



# Constraint Formalization for Automated Assessment of Enterprise Models

Stef Joosten<sup>1</sup> <sup>a</sup>, Ella Roubtsova<sup>1</sup> <sup>b</sup> and El Makki Haddouchi<sup>2</sup>

<sup>1</sup>Open University of the Netherlands, Department of Information Science, Heerlen, The Netherlands

<sup>2</sup>University Medical Center Utrecht, The Netherlands

**Keywords:** Enterprise Architecture Model, ArchiMate, Modelling Conventions, Modelling Constraints, Ampersand-tool.

**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 is used to discuss the added value of formalization and automated assessment of modelling constraints in enterprise modelling.


## 1 INTRODUCTION


The Open Group enterprise modeling standard language ArchiMate (The Open Group, 2019) 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, that has its elements and relations and used by an enterprise architect to visualize an enterprise layer, a subsystem or a pattern (Lankhorst et al., 2010).

ArchiMate gives its user maximal freedom in modeling. 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 modelling 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”(The Open Group, 2019) 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

<sup>a</sup>  <https://orcid.org/0000-0001-8308-0189>

<sup>b</sup>  <https://orcid.org/0000-0002-4067-3088>

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 and discusses a case study applying the proposed approach to enterprise modelling views and modelling conventions used in a medical center.
- Section 7 lists the pros and cons of the proposed approach and the stages of the Design Science Research that have been completed.
- Section 8 concludes the paper and draws on future work.

## 2 CONSTRAINTS IN ArchiMate

To understand how we impose constraints on ArchiMate models, let us briefly summarize the ArchiMate language. In each single view, ArchiMate shows a set of elements (boxes) and relations (lines) between those elements. An element has a shape that has been picked by the modeler from a limited set of shapes to reflect the type of the element. A line that connects two elements refers to a relationship between the two corresponding ArchiMate elements and also picked from a limited set of lines. Every view is perceived as a graphical diagram that shows specific details of an entire architecture. Each enterprise model has an internal ArchiMate model being a collection of elements, a collection of relationships, and a collection of views in which elements and relationships are shown. An attractive characteristic of ArchiMate is that different views can share the same elements and relationships. One element can even have a different shape in different views, but still be the same element.

The notion of "the same" is not defined in the ArchiMate reference document (The Open Group, 2019), but ArchiMate tools typically use an internal key to identify elements. Every element is categorized as a strategic, business, application, technological, motivational, or implementation element. All views are scoped in the namespace of the internal ArchiMate model.

An enterprise always exists in environment and works within restrictions that mean modelling conventions or constraints on the model of the enterprise. Analysing research literature, we have found numerous attempts to specify constraints in ArchiMate. Often, these attempts concern access, security, and privacy.

Mayer et al.(2019) propose to extend the ArchiMate metamodel with security policies (Mayer et al., 2019).

Zhi et al.(2018) have modeled assurance security cases graphically within an enterprise model (Zhi et al., 2018). They have extended the enterprise model with security policies, but these policies are not used to verify this enterprise model.

Blanco-Lainé et al.(2019) have modeled privacy policies in ArchiMate. The authors have a global look on privacy policies. Their work "addresses the modeling of a given regulation (GDPR) as an EAM fragment that needs to be integrated into a more global EAM"(Blanco-Lainé et al., 2019, page 14). They have identified 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 (and many other) case studies is that constraints are modelled without automated verification. This is not surprising, as ArchiMate tools offer little functionality for automated verification.

The "Constraint" element in ArchiMate allows the user to describe a constraint as text, but ArchiMate has no means to compute violations of that constraint.

ArchiMate does not distinguish between a constraint and a policy. In this research we do. A policy is a normative agreement (typically written in natural language) of the business. A constraint is a formal expression of which violations can be computed. To apply a policy in an enterprise architecture a policy must be made into a modelling convention and then into a constraint. Not all policies are amenable to that.

Modeling conventions that may start as policies must therefore be reformulated as constraints. We use a constraint to verify that modeling conventions are adhered to in ArchiMate models. The word "constraint" stands for a 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. Examples of constraints can be found in the sequel.

