Design Policy Checking in ArchiMate. On the Cases Technology Policies in a Medical Center

Stef Joosten, El Makki Haddouchi and Ella Roubtsova

September 2, 2020

Design Policy Checking in ArchiMate

On the cases technology policies in a Medical Center

Stef Joosten [0000-0001-8308-0189], El Makki Haddouchi, Ella Roubtsova [0000-0002-4067-3088]

Open University of the Netherlands, Valkenburgerweg 177, 6419 AT, Heerlen, The Netherlands
Stef.Joosten@ou.nl,E.M.Haddouchi@umcutrecht.nl, Ella.Roubtsova@ou.nl

Abstract. This paper presents the results of an investigation on the automatic verification of organization-specific policies in the context of the Open Group enterprise modeling language ArchiMate. This paper analyses a set of technical design policies from a hospital, reformulating them for automated checking within the hospital's own architecture models. The policies have been modeled in a semantic language called Ampersand, which is based on relation algebra. This investigation has used Ampersand as an architecture checking tool. By analyzing 10 real-world policies from a hospital, we demonstrate that policies can be verified automatically in ArchiMate models. This work is a step towards the automated checking of architecture policies, which enhances the possibilities of ArchiMate beyond visualization only.

Keywords: Enterprise Architecture, ArchiMate, policy, automated verification, Ampersand tool

1 Introduction

The Open Group enterprise modeling standard language ArchiMate [2, 18, and 19] is available to enterprise architects for visualizing and sharing ideas. Enterprise architects create ArchiMate models, which consist of elements, relationships between those elements, and views. Each view corresponds to a diagram, which an enterprise architect uses to visualize whatever she considers useful. However, the purpose of enterprise architecture is not just to draw diagrams but to ensure coherence (i.e. consistency and completeness) across the enterprise. Models and diagrams are instruments, not the results of enterprise architecture.

In practice, architects translate the enterprise's visions to practical policies, to guide the enterprise in the direction of the desired vision. This paper addresses the problem of verifying such policies in ArchiMate models, as a step towards automated verification of enterprise architecture.

This investigation addresses the question of whether we can formalize modeling policies of enterprise architects so that these policies can be verified automatically on an entire ArchiMate model (i.e. across all diagrams together).

The fact that ArchiMate allows an enterprise architect to model almost anything complicates things. Different architects use different modeling practices. A lack of uniformity makes it hard to verify architecture policies, so automated support might be very welcome. ArchiMate itself, however, provides little help because it is very permissive. And understandably so, because a tool should not prescribe how it is used.

To address this problem, we have formalized policies to verify an ArchiMate model wrt these policies. This paper presents an experiment that was conducted to test this method in practice, in a hospital. Section 2 describes the published attempts of specification and verification of different policies in ArchiMate. Section 3 presents our method for policy presentation and verification useful for ArchiMate enterprise models. Section 4 presents the real policies found in documents of a hospital and shows their preparation for automatic verification. The results of the verification are transformed into reports about the conformance of enterprise models to policies. Section 5 discusses the results of our experiment on policy specification and verification in ArchiMate. Section 6 concludes the paper and presents future work.

2 Policies in ArchiMate

An Enterprise model presents an enterprise with the concepts and relations of different layers of an enterprise [17]. Its purpose is to support architects to visualize structure in different ways for different stakeholders. ArchiMate is a standard of Enterprise modeling [2]. The visual elements of ArchiMate are elements (visually represented by different shapes with a surface) and relationships (visually represented by shapes without surfaces, such as arrows and lines). The visual elements can be included in other elements, for example by aggregation, composition, and grouping relations. Elements are categorized as strategic, business, application, technological, motivational, or implementation elements. One model contains an arbitrary number of diagrams (called "view"). Different views can share elements and relationships, i.e. all views are scoped in the namespace of the entire model.

In June 2016, the Open Group released version 3.0 of the ArchiMate Specification [2]. Version 3.0 offers the modeling element in its motivation layer: Constraint (Policy), shown in Fig. 1.

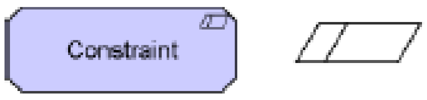


Fig. 1. Constraint (Policy) in ArchiMate® 3.1 Specification [2]

This means that organization-specific constraints (policies) can be entered as text in the ArchiMate language. Throughout this paper we use the word "constraint" for a verifiable statement that is intended to be true all the time, e.g. "an invoice must be paid within 30 days". We use the word policy for an intention, which is not necessarily verifiable, e.g. "Our company is transparent". A constraint is a specialization of a policy. This paper uses constraints in the context of enterprise models. They are expressed in terms of the elements and relationships of the enterprise model.

This paper assumes that policies are formulated by management, not by enterprise architects. In practice, this means that:

- The policies are to be found in various documents scattered around the organization. An enterprise architect may have to track them down.
- For automatic verification, an enterprise architect must reformulate policies into constraints. If a policy is not verifiable, the enterprise architect must choose to make it verifiable or to ignore it.
- Reformulation of policies may involve some validation activity in collaboration with management.

The possible added value of including constraints and policies into enterprise models is the possibility of their automated checks.

3 Related Work

Researchers have made numerous attempts to specify constraints for ArchiMate models. Often, these attempts concern access, security, and privacy.

For example, Korman, M., Lagerström, R., and Ekstedt, M. [12] 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” 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 modeled but not subjected to automated verification.

