Brocacef supplier (orders)'has no access with
the Data Warehouse.
Application' Central application (Chipsoft HiX)' has no access with
the Data Warehouse.
Application 'Autopharma' has no access with the Data Warehouse.
Application 'Tenant management Office 365' has no access with
the Data Warehouse.
Application 'App-V/ThinApp'has no access with the Data Warehouse.
Application 'SCCM' has no access with the Data Warehouse
```

6. **Try to understand the reasons of violations and decide what to do with them.** In both models, it was decided not to visualize the Data Warehouse (EDW). As a result, we cannot assess whether an application has access to the EDW.

6.9 Constraint-policy 9 - Every data and data type has someone responsible.

1. **Agree on a policy to be assessed by the ArchiChecker.**
   Every data object must have precisely one owner (Figure 14).

2. **Agree on a set of modeling conventions for this policy.**

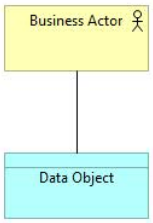

Fig. 14: Every data object must have precisely one owner.

3. **Formulate a logical expression of the policy.**

$$\forall b \in DataObject\ \exists c \in BusinessActor : \ c\ association\ b.$$

4. **Formulate the rule in Ampersand.**

```
RULE "MC 9":
I [DataObject] |- association [DataObject*BusinessRole];
                  association [DataObject*BusinessRole]~
VIOLATION (TXT "DataObject\'", SRC name, TXT "\'a data object
has no responsible")
```

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.**
   There is one violations of RULE “MC 9” in model CS Pharmacy, Figure 4:
   `DataObject' Patient data' a data object has no responsible.`
6. **Try to understand the reasons of violations and decide what to do with them.**
   The visualized data object in model CS Pharmacy, Figure 4 has no any responsible. Possibly, it is not the focus of the view.

6.10 Policy 10 - Use of central application is mandatory.

This policy has been discussed in section 5.

1. **Agree on a policy to be assessed by the ArchiChecker.**
   An application component is central if it is inside one of the communication networks of the organization.
2. **Agree on a set of modeling conventions for this policy.**
3. **Formulate a logical expression of the policy.**

$$\forall a \in ApplicationComponent : \exists c \in CommunicationNetwork : \ a\ inside\ c.$$

4. **Formulate the rule in Ampersand corresponding to the logical expression;**

Listing 7: Constraint in Ampersand

```
RULE "example":
I[ApplicationComponent] |- inside ; I[CommunicationNetwork] ; inside~
VIOLATION (TXT "Application Component \'", SRC name,
TXT "\" in view \"", TGT inView;name, TXT "\" is not inside a LAN")
```

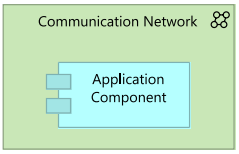

Fig. 15: An application component is central if it is inside one of the communication networks of the organization.

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations;**

```
There are 8 violations:
Application Component 'Brocacef supplier (orders)" in view "Pharmacy"
is not inside a LAN
Application Component 'Tenant management Office 365" in view "Office 365"
is not inside a LAN
Application Component 'App-V/ThinApp" in view "Office 365" is not inside a LAN
Application Component 'Azure AD" in view "Office 365" is not inside a LAN
Application Component 'National Exchange Point " in view "Pharmacy"is not inside a LAN
Application Component 'SCCM" in view "Office 365" is not inside a LAN
Application Component 'Active Directory local" in view "Office 365" is not inside a LAN
Application Component 'EZorg" in view "Pharmacy" is not inside a LAN
```

6. **Try to understand the reasons of violations and decide what to do with them.**
   The application components mentioned violations are in views presented on figures 4 and 5. When confronted with the violation list, architects must decide what to do. If an application is central, then it has to be included in one of the communication networks in all views. If an application is not central, it is clearly not preferred, so they might want to trade it for a preferred application.

7 Reasons of difficulties of formalization and automated assessment of modelling constraints on enterprise models

The results of using of our Six Steps Approach for formalising modeling conventions for the ArchiChecker of ArchiMate views clarify the reasons of the difficulties in formalization and automated checks.

The ArchiMate metamodel is an intermediate step helping on the way between informal policy and a verifiable rule. The steps of our Six Steps Approach direct enterprise architects on the way from an informal policy to agreement on a modelling conventions to a rule formalization. Let us present our observations step by step.

1. **Agree on a policy to be assessed by the ArchiChecker.**
   It is difficult to agree on informal policy. It is because an informal policy can be differently understood. We have observed that some policies in Project Start Architectures (PSA) do not contain sufficient information for precision and formalization. A precise reformulation helps to achieve agreement on a policy. For example, the initial “Policy 5 - Healthcare providers and patients work with one shared file” has been refined to a set of statements: (1)“For every patient, there must be one file called Patient’s file.” (2)”Each patient must have access to his or her Patient’s file.” (3)“Each health worker directly involved in medical care for a patient must have access to that Patient’s file.” (4)“Others have no access to this Patient’s file.” Only after this refinement and reformulation in natural language a

modelling convention can be formulated in terms enterprise models.