The remainder of this paper represents policies and modeling conventions as text (hence informal) and constraints as formulas. To formulate a constraint requires the knowledge of the ArchMate palette and the skill to formalize logical expressions. That is a skill for enterprise architects.

### 3 AUTOMATED CONSTRAINT VERIFICATION

Automated constraint verification has a long history, especially in the context of systems specification and enterprise architecture (Chapurlat and Braesch, 2008). Many tools for verification are available as query systems and analyzers. In this work we are interested specifically in verification of enterprise architecture models.

Babkin and Ponomarev (2017) take an approach based on relation algebra which is similar to ours (Babkin and Ponomarev, 2017). They have made a metamodel of the ArchiMate language in Alloy (Jackson, Daniel, 2006) and used it to analyse the ArchiSurance (Jonkers et al., 2016) 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 have shown with examples that their approach to analysing ArchiMate models works. We take this result one step further by exploring a practical perspective: What do enterprise architects need to benefit from analysing their ArchiMate models?

Arriola and Markham (2018) 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 (Arriola and Markham, 2018).

Marosin et al.(2014) report an experience in the ontological specification of enterprise architecture and queries for verification of architectural specifications (Marosin et al., 2014).

The semantic web inspires researchers to represent constraints as OWL2 RL Axioms. Kharlamov et al.(2016) conclude that “the main challenge that we encountered was to capture the constraints of the models using ontological axioms” (Kharlamov et al., 2016, page 7).

The tool Archi (Beauvoir and Sarrodie, 2018) has recently been enriched with a JavaScript-based scripting plug-in called jArchi (Beauvoir and Sarrodie, 2019). It is built on the Oracle Nashorn engine. With jArchi, an enterprise architect can write JavaScript to encode his own ArchiMate checker.

From all these sources we conclude that there is a research problem of constraint elicitation and assessment on an enterprise model. In order to solve this problem, a process should be designed and followed. With this focus, our work can be classified as a Design Science Research (Peffer et al., 2007). The designed process includes formulating constraints, agreeing on their meaning and formalizing them in such a way that a constraint checker may produce the constraint violations. The produced violations have to be understood and evaluated (Cleven, Anne and Gubler, Philipp and Hünér, Kai M, 2009).

This work proposes a process of constraint elicitation and assessment of ArchiMate models. Having this focus, our work builds on the earlier work with the Ampersand tool (Joosten, 2018). Ampersand uses relational algebra (the same as Alloy) as a language to represent constraints, which allows verification of enterprise architecture against constraints. Unlike the approach in Alloy (Babkin and Ponomarev, 2017), we did not make an ArchiMate metamodel. Instead, the ArchiChecker derives the metamodel from the ArchiMate model. This ensures that there can be no discrepancy between the metamodel and the ArchiMate data. This has saved us not only the time to make a metamodel, but it also prevented programming mistakes when matching the data from an ArchiMate model to the metamodel.

Ampersand is also used for constraint checking but it also has an established way of rule elicitation (Wedemeijer, 2014). The Ampersand compiler has been extended with a parser that reads ArchiMate repositories (Filet et al., 2019). Building on these earlier results, we have used Ampersand to complement ArchiMate with respect to automated verification.

### 4 ArchiChecker- HOW IT WORKS

To understand how the constraint checker (ArchiChecker) works, let us observe the tools (Figure 1).

Two tools serve the enterprise architect: an ArchiMate modeling tool and the ArchiChecker. For modelling we use the open source tool Archi, which stores an ArchiMate model in the form of an XML-file with extension “.archimate”. In Figure 1 this file is called “repo.archimate”.