Tepandi, J. et.al. [22] 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 on the Once-Only Principle and its scenario-based evaluation. The architecture, constraints, and policies are presented informally and not verified.

Security policies have got an ArchiMate meta-model extension proposed by Mayer, N., Aubert, J., Grandry, E., Feltus, C., Goettelmann, E., & Wieringa, R. [13]. Assurance security cases graphically modeled within the enterprise model by Zhi, Q., Yamamoto, S., & Morisaki, S. [14]. A developed security policy extends the enterprise model, but the policies are not used for the verification of enterprise architecture.

Privacy policies have been modeled in the ArchiMate by Blanco-Lainé G., Sottet J-S, Dupuy-Chessa S. [15]. 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”. 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 ArchiMate modelers are modeling policies and constraints without automated verification. This is not surprising, as ArchiMate tools offer little functionality for automated verification.

Automated constraint verification in models has a long history, especially in the context of systems specification and enterprise architecture [1, 3]. Many tools for verification are available as query systems and analyzers.

In this work we are interested specifically in verification of the enterprise architecture models. The current paper builds on earlier work with the Ampersand tool [8].

Ampersand uses relational algebra as a language to represent constraints, which allows verification of enterprise architecture against constraints.

Like the current paper, Babkin, E. A., & Ponomarev, N. O. [6] take an approach based on relation algebra. The MIT Alloy Analyzer searches for contradictions in the enterprise architecture models. Babkin & Ponomarev propose a meta-model of the ArchiMate specification. The authors use the standard case (coming with the ArchiMate installation) ArchiSurance and one example of a constraint to illustrate the use of the MIT Alloy Analyzer.: "If an application component uses data, it must have access to a technology service that extracts that data." This research found that the main difficulty is to find the constraints that need to be checked in organizations.

Arriola, L., & Markham, A. [7] 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 & Markham.

The semantic web inspires researchers to represent constraints as OWL2 RL Axioms. Kharlamov et.al. [4]). Marosin et.al. [5] report their experience in the ontological specification of enterprise architecture and use queries for verification of architectural specifications. They too found that *"the main challenge that we encountered was to capture the constraints of the models using ontological axioms".* The common ground in these publications is that identification and formulation of constraints is the major challenge.

A more generic way to represent constraints is to extend a concrete tool. The tool Archi [11, 20] has recently been enriched with a JavaScript-based scripting plug-in called jArchi [10]. It is built on the Oracle Nashorn engine. With jArchi, an enterprise architect can write JavaScript to encode his own ArchiMate checker. That option is more generic than constraint verification, since any plugin can be written. However, it is confined to Archi.

In practice, organizations use combinations of design, security, privacy, and other policies. In order to make sense to the business, these policies are formulated using the language and jargon of the business. So capturing business language is required. Besides, formality is required to do automated verification. Therefore this work uses a formal language, Ampersand [8] in which an enterprise architect can capture the language, i.e. the concepts and relations, of the business. Since ArchiMate has no embedded constraint language, we have adapted the Ampersand compiler to read ArchiMate repositories [9]. That work presents a parser that extracts the ArchiMate metamodel from an ArchiMate repository and pours the ArchiMate models from that repository into Ampersand. From that point onward, Ampersand takes over to do all the checking.

4 Research method to study the checking of policies

Our research method is based on the presentation of an enterprise architecture in ArchiMate®Model Exchange File Format [18] and its compilation into a metamodel with an Ampersand-compiler called ArchiChecker.

4.1 Our verification method in steps

1. Automated step. <u>ArchiMate model importing and parsing</u>.

The investigator imports an ArchiMate model. ArchiChecker parses the ArchiMate model and generates a metamodel, which is populated by the contents of the ArchiMate model.

Suppose the repository contains the following model fragment:

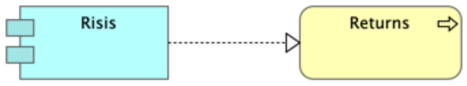


Fig. 2. A fragment of an ArchiMate model

Then the metamodel contains an ApplicationComponent **Risis**, a BusinessProcess R**eturns** and a relation **realization**. i.e the pair **(Risis, Returns) ∈ realization.**

2. Manual step. <u>Policy formalization.</u>

The researcher formalizes each policy in the language Ampersand as a constraint on elements and relations accommodated in the metamodel of the ArchiMate model. This requires involvement of a researcher mastering formalization of an informal policy into an Ampersand constraint.

The policy in hands is formulated (1)in natural language; (2)presented as an ArchiMate picture; (3)presented in Predicate Logic;(4) transformed into Ampersand language for ArchiChecker.

3. Automated step. <u>Checking process.</u>

An ADL file is created as it is shown in Fig.3. The ArchiMate model and the set of relations and the constraint corresponding to a policy are used in a constraint checking process, in which the content of the relations is queried to find violations of the policy in the metamodel of the ArchiMate model.

```
CONTEXT ArchiMate Model
INCLUDE "ArchiMate Model Name"
RELATION metamodel of the ArchiMate model
RULE policy in Ampersand language for ArchiChecker.
VIOLATION  generated by the ArchiChecker
 ENDCONTEXT
```

Fig. 3. ADL File - the selected constraint in Ampersand for verification

In order to automatically detect violation, the Archi file is imported in Ampersand using the following script in Docker:

*docker run -it -v **C:\(workdirectory)**:/scripts ampersandtarski/ampersand:**(version)** check example.archimate*

During the script execution, the files 'ArchiCount.txt and ArchiMetaModel.adl' will be added to the workdirectory.