2. **Agree on a set of modeling conventions for this policy.**
   The polices are expressed in business terms, such as “Autopharma”, and “Chipsoft HiX”. In order to formalize them, the enterprise architect has to mentally categorise each of the concepts of a policy to element types and relation types of the ArchiMate language, i.e. using phrases such as “application component” and “realization”, rather than phrases from the business. The visualization of a policy using ArchiMate objects and relations makes it easier to categorise the concepts and relations of a policy.

3. **Formulate a logical expression of the policy.**
   Predicate Logic is used as a universal tool for precise understanding of the meaning of the visualised modelling convention. A logical expression is further used to control the rule in relational algebra written for ArchiChecker.

4. **Formulate the rule in Ampersand.**
   The ArchiChecker rules are needed as an input of the ArchiChecker. Any expression in Predicate Logic can be always rewritten in the Ampersand notation understood by the ArchiChecker.

5. **Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.**
   The ArchiChecker produces violations that mention instances of ArchiMate elements and relationships types in the views, which are known by the business (In our case by the University Medical Center).

6. **Try to understand the reasons of violations and decide what to do with them.**
   The list of violations, the visual presentation of the policy and it’s meaning in Predicate Logic can be used for understanding the reasons of violations.
   The ArchiChecker identifies relatively large amounts of violations of organizational policies. These violations do not necessarily mean that something is wrong with the architecture. An enterprise architect may use a constraint just to notify peculiar situations or situations that deserve a little extra attention. Our experience shows that that both the modelling constraints (the input of the ArchiChecker) and the list of violations (the output of the ArchiChecker) cannot be directly shared verbatim with the business. In both directions the interpretation of the enterprise architect is needed to qualify the implications for other stakeholders. If violations turn out to have unexpected results, it can mean many things. One option is that the policies have not been precisely formulated and have not been understood. Another possibility is that the enterprise architect has erred when formalizing the policy. Hence it is wise to inspect all violations by hand and try to find out in the model why this violation occurs. The understanding and interpretation of policies by different professionals often demands communication between policymakers and enterprise architects.

We have learned that modeling conventions are proper to a specific group of enterprise architects, so they need the skills to formalize their modeling conventions (merely multiplicity constraints) if they want them checked. However, we have observed that many modeling conventions can be reused because they present the wisdom of enterprise architects and some principles of a good enterprise architecture.

We have experienced some support of our research and interest both from management and enterprise architects, motivated as follows:

- Conformance to modelling conventions can be easily reported and valued by management.
- Violation of modelling conventions triggers the improvement of enterprise architecture or refinement of the modelling conventions.

8 Conclusion and Future Work

Many authors have noted that the acquisition of constraints is the real challenge the enterprise architects face [2,8,20,5].

In this paper we have proposed a Six Steps Approach for acquisition of modeling conventions and formalising them as modelling constraints for the ArchiChecker of ArchiMate views. The approach is novel in the sense that it uses the ArchiMate metamodel element types and relationships to prepare both the modelling conventions and the enterprise models for constraint checks.

Our approach provides an evidence that a mathematical formalism such as predicate logic or relation algebra is to be preferred over a programming notation to formulate constraints. A mathematical formalism is declarative by nature, making it easier to check whether it captures a modeling convention accurately. A programming formalism (such as jArchi, which uses JavaScript) will work, but it requires a programming effort for every single modeling convention and it comes with a larger risk of making mistakes.

The practical advantage of the visualization and formalization of modelling conventions in the ArchiMate Enterprise modeling language and their automatic conformance checks open a possibility to assess and report conformance to or violation of policies in the enterprise or even enforce them when necessary. It helps in understanding of the strong and weak points of enterprise architecture, which is important to make decisions and plan actions.

The theoretical perspective of the formalization of modelling conventions in the ArchiMate Enterprise modeling language is the possibility to specify patterns and standards. The formalization of patters from ISO standards and the compliance assessment of Enterprise models to ISO standards and reference architecture models have been addressed by by Kim.H.M and Fox M.S [15], but their formalization has used an ontology for presentation ISO 9000, i.e they had not used ArchiMate. Patterns are needed to use both the top-down and bottom up enterprise modelling [10], but they should be presented in ArchiMate. A modelling constraint can be used as a pattern for Enterprise modeling. The structural couplings of assets studied by Bider [5] may be the candidates for more complex modelling constraints. We may see the resemblance of these models with Policy 7 expressing an owner related to a business process and

Policy 9 expressing an owner to a data type. However, the generalisation of patterns is the future work.