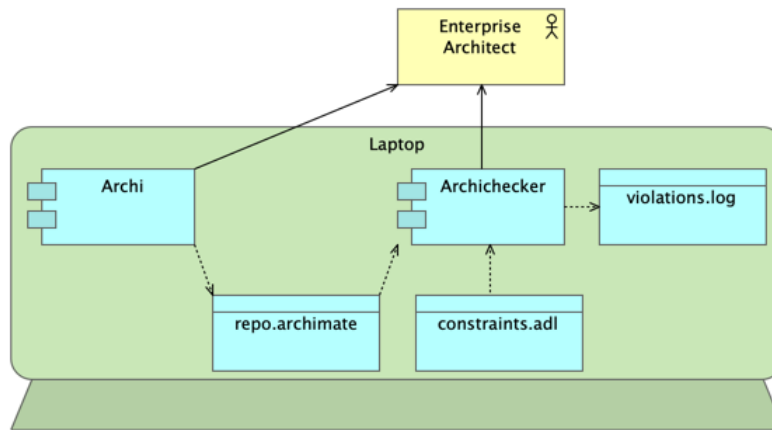


Figure 1: A toolset for an enterprise architect.

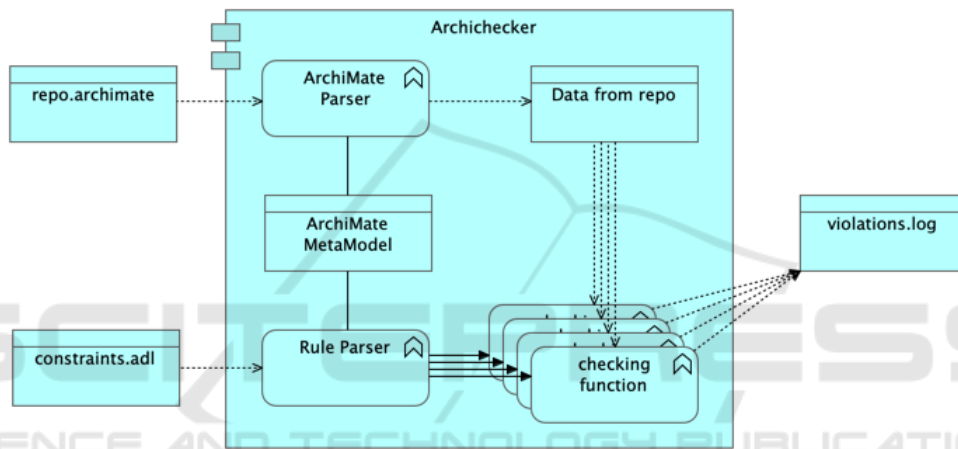


Figure 2: Under the hood of the ArchiChecker.

The component “ArchiChecker” (Figure 2) functions as a compiler that compiles the ArchiMate model (“repo.archimate”) together with the constraints (constraints.adl) into a set of violations (“violations.log”). It parses the ArchiMate model (repo.archimate) according to the ArchiMate Meta-model. It parses constraints (constraints.adl) according to the same meta-model because a constraint checker must “know” where to find the data to be checked. ArchiChecker emits all violations to the log.

To gain some understanding of the internal functioning, let us look at a small fragment (in Ampersand (Joosten, 2018)):

```
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,TOT]
```

```
RELATION name [ApplicationFunction*Text] [UNI]
```

The metamodel consists of relations. 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 Table 1, each of which contains pairs.

Table 1: ArchiMate notions vs. Ampersand notions.

ArchiMate notion	Ampersand notion
relationship	relation
element type	concept
element	atom
constraint	rule
modeling convention	-
model	context
view	-

For example, the relation realization represents a set of pairs, each of which relates one particular Business Function to one Business Service. An af-