Then the 'example.adl' is imported using the following script in Docker:

*docker run -it -v **C:\(workdirectory)**:/scripts ampersandtarski/ampersand**:(version)** check constraintexample.adl*

The ArchiChecker generates the violations of the selected constraint (policy) in the selected EA model.

4. Partially automated step. Analysis of violations.
ArchiChecker presents every violation in a human-readable form, which the researcher has prepared in Ampersand. The analysis is fulfilled by the enterprise architect.

5 A case study of formalizing and checking of ten design policies in two enterprise architecture models

5.1 Description of the case study

The Enterprise Architecture (EA) of the Utrecht Medical Center (UMC) in the Netherlands is partly derived from the Hospital Reference Architecture ZiRA [16]. It contains a collection of policies and models for each layer. The guidelines formulated in the organization and processes layer are based on the policy made by the Executive Board, the divisions and the departments. These are generic policy statements that guide all layers. Every project selects relevant policies (i.e. principles and guidelines) from the EA. The selection is then included in the "Project Start Architecture" (PSA), to focus the attention of all project members to these policies. To demonstrate our approach we have selected (more or less randomly) two distinct models from distinct domains in the actual enterprise architecture of the UMC. One was taken from the PSA “CS Pharmacy” and the other from the PSA “Office 365”. Using two models can show which rules are applicable to both models, even though these models are unrelated. Let us briefly discuss each model.

PSA: CS Pharmacy. In this existing project start architecture two information systems are used: MIRA (CGM Pharmacy) and Chipsoft HiX. The systems do not communicate with each other because of the separated Infrastructures. CGM and Chipsoft HiX can be seen in Fig. 4. MIRA (CGM Pharmacy) runs completely separate from the UMC Utrecht infrastructure. Management is carried out by the supplier. The aim of the CS Pharmacy project is to change the enterprise architecture by deploying the application HiX CS Pharmacy within the UMC Utrecht Lan to make available all necessary functionality for the pharmacy.

PSA: Office 365. UMC Utrecht has developed a strategy that combines both CGM 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 Fig.5. It shows the support of Personnel administration, Access facilities, Access to ICT, conferencing, Suppl chat, e-mail, address, agenda.