In future work we also see different usage of the formalization of modelling constraints in ArchiMate. The modelling constraints can be derived from both from policies in organisations and compared with the constraints formalised in this paper. The modelling constraints can be derived from a law or a regulation. We plan to use our Six Steps Approach in the education process for students studying Enterprise Managements and Information Technology. This may be seen as valorization of the scientific knowledge and contribution to the culture of enterprise modelling in practice.

Acknowledgements The authors thank the enterprise architects of the University Medical Center Utrecht for their cooperation during conducting the case study.

References

1. Arriola, L., Markham, A.: Towards an enterprise architecture controlling framework. In: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, pp. 1–7 (2018)
2. Babkin, E.A., Ponomarev, N.O.: Analysis of the consistency of enterprise architecture models using formal verification methods. Business Informatics (3), 30–40 (2017)
3. Beauvoir, P., Sarrodie, J.: Archi-the free archimate modelling tool. User Guide. The Open Group (2018)
4. Beauvoir, P., Sarrodie, J.B.: Archi-Open Source Archimate Modelling (2019)
5. Bider, I.: Structural Coupling, Strategy and Fractal Enterprise Modeling. In: International Conference on Research Challenges in Information Science, pp. 95–111. Springer (2020)
6. Blanco-Lainé, G., Sottet, J.S., Dupuy-Chessa, S.: Using an enterprise architecture model for GDPR compliance principles. In: IFIP Working Conference on The Practice of Enterprise Modeling, pp. 199–214. Springer (2019)
7. Chapurlat, V., Braesch, C.: Verification, validation, qualification and certification of enterprise models: Statements and opportunities. Computers in Industry **59**(7), 711–721 (2008)
8. Chatzikonstantinou, G., Kontogiannis, K.: Policy modeling and compliance verification in enterprise software systems: A survey. In: 2012 IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), pp. 27–36. IEEE (2012)
9. Filet, P., van de Wetering, R., Joosten, S.: Enterprise architecture alignment. researchgate.net. Department of Information Sciences, Open University of the Netherland. (2019)
10. Fossland, S., Krogstie, J.: Combining top-down and bottom-up enterprise modelling. CEUR Workshop Proceedings (2017)
11. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press (2006)
12. Jonkers, H., Band, I., Quartel, D., Lankhorst, M.: ArchiSurance Case Study version 2. The Open Group (2016)
13. Joosten, S.: Relation Algebra as programming language using the Ampersand compiler. Journal of Logical and Algebraic Methods in Programming **100**, 113–129 (2018)
14. Kharlamov, E., Grau, B.C., Jiménez-Ruiz, E., Lamparter, S., Mehdi, G., Ringsquandl, M., Nenov, Y., Grimm, S., Roshchin, M., Horrocks, I.: Capturing industrial information models with ontologies and constraints. In: International Semantic Web Conference, pp. 325–343. Springer (2016)
15. Kim, H.M., Fox, M.S.: Using enterprise reference models for automated ISO 9000 compliance evaluation. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences, pp. 1278–1287. IEEE (2002)
16. Korman, M., Lagerström, R., Ekstedt, M.: Modeling enterprise authorization: a unified metamodel and initial validation. Complex Systems Informatics and Modeling Quarterly (7), 1–24 (2016)
17. Lankhorst, M.M., Proper, H.A., Jonkers, H.: The anatomy of the ArchiMate language. International Journal of Information System Modeling and Design (IJISMD) **1**(1), 1–32 (2010)
18. Marosin, D., Ghanavati, S., Van Der Linden, D.: A Principle-based Goal-oriented Requirements Language (GRL) for Enterprise Architecture. In: iStar (2014)

19. Mayer, N., Aubert, J., Grandry, E., Feltus, C., Goettelmann, E., Wieringa, R.: An integrated conceptual model for information system security risk management supported by enterprise architecture management. Software & Systems Modeling **18**(3), 2285–2312 (2019)
20. Ramos, A., Gomez, P., Sánchez, M., Villalobos, J.: Automated enterprise-level analysis of Archimate models. In: Enterprise, Business-Process and Information Systems Modeling, pp. 439–453. Springer (2014)
21. Tepandi, J., Grandry, E., Fieten, S., Rotuna, C., Sellitto, G.P., Zeginis, D., Draheim, D., Piho, G., Tambouris, E., Tarabanis, K.: Towards a cross-border reference architecture for the once-only principle in europe: An enterprise modelling approach. In: IFIP Working Conference on The Practice of Enterprise Modeling, pp. 103–117. Springer (2019)
22. The Open Group: ArchiMate 3.1 Specification. (2012-2019)
23. Wedemeijer, L.: A relation-algebra language to specify declarative business rules. In: 4th International Symposium on Business Modeling and System Design, BMSD, pp. 63–73 (2014)
24. Zhi, Q., Yamamoto, S., Morisaki, S.: IMSA-Intra Model Security Assurance. J. Internet Serv. Inf. Secur. **8**(2), 18–32 (2018)
25. ZIRA: Ziekenhuis Referentie Architectuur) https://sites.google.com/site/zirawiki/ (2019)