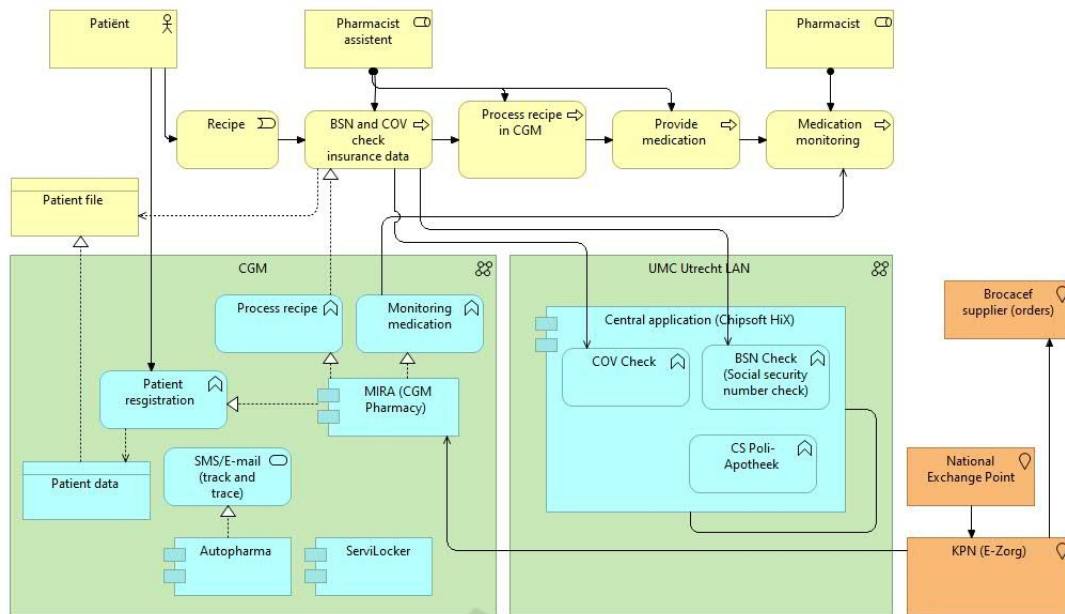


Figure 3: Enterprise model “CS Pharmacy”.

fix (e.g. [UNI]) is used to denote a set of multiplicity properties, of which there are four: UNI (which means univalent), TOT (total), INJ (injective), and SUR (surjective). They can be used in any combination.

**Data.** The data in an ArchiMate model are interpreted in terms of pairs in Ampersand and each pair is contained in the appropriate Ampersand relation. The following fragment illustrates how data from ArchiMate is allocated to relations.

```
Data from ArchiMate is allocated to relations:
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 model. The long numbers that constitute pairs are the internal ArchiMate keys for ArchiMate elements.

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

**Constraints.** An example of a constraint, written in Ampersand, which resides in a file with extension .adl

is shown below.

```
RULE "MC 3":
I[ApplicationComponent]
|- serving ;
serving[ApplicationComponent*BusinessProcess];
serving`
```

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.

The formulation of violation is specified for each rule. For example, the violation specification for RULE "MC 3":

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

Using this specification ArchiChecker writes to the log the found violations:

```
Violations of RULE "MC 3":
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 model by writing the modeling conventions in the form of constraints, followed by running the ArchiChecker on the model and the constraints.

## 5 A METHOD FOR CONSTRAINT PREPARATION FOR AUTOMATED ASSESSMENT OF ENTERPRISE MODELS

In business documents, one will rarely find constraints that are ready for automatic checks. Business policies abound, however. They are mostly formulated in natural language.

A **method** of formalizing a business policy for ArchiChecker looks as follows.

### Given:

- An ArchiMate view (a visual enterprise model and the corresponding internal ArchiMate model).
- A policy found in an enterprise architecture document.

1. Reformulate the policy in natural language.
2. Visualize the policy in terms of ArchiMate element types, properties and relationships.
3. Formulate an expression of the policy using Predicate Logic. Formulate the rule in Ampersand.
4. Run the ArchiChecker on the ArchiMate model to generate a list of policy violations.
5. Analyse the found violations and the given ArchiMate view to find possible reasons of violations and possible actions.

## 6 A CASE STUDY

To test our method in practice, we have conducted a case study in the Utrecht Medical Center (UMC) in the Netherlands. The Enterprise Architecture of the UMC is partly derived from the nation-wide Hospital Reference Architecture ZiRA (ZIRA, 2019). For this paper we show one ArchiMate view (a model at one level of abstraction) CS Pharmacy (Figure 3).

**Pharmacy View.** In the real 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. The aim of the CS Pharmacy project is to change the enterprise architecture by deploying the application HiX CS Pharmacy within the UMC Utrecht local area network (UU-LAN) to make

available all necessary functionality for the pharmacy.