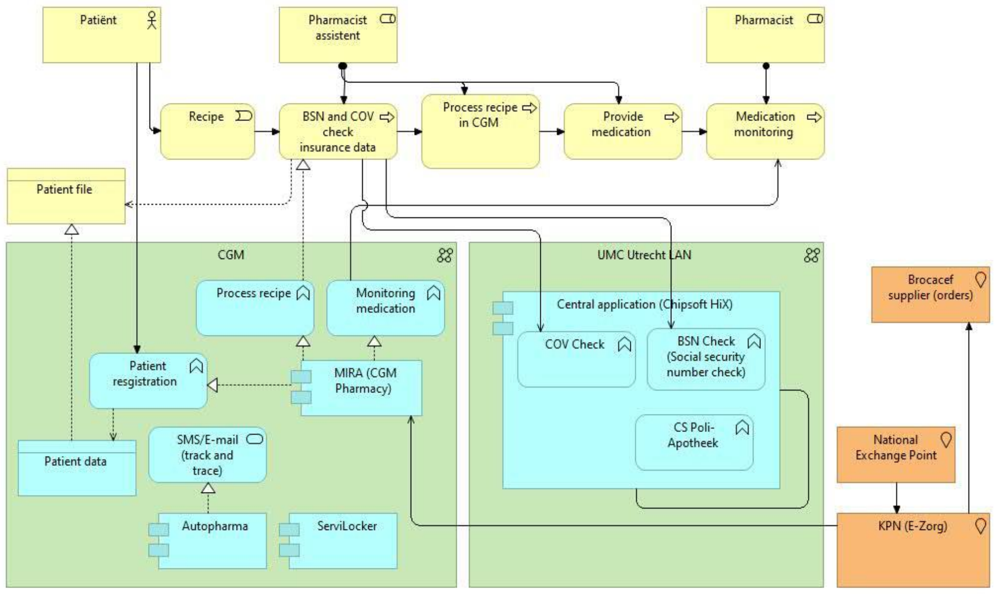


Fig. 4. PSA: CS Pharmacy

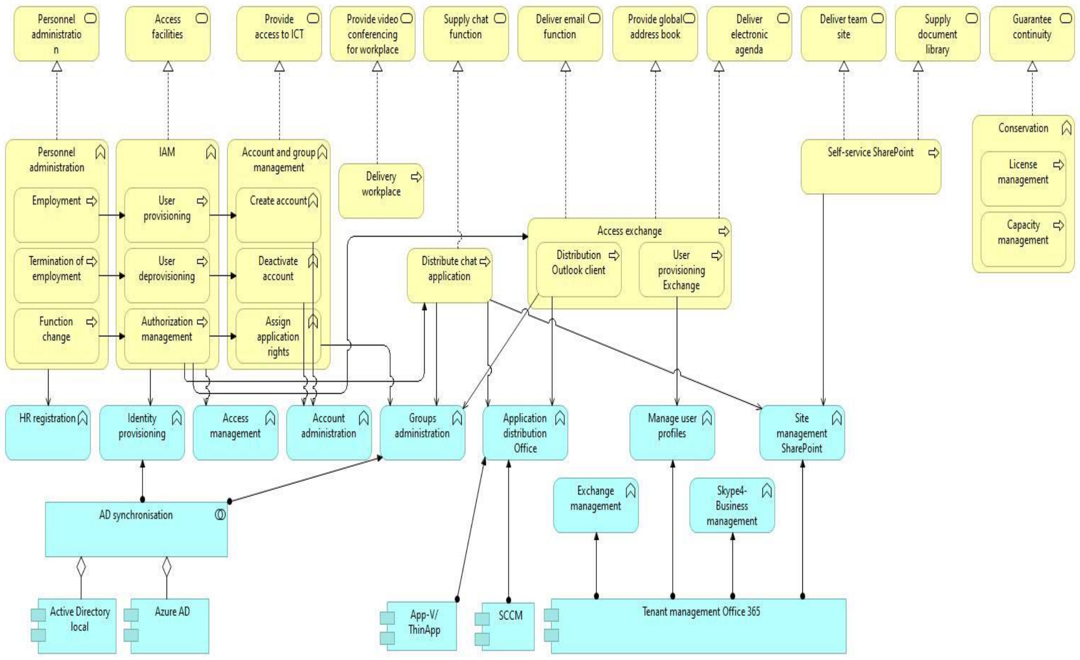


Fig. 5. PSA: Office

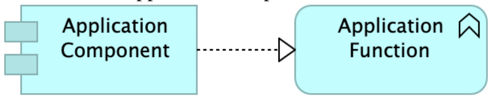
Policies and Constraints. By analyzing 10 different policies we can obtain a sense of the problems we encounter when formalizing architectural constraints in practice. The selection consists of the following policies:

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. *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.*

Our method for policy checks demands the formalization of constraints. The formalization is presented in tables 1and 2 and in Appendix. Let us now present and discuss the results, one by one.

5.2 Results of policy checking

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

Reformulated in natural language	Every functionality must be realized by precisely one application.
ArchiMate Practice	For every application function, there must be precisely one incoming realization relation from an application component. 
Predicate Logic	(Policy 2.1) $\forall f \in \mathit{ApplicationFunction}\ \exists c \in \mathit{ApplicationComponent} : c\ \mathit{realization}\ f$ (Policy 2.2) $\forall f \in \mathit{ApplicationFunction}\ \forall a,\ c \in \mathit{ApplicationComponent} : a\ \mathit{realization}\ f \wedge c\ \mathit{realization}\ f \Rightarrow a=c$

ArchiChecker rules	`RULE "Policy 2.1":`
	`I[ApplicationFunction] |- realization[ApplicationCompo-nent*ApplicationFunction]~ ;`
	`realization[ApplicationComponent*ApplicationFunction]`
	`VIOLATION (TXT "ApplicationFunction \'", TGT name, TXT "\' is not realized by any ApplicationComponent.")`

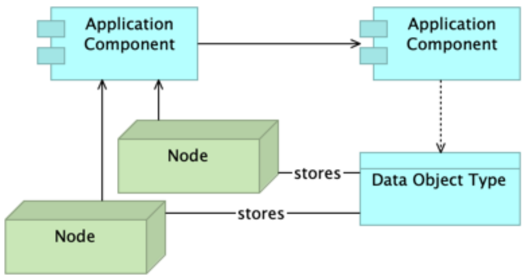
	`RULE "Policy 2.2":`
	`realization[ApplicationComponent*ApplicationFunction]~`
	`   ; -I[ApplicationComponent]`
	`   ; realization[ApplicationComponent*ApplicationFunc-tion] |-`
	`-I[ApplicationFunction]`
	`VIOLATION (TXT "ApplicationFunction \'", SRC name, TXT "\' is realized by an ApplicationComponent ", SRC reali-zation[ApplicationComponent*ApplicationFunction]~;name, TXT ".")`

Violations from Archichecker	*There are 11 violations of RULE "Policy 2.1":* *1) ApplicationFunction 'HR registration' is not realized by any ApplicationComponent*.**(the reader may see in Fig.4)** *2)ApplicationFunction 'Identity provisioning' is not re-alized by any ApplicationComponent.* **Fig.4** *3)ApplicationFunction 'Access management' is not real-ized by any ApplicationComponent.* **Fig.4** *4)ApplicationFunction 'Account administration' is not realized by any ApplicationComponent.***Fig.4** *5)ApplicationFunction 'Groups administration' is not re-alized by any ApplicationComponent.* **Fig.4** *6)ApplicationFunction 'Application distribution Office' is not realized by any ApplicationComponent.* **Fig.4** *7)ApplicationFunction 'Exchange management' is not real-ized by any ApplicationComponent.***Fig.4** *8)ApplicationFunction 'Manage user profiles' is not re-alized by any ApplicationComponent.* **Fig.4** *9)ApplicationFunction 'Site management SharePoint' is not realized by any ApplicationComponent.* **Fig.4** *10)ApplicationFunction 'Skype Business Management' is not realized by any ApplicationComponent.* **Fig.4** *11)ApplicationFunction 'CS Poli-Apotheek ' is not real-ized by any ApplicationComponent.* **Fig.3**

The reasons of violations need to be discussed with enterprise architects. It may be that not all functions have been implemented.

Table 2. Policy2: Unambiguous and one-time recording of data (and multiple use).

