MASTER'S THESIS

Development of a Method for Consistent Enterprise Modelling in ArchiMate

Severin, Sefanja

Award date:
2022

Development of a Method for Consistent Enterprise Modelling in ArchiMate