**Business Policies in Documentation.** We have found business policies in documents called Project Start Architecture (PSA) and selected ten that were suitable for turning into modeling conventions. We used the method shown in section 5 for all ten policies and checked the ArchiMate CS Pharmacy view for violations.

The selection consists of the following policies:

1. Only one information system is in use for each functionality.
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. Use of central applications is mandatory.
9. The core of information provision is an Enterprise Data Warehouse (EDW).
10. Every data and data type has someone responsible.

In this paper we show the formalization of 8 policies. The formalization for all constraints and assessment of two ArchiMate views are available in our report (Joosten et al., 2020).

### 6.1 Policy 1 - Only One Information System Is in Use for Each Functionality

1. **Reformulated in Natural Language.** Every functionality must be realized by precisely one application.
2. **Visualisation in ArchiMate**



3. **Predicate Logic.** The phrase “exactly one” gives rise to two distinct expressions:

For every application function, there exists at least one incoming realization relation from an application component.

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

For every application function, there exists at most one incoming realization relation from an application component.

$\forall f \in ApplicationFunction$   
 $\forall c_0, c_1 \in ApplicationComponent :$   
 $c_0 \text{ realization } f \wedge c_1 \text{ realization } f \Rightarrow c_0 = c_1.$

**Rules 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 ".")
    
```

4. **Violations Generated by ArchiChecker.** There is one violation of the Rule “MC 1.a” in view CS Pharmacy in Figure 3.

ApplicationFunction ‘CS Poli-Apotheek ’ is not realized by any ApplicationComponent.

There are no violations of RULE “MC 1.b”.

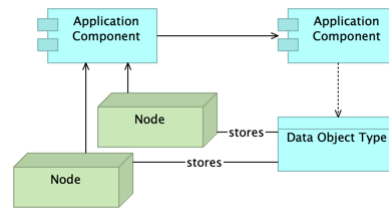
5. **Understanding the Reasons of Violations.** The ArchiChecker has identified an ApplicationFunction that is not realized by any ApplicationComponent. This might indicate that the model is not complete.

**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 don’t 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. **Reformulated in Natural Language.** Store once and use manifold, to ensure that users get correct data.
2. **Visualisation in ArchiMate.** 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.



**3. Predicate Logic**

$\forall d \in DataObject, \forall n, n_1 \in Node,$   
 $\forall s, s_1 \in ApplicationComponent :$   
 $(n \text{ stores } d \wedge n \text{ serving } s) \wedge$   
 $(n_1 \text{ stores } d \wedge n_1 \text{ serving } s_1) \Rightarrow s = s_1$

**Rules 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])
    
```

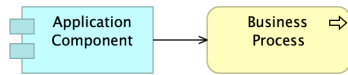
4. **Violations Generated by ArchiChecker.** This script of ArchiMate yields no violations. For architects it is interesting to experiment with the ArchiMate model to make examples to violate this rule deliberately. In this way they can try out their modeling conventions in practice.

5. **Understanding the Reasons of Violations.** The selected policy does not constitute any violation (Figure 3). 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

We have found that Policy 3 needs an explanation of its goal, namely, the business should know the serving application system, i.e. should be served with an unambiguous source system to which other applications are connected.

1. **Reformulated in Natural Language.** Each business process is served by at least one application component.
2. **Visualisation in ArchiMate.** For every business process, there must be precisely one incoming serving relation from an application component.



#### 3. Predicate Logic

$\forall f \in ApplicationComponent$   
 $\exists c \in BusinessProcess : c \text{ serving } f.$

#### Rules 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.")
    
```

#### 4. Violations Generated by ArchiChecker. Violations of RULE "MC 3" in the model CS Pharmacy, Figure 3:

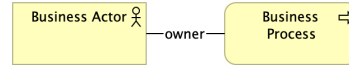
```

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.
    