Reformulated in natural lan-guage	Store once and use manifold, to prevent data pollution and ensure that users get cor-rect data.

ArchiMate Practice	(Policy 1.1) When an application accesses a data object, it must do that through the (unique) application that manages these types of objects. (Policy 1.2) For every type of DataObject we must know the store(s) in which objects are stored. For example, “Personnel records will be stored in the HRM-database”. Let us agree to model the store as a Node, and let us relate the data object type with the node by means of an association relationship called “stores”. (Policy 1.3) Access to a specific data object type is channelled 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.  Application Component Application Component Node stores Data Object Type Node stores Note that data can be stored on multiple nodes. Access is given through one specific application component, which controls the integrity of all data objects of that type. Note too that Policy 1.2 follows logically from Policy 1.1 because of the way we chose to model this in ArchiMate. As a consequence, there is no need to check policy 1.2 in Ampersand.
Predicate Logic	(Policy 1.1) $\forall a \in ApplicationComponent, d \in DataObject :$ $\exists s \in Node,\ store \in ApplicationComponent : a\ access\ d \Rightarrow n\ stores\ d \wedge n\ serving\ store \wedge store\ serving\ a$ (Policy 1.2) $\forall d \in DataObject : \exists s \in Node : \exists a \in ApplicationComponent : n\ stores\ d \wedge n\ serving\ store$ (Policy 1.3) $\forall d \in DataObject : \forall n, n' \in Node : \forall s, s' \in ApplicationComponent : n\ stores\ d \wedge n\ serving\ s \wedge n'\ stores\ d \wedge n'\ serving\ s' \Rightarrow s = s'$

ArchiChecker rules	RULE "Policy 1.1": access[ApplicationComponent*DataObject]~ |- stores~ ; serving[Node*ApplicationComponent] ; (serving~ \/ I) VIOLATION (TXT "Application component \'", TGT name, TXT "\' has access to objects of type \'", SRC name, TXT "\'but there is nothing in which to store \'", SRC name, TXT "\'.") RULE "Policy 1.3": stores~ ; serving ; -I[ApplicationComponent] ; serv- ing[Node*ApplicationComponent]~ ; stores |- -I[DataOb- ject] VIOLATION (TXT "Data objects of type \'", SRC name, TXT "\' are stored in ", SRC stores~ ; serving[Node*Applica- tionComponent]
Violations from ArchiChecker	This script of ArchiMate contains no violation

In both EA models, the selected policy does not constitute any violation. This does not mean that the included rule for checking on the EA model does not constitute a violation. But in this case it is because the recorded objects are not visualized in the EA models. The architects have not chosen to visualize the rule in the models.

The ArchiCheck tables for other 8 policies are shown in the Appendix.

Table 3 presents the numbers of violations found per policy.

Table 3. Numbers of violations found per policy.

	Policy	N violations
1.	Only one information system is in use for each functionality.	11
2.	Unambiguous and one-time recording of data (and multiple use).	0
3.	Each business process should be realized by at least one application system.	12
4.	Every Business process has precisely one owner.	18
5.	Healthcare providers and patients work with one shared file.	1
6.	A data or data group uses one or more business objects.	0
7.	The continuity of critical systems of the Medical Center is guaranteed.	12
8.	The core of information provision is an Enterprise Data Warehouse (EDW).	12
9.	Every data and data type has someone responsible.	1
10.	Use of central applications is mandatory	7

6 Discussion of the case study of specification and verification of policies in ArchiMate

Although there is no specific viewpoint "constraint" in ArchiMate, yet we were able to express all constraints in ArchiMate graphically. We were able to formalize all constraints and this means that we can validate them in an Enterprise Model using the ArchiChecker. In this sense, the formalization of a constraint (policy) can be used as a pattern for EA modeling.

Tables 1, 2 and the tables in the Appendix demonstrate the results of the application of our method to a combination of two EA models and ten constraints (policies). All constraints-polices were applied to both models. The rules were expressed in the ArchiMate language (i.e. using phrases such as "application component" and "realization") rather than phrases from the business (with words such as "Autopharma", and "Chipsoft HiX"). But since the ArchiMate models are used to populate Ampersand's relations, the violations come out in the language of UMC, yielding meaningful violation statements.

The ArchiChecker identifies relatively large amounts of violations of organizational policies (Table 3). These violations do not necessarily mean that something is wrong with the architecture. An enterprise architect may use a rule just to notify peculiar situations or situations that deserve a little extra attention. So both the input (the formalized rules) and the output (the list of violations) cannot be 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 of policies is a specific point of attention. The enterprise model can use one set of concepts, but the policies can be formulated using another set of concepts. The understanding and interpretation of policies by different professionals demands communication between policymakers and enterprise architects.

Also, we have observed that some policies do not contain sufficient information for formalization. For example, policy 5 –" Healthcare providers and patients work with one shared file" (Appendix) has been refined to a set of statements "For every patient, there must be one file called "Patient's files". 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." Only after this refinement, policy 5 can be verified on enterprise models. This means that the policies should be refined and precisely formulated.

A remarkable observation is that all policies are merely multiplicity constraints. We can only speculate why there are so few more complicated constraints. Future investigation of organization-specific policies is needed to answer this question.

The formalized constraints, their automatic verification and found violation of policies may give policies another value in the organization. We have experienced some support of our research and interest both from management and enterprise architects, motivated as follows:

- Conformance to automatically verified policies can be easily reported and valued by management;
- Violation of policies triggers the improvement of enterprise architecture. If verification shows a violation of a policy in an enterprise model, it localizes the place in the EA model that does not conform to the policy. So, the reasons for violation can be analyzed and this may result in the correction of the Enterprise Architecture or to a refinement of the policy.

7 Conclusion and Future Work

This paper presents the results of an exploration of automatic verification of organization-specific constraints (policies). We have shown that this is possible in ArchiMate, which offers IT-architects new possibilities for consistency checks in Enterprise Architectures.

Verification demands both the verified architecture and the rules (policies) to be checked. Our approach consists of formalizing organizational policies to enable an architecture checker to compute violations of those policies in ArchiMate models.

We found that most of the time is spent in formulating the constraints, because policies are often too ambiguous to compute violations. This experience corresponds with findings in related research.

We have also found that constraints are largely organization specific, because they constrain ArchiMate models. So, what we measured were modeling conventions rather than corporate policies. Architects were needed to interpret policies and produce constraints to be checked. This produces organization specific constraints, so we cannot claim that this set of constraints is reusable over an entire domain (e.g. “Hospitals).

We expect that our tool will prompt enterprise architects to think about their modeling conventions, which has the obvious benefit of yielding more consistent models. Further research is needed to see how architects work with such a tool. A question is whether architects will benefit enough to invest time in learning how to formalize policies.

Our research is based on the EA models and design policies from one single organization. Further work is needed to find ways to reuse constraints in other organizations. We expect that this approach works similarly for other domains than healthcare, but more experience is needed to confirm that.

Acknowledgement

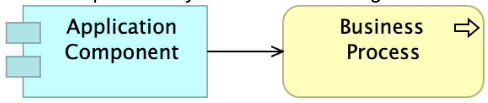
The authors thank the Utrecht Medical Center (UMC) for access to documents and support of our research.

References

1. 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, September).
2. ArchiMate® 3.1 Specification © The Open Group. (2012-2019)
3. Chapurlat, V., & Braesch, C. Verification, verification, qualification and certification of enterprise models: Statements and opportunities. Computers in Industry, 59(7), 711-721. (2008).
4. Kharlamov, E., Grau, B. C., Jiménez-Ruiz, E., Lamparter, S., Mehdi, G., Ringsquandl, M., & Horrocks, Capturing industrial information models with ontologies and constraints. In *International Semantic Web Conference* (pp. 325-343). Springer, Cham. (2016, October).
5. Marosin, D., Ghanavati, S., & Van Der Linden, D. A Principle-based Goal-oriented Requirements Language (GRL) for Enterprise Architecture. In *iStar* (2014).
6. Babkin, E. A., & Ponomarev, N. O. Analysis of the consistency of enterprise architecture models using formal verification methods. *Business Informatics*, (3), 30-40. (2017).
7. 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, September).
8. Joosten, S. Relation Algebra as a programming language using the Ampersand compiler. *Journal of Logical and Algebraic Methods in Programming*, *100*, 113-129. (2018).
9. Filet, P., van de Wetering, R., & Joosten, S. ENTERPRISE ARCHITECTURE
10. ALIGNMENT. (2019).
11. Beauvoir, P. jArchi . https://github.com/archimatetool/archi-scripting-plugin/wiki. (2019).
12. Beauvoir, P. Archi, archimate modelling tool. http://www.archimatetool.com/. (2018).
13. Korman, M., Lagerström, R., & Ekstedt, M. Modeling enterprise authorization: a unified metamodel and initial verification. *Complex Systems Informatics and Modeling Quarterly*, (7), 1-24. (2016).
14. 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).
15. Zhi, Q., Yamamoto, S., & Morisaki, S. IMSA-Intra Model Security Assurance. *J. Internet Serv. Inf. Secur.*, *8*(2), 18-32. (2018).
16. Blanco-Lainé G., Sottet J-S, Dupuy-Chessa SUsing an enterprise architecture model for GDPR compliance principles. 12th IFIP Conference on Practice of Enterprise Modeling, POEM'2019, Nov 2020, Luxembourg, Luxembourg. hal-02482761. (2020).
17. ZiRA. https://sites.google.com/site/zirawiki/ (2019).
18. Lankhorst M. M., Proper, H. A. and H. Jonkers H. The Anatomy of the ArchiMate Language. IJISMD, 1(1):1-32. (2010).
19. ArchiMate® Model Exchange File Format: Schema Documentation for ArchiMate® 2.1 (2015)
20. Binding; Document Number: C154A; Published by The Open Group, (August 2015).
21. Archi@ Version 4.6.0. Archi User Guide. (2019)
22. Tepandi, Jaak, et al. "Towards a cross-border reference architecture for the once-only principle in Europe: An enterprise modelling approach." *IFIP Working Conference on the Practice of Enterprise Modeling*. Springer, Cham, (2019).

Appendix. Formalization and ArchiChecks of polices 3-10.

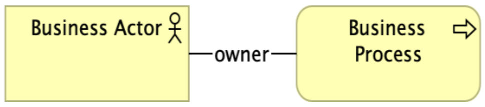
Policy 3: Each business process should be served by at least one application system

Reformu- lated in natu- ral language	Each business process is served by at least one application component.
ArchiMate Practice	For every business process, there is an application component that serves the business process by means of a serving relation. 
Predicate Logic	$\forall b \in BusinessProcess\ \exists c \in ApplicationComponent :\ c\ serving\ b$

Ampersand
language for
Ar-
chiChecker
	`RULE "Policy 3":`
	`  I[ApplicationComponent]`
	`  |- serving ; serving[ApplicationComponent*BusinessPro-`
	`cess] ; serving~`
	`VIOLATION (TXT "Application component \'", TGT name, TXT`
	`"\' is not serving a Business process.")`

Violations from Ar- chichecker	*There are 12 violations of RULE "Policy 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 'Tenant management Office 365' is not serving a Business Process.* *Application Component 'App-V/ThinApp' 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 'Azure AD' is not serving a Business Process.* *Application Component 'National Exchange Point ' is not serving a Business Process.* *Application Component 'SCCM' is not serving a Busi-ness Process.* *... (2 more)*
Comment	Among the violations mentioned above, 4 are about figure 6 (case: pharmacy) and 6 violations are about figure 7 (case: office 365). In both models (Figure 6, Figure 7) there are 12 application components that are not serving any business process. The conclusion is equal to policy 1. In this case all the applications do not serve any business process. It does not mean that the included rule for checking on the EA model does not constitute a violation. The architects have not chosen to visualize the rule in both mo-dels.

Policy 4. Every business process has precisely one owner.

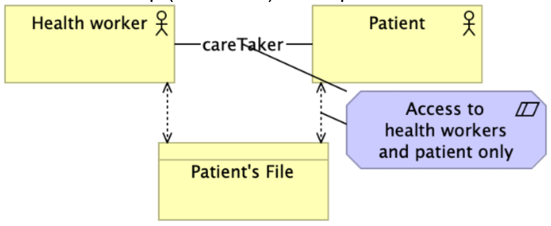
Reformulated in natural language	Every business process must have precisely one owner.
ArchiMate Practice	Every business actor relates to precisely one business process by means of an association relation called “owner”. 
Predicate Logic	(Policy 4.1) $\forall s \in BusinessProcess\ \exists a \in BusinessActor : a\ owner\ s$ (Policy 4.2) $\forall s \in BusinessProcess\ \forall a,\ b \in BusinessActor : a\ owner\ s \land b\ owner\ s \Rightarrow a = b$

Ampersand language for ArchiChecker	`RULE "Policy 4.1":`
	`  I [BusinessProcess] |- owner[BusinessActor*BusinessPro-cess]~ ; owner[BusinessActor*BusinessProcess]`
	`VIOLATION (TXT "Business Process \'", SRC name, TXT "\' does not have an owner.")`
	
	`RULE "Policy 4.2":`
	`  owner[BusinessActor*BusinessProcess] |- -(-I[Busi-nessActor] ; owner)`
	`VIOLATION (TXT "Business Process \'", TGT name, TXT "\' has multiple owners.")`

Violations from Ar-chichecker	*There are 18 violations of RULE "Policy 4.1":* *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 'BSN and COV check insurance data' 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)*

Comment	Policy 4.1: To visualize an owner for each business process, ensure that the models are displayed extensively. To a certain extent, it is assumed that an owner has been assigned for each business process. In the case of PSA CS Pharmacy (Figure 6) a role / owner has been assigned for a crucial process, namely ‘medication monitoring’. Policy 4.2: This script of ArchiMate contains no violations

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

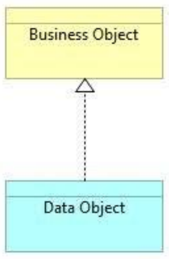
Reformu-lated in nat-ural lan-guage	For every patient, there must be one file called “Patient’s File”. Each patient must have access to his or her patient file. Each health worker directly involved in medical care for a patient must have access to that patient’s file. Others have no access to this file.
ArchiMate Practice	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.  Health worker careTaker Patient Access to health workers and patient only Patient's File
Predicate Logic	Let be $patient$ 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 to their own patients. $\forall a \in BusinessActor\ \exists o \in BusinessObject$: $patient\{a\} \Rightarrow patientfile\{o\} \wedge a\ access\ o \wedge o = fileof(a)$ $\forall a,\ b \in BusinessActor\ \forall o \in BusinessObject$: $a\ caretaker\ b \wedge b\ access\ o \Rightarrow a\ access\ o$

Ampersand language for Ar-chiChecker	`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[Pa-tient*PatientFile]~`
	`careTaker;access = access[Patient*PatientFile]~`
	
	`RULE "Policy 5":I [BusinessObject] |- access [BusinessOb-ject*BusinessActor]; access [BusinessObject*Busi-nessActor]~`
	
	`MEANING "If (a,b) is in the relation caretaker, then per-son a is a health worker directly involved in medical care for patient b."`
	`VIOLATION ( TXT "PatientFile (Business Object) \'", SRC name, TXT "\' is not accessed by a caretaker/Patient (Health worker")`

Violations from Ar-chichecker	RELATION access[BusinessObject*BusinessActor] There is one violation of RULE "Policy 5": PatientFile (Business Object) 'Patient file ' is not accessed by a care-taker/Patient (Health worker)
Comment	In Figure 6, a violation has been detected by ArchiChecker. This is because of the omitted visualization that the caretaker has access to the patient file. The architecture was visualised to replace the existing pharmacy system, so that the visualization of the access of a caretaker to the file of his patient was not important.