```

#### 5. Understanding the Reasons of Violations. The reason of violations in this case is the difficulty in visualization of this policy. The architects have decided to make a separate table presenting the serving relations demanded by Policy 3. This table is aimed to accompany Figure 3.

### 6.4 Policy 4 - Every Business Process Has Precisely One Owner

1. **Reformulated in Natural Language.** Every business process has precisely one owner.
2. **Visualisation in ArchiMate**



#### 3. Predicate Logic

MC 4.1.

$\forall b \in BusinessProcess \exists a \in BusinessActor : a \text{ owner } b.$

MC 4.2.

$\forall s \in BusinessProcess \forall a, b \in BusinessActor : a \text{ owner } s \wedge b \text{ owner } s \Rightarrow a = b.$

#### Rules 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
    
```

#### 4. Violations Generated by the ArchiChecker. The violations of RULE "MC 4.1" in enterprise model CS Pharmacy:

```

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

There are no violation of "MC 4.2".

#### 5. Understanding the Reasons of Violations. In the case of CS Pharmacy (Figure 3) a role / owner has been assigned for a crucial process, namely "medication monitoring". The architect have chosen not to visualize other owners. It is assumed that an owner has been assigned for each business process.

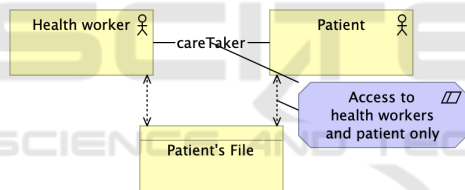
### 6.5 Policy 5 - Healthcare Providers and Patients Work with One Shared File

This policy needs explanation of its goal. 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 information that can be



easily found 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. Reformulated in Natural Language.** For every patient, there must be one file called “Patient’s 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. Visualisation in ArchiMate.** The rule “A patient’s 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.



**3. Predicate Logic**

Let *patient* be a predicate on BusinessActor.  
 Let *patientfile* be a predicate on BusinessObject.  
 Let *careTaker* be a relation BusinessActor × BusinessActor that relates health workers and their own patients.

$$\forall a, b \in BusinessActor \forall o \in BusinessObject : a \text{ caretaker } b \Rightarrow b \text{ access } o \wedge a \text{ access } o.$$

$$\forall a, b \in BusinessActor \forall o \in BusinessObject : a \text{ caretaker } b \Rightarrow b \text{ oaccess } o \wedge a \text{ access } o.$$

**Rules 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
```

**4. Violations Generated by the ArchiChecker**

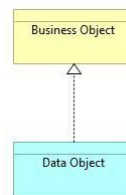
```
PatientFile (Business Object) 'Patient file '
is not accessed by
a caretaker/Patient (Health worker).
```

- 5. Understanding the Reasons of Violations.** The architects have acknowledged that visualization of the fact that the caretaker has access to the patient 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 are 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 of Policy 6 is to spot redundant data or incomplete architecture models.

- 1. Reformulated in Natural Language.** Every Data Object realizes one or more Business Objects.
- 2. Visualisation in ArchiMate**



- 3. Predicate Logic.**  $\forall b \in DataObject \exists o \in BusinessObject : c \text{ realization } b.$

**Rules 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")
```

- 4. Violations Generated by the ArchiChecker.** This script of ArchiMate contains no violations.

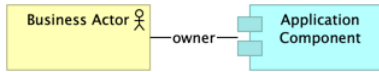
5. **Understanding the Reasons of Violations.** There is just one data object, 'Patient data', which resides in model CS Pharmacy Figure 3. Since it is connected to Business object 'Patient file' by a realization relation, there are no violations of this policy.

### 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.

1. **Reformulated in Natural Language.** Each IT system has an owner responsible for ensuring continuity and taking adequate measures.

#### 2. Visualisation in ArchiMate



#### 3. Predicate Logic

$$\forall a \in ApplicationComponent$$

$$\exists b \in BusinessActor : b \text{ owner } a.$$

$$\forall b, d \in BusinessActor,$$

$$\forall a \in ApplicationComponent :$$

$$b \text{ owner } a \wedge d \text{ owner } a \Rightarrow b = d.$$

#### Rules 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.")
    
```

#### 4. Violations Generated by the ArchiChecker

```

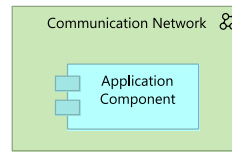
Application Component 'Brocacef supplier (orders)'
does not have an owner.
Application Component
'Central application (Chipsoft HiX)'
does not have
an owner.
Application Component 'Autopharma '
does not have an owner.
Application Component 'MIRA (CGM Pharmacy)'
does not have an owner.
Application Component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

There are no violations of "MC 7.2".

5. **Understanding the Reasons of Violations.** Non of application components of the enterprise model CS Pharmacy (Figure 3) has an owner. The continuity of data of these components is not guaranteed.

### 6.8 Policy 8 - Use of the Central Application Is Mandatory

1. **Reformulated in Natural Language.** An application component is central if it is inside one of the communication networks of the organization.
2. **Visualisation in ArchiMate**



#### 3. Predicate Logic

$$\forall a \in ApplicationComponent :$$

$$\exists c \in CommunicationNetwork :$$

$$a \text{ inside } c.$$

#### Rules in Ampersand

```

RULE 10:
  I[ApplicationComponent] |- inside ;
  I[CommunicationNetwork] ; inside
VIOLATION (TXT "Application Component '\'",
  SRC name,
  TXT "\" is not inside a LAN")
    
```

#### 4. Violations Generated by the ArchiChecker

```

There are 3 violations:
Application Component 'Brocacef supplier
(ordere)
is not inside a LAN
Application Component 'National Exchange
Point
is not inside a LAN
Application Component 'EZorg
is not inside a LAN
    