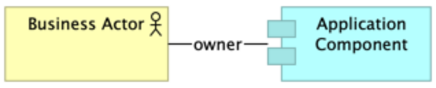
Policy 6: A data or data group uses one or more business objects.

Reformulated in natural language	Every Data Object realizes one or more Business Objects
ArchiMate Practice	
Predicate Logic	$\forall b \in \ DataObject\ \exists c \in \ BusinessObject : \ c\ realization\ b$

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

Violations from Archichecker	`This script of ArchiMate containts no violations`
Comment	There is just one data object, 'Patient data', which resides in Figure 6. Since it is connected to Business object 'Patient file' by a realization relation, there are no violations of this policy.

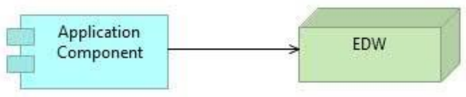
Policy 7. The continuity of critical systems at the Medical Centre is guaranteed

Reformulated in natural language	Each IT system has an owner. The owner is responsible for ensuring continuity and taking adequate measures.
ArchiMate Practice	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 don't want multiple owners for one application component. 
Predicate Logic	$\forall a \in ApplicationComponent\ \exists b \in BusinessActor : b\ owner\ a$

Ampersand language for ArchiChecker	`RULE "Policy 7.1":`
	`  I [ApplicationComponent] |- owner[BusinessActor*ApplicationComponent]~ ; owner[BusinessActor*ApplicationComponent]`
	`VIOLATION (TXT "Application Component \'", SRC name, TXT "\' does not have an owner.")`
	`RULE "Policy 7.2":`
	`  owner[BusinessActor*ApplicationComponent] |- -(-I[BusinessActor] ; owner)`
	`VIOLATION (TXT "Application Component \'", TGT name, TXT "\' has ", SRC name, TXT " an owners.")`

Violations from Ar-chichecker	*There are 12 violations of RULE "Policy 7.1":* *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 'Tenant management Office 365' does not have an owner.* *Application Component 'App-V/ThinApp' does not have an owner.* *Application Component 'MIRA (CGM Pharmacy)' does not have an owner.* *Application Component 'ServiLocker' 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.* *... (2 more)*
Comment	Policy 7.2: This script of ArchiMate contains no type errors. The figure 6 and 7 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 be-ing modelled.

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

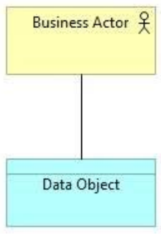
Reformulated in natural language	Enable integrated data and information provision on various domains.
ArchiMate Prac-tice	
Predicate Logic	$\forall \ \in A\ \exists c \in\ Node:\ c\ serving\ b$

Ampersand lan-guage for Ar-chiChecker	`RULE "Policy 8": I[ApplicationComponent] |- serving [ApplicationComponent*Node]; serving [ApplicationCom-ponent*Node]~`
	`VIOLATION (TXT "Application MIRA\'", SRC name, TXT "\'has no access with the Data Warehouse")`

Violations from Archichecker	`There are 12 violations of RULE "Policy 8":`
	`    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 '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 ac-cess with the Data Warehouse`
	`    Application 'SCCM' has no access with the Data Warehouse`
	`      ... (2 more)`

Comment	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.

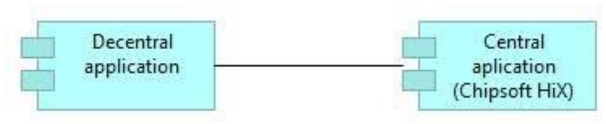
Policy 9 - Every data and data type has someone responsible.

Reformulated in natural language	Every data object must have precisely one owner.
ArchiMate Practice	For each data group / data domain, an administrator has been appointed who is (ultimately) responsible for definitions and content. 
Predicate Logic	$\forall b \in \mathit{DataObject}\ \exists c \in \mathit{BusinessActor} : c\ \mathit{association}\ b$

Ampersand language for ArchiChecker	`RULE "Policy 9": I [DataObject] |- association [DataObject*BusinessRole]; association [DataOb-ject*BusinessRole]~`
	`VIOLATION (TXT "DataObject\'", SRC name, TXT "\'a data object has no responsible")`

Violations	*There is one violation of RULE "Policy 9":* *DataObject' Patient data' a data object has no responsible*
Comment	The visualized data object in figure 6 has no responsible.

Policy 10 - Use of central application is mandatory.

Reformulated in natural language	The use of preferred central applications is mandatory.
ArchiMate Practice	
Predicate Logic	$\forall b \in ApplicationComponent\ \exists c \in\ ApplicationComponent:\ c\ association\ b$
Ampersand language for ArchiChecker	`RULE "Policy 10": I [ApplicationComponent] |- association [ApplicationComponent*ApplicationComponent]; association [ApplicationComponent*ApplicationComponent]~ VIOLATION (TXT "Application\'", SRC name, TXT "\'is not connected to the central application")`

Violations	`There are 7 violations of RULE "Policy 10":`
	`Application'Azure AD'is not connected to the central application`
	`Application'Autopharma 'is not connected to the central application`
	`Application'App-V/ThinApp'is not connected to the central application`
	`Application'Active Directory local'is not connected to the central application`
	`Application'SCCM'is not connected to the central application`
	`Application'ServiLocker'is not connected to the central application`
	`Application'Tenant management Office 365'is not connected to the central application`

Comment	For the applications in both figures there is no connection visualized with the central application Chipsoft HiX.