```

5. **Understanding the Reasons of Violations.** The application components mentioned in violations are presented in Figure 3. 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 DISCUSSION

In this paper we have presented a method

- for transformation of business policies to modeling conventions and constraints on enterprise models in ArchiMate and
- for automatic assessment of ArchiMate model in form of generated violations of constraints expressed in terms of analysed enterprise models.

In order to increase the readability of our paper, we have presented a part of the testing results of the method on the Enterprise model “CS Pharmacy” and eight policies found in the enterprise architecture documents. In fact, the proposed method has been tested on two ArchiMate models and ten policies. The second ArchiMate model used for testing of our method is called “CS Office 365”. It shows the implementation of the standardized platform Office 365 for collaboration within UMC Utrecht and outside the organization. The results of testing can be found in (Joosten et al., 2020).

The application of our method to the enterprise models “CS Pharmacy” and “CS Office 365” and all policies have shown several interesting observations:

- Formalization of modeling conventions is far from trivial and should not be underestimated.

Multiple authors have noted that the acquisition of constraints is the real challenge architects face (Babkin and Ponomarev, 2017; Ramos et al., 2014; Chatzikonstantinou and Kontogianis, 2012). Our effort to analyze enterprise models and their constraints underwrites this claim.

- The policies found in an organization can be reused for its enterprise models.

The formalized constraints corresponding to policies can be reused by renaming of elements. A pool of often used policies can be collected for reuse in each organization. We have reused the pool of ten policies for two enterprise models.

- The formalization of a policy into a constraint makes the policies clearer to enterprise architects and business.

The formalization of policies contributes to the quality of enterprise models as models are corrected to meet constraints.

Having tested our method for two ArchiMate models and ten policies we have covered all phases of the Design Science Research including evaluation (Cleven, Anne and Gubler, Philipp and Hüner, Kai M, 2009).

## 8 CONCLUSIONS AND FUTURE WORK

A method of constraint formalization for automated assessment of enterprise models presented in this paper can be used as it has been designed and tested.

It should be, however, known, that the method is labor-consuming. The difficulties are caused by the agreements that should be achieved in understanding of policies by people having different roles in organization. Also the transformations of agreements into constraints demands accuracy. The enterprise models serve as clarification and transformation means for policies.

Experimenting with our method, we have found that a business policy and the corresponding constraint cannot be understood without understanding the goal of the Enterprise Model in hand and the goal of the business policy in it. The method and tooling presented in this paper can be seen as a part of Enterprise modelling approach that includes goal modelling. We are working on an approach for consistent modelling in ArchiMate that includes goal models refined to requirements and associated with policies and constraints within each enterprise multi-view model. The applicability of constraints associated with a goal model is easier to understand and easier to make decisions when views violate such constraints. In future work, we are planning to include the method for constraint checking proposed in this paper into the approach for consistent modelling in ArchiMate.

## ACKNOWLEDGEMENTS

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

## REFERENCES

- Arriola, L. and Markham, A. (2018). Towards an enterprise architecture controlling framework. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, pages 1–7.
- Babkin, E. A. and Ponomarev, N. O. (2017). Analysis of the consistency of enterprise architecture models using formal verification methods. *Business Informatics*, (3):30–40.
- Beauvoir, P. and Sarrodie, J. (2018). Archi-the free archi-mate modelling tool. *User Guide. The Open Group*.
- Beauvoir, P. and Sarrodie, J.-B. (2019). Archi-Open Source Archimate Modelling.

- Blanco-Lainé, G., Sottet, J.-S., and Dupuy-Chessa, S. (2019). Using an enterprise architecture model for GDPR compliance principles. In *IFIP Working Conference on The Practice of Enterprise Modeling*, pages 199–214. Springer.
- Chapurlat, V. and Braesch, C. (2008). Verification, validation, qualification and certification of enterprise models: Statements and opportunities. *Computers in Industry*, 59(7):711–721.
- Chatzikonstantinou, G. and Kontogiannis, K. (2012). 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)*, pages 27–36. IEEE.
- Cleven, Anne and Gubler, Philipp and Hüner, Kai M (2009). Design alternatives for the evaluation of design science research artifacts. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, pages 1–8.
- Filet, P., van de Wetering, R., and Joosten, S. (2019). Enterprise architecture alignment. *researchgate.net. Department of Information Sciences, Open University of the Netherland*.
- Jackson, Daniel (2006). *Software Abstractions: Logic, Language, and Analysis*. The MIT Press.
- Jonkers, H., Band, I., Quartel, D., and Lankhorst, M. (2016). *ArchiSurance Case Study version 2*. The Open Group.
- Joosten, S. (2018). Relation Algebra as programming language using the Ampersand compiler. *Journal of Logical and Algebraic Methods in Programming*, 100:113–129.
- Joosten, S., Haddouchi, E. M., and Roubtsova, E. (EasyChair, 2020). Design policy checking in archimate. on the cases technology policies in a medical center. EasyChair Preprint no. 4129.
- Kharlamov, E., Grau, B. C., Jiménez-Ruiz, E., Lamparter, S., Mehdi, G., Ringsquandl, M., Nenov, Y., Grimm, S., Roshchin, M., and Horrocks, I. (2016). Capturing industrial information models with ontologies and constraints. In *International Semantic Web Conference*, pages 325–343. Springer.
- Lankhorst, M. M., Proper, H. A., and Jonkers, H. (2010). The anatomy of the ArchiMate language. *International Journal of Information System Modeling and Design (IJISMD)*, 1(1):1–32.
- Marosin, D., Ghanavati, S., and van der Linden, D. (2014). A Principle-based Goal-oriented Requirements Language (GRL) for Enterprise Architecture. In *iStar*.
- Mayer, N., Aubert, J., Grandry, E., Feltus, C., Goettelmann, E., and Wieringa, R. (2019). An integrated conceptual model for information system security risk management supported by enterprise architecture management. *Software & Systems Modeling*, 18(3):2285–2312.
- Peffer, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77.
- Ramos, A., Gomez, P., Sánchez, M., and Villalobos, J. (2014). Automated enterprise-level analysis of Archimate models. In *Enterprise, Business-Process and Information Systems Modeling*, pages 439–453. Springer.
- The Open Group (2012-2019). ArchiMate 3.1 Specification.
- Wedemeijer, L. (2014). A relation-algebra language to specify declarative business rules. In *4th International Symposium on Business Modeling and System Design, BMSD*, pages 63–73.
- Zhi, Q., Yamamoto, S., and Morisaki, S. (2018). IMSA-Intra Model Security Assurance. *J. Internet Serv. Inf. Secur.*, 8(2):18–32.
- ZIRA (2019). Ziekenhuis Referentie Architectuur <https://sites.google.com/site/zirawiki/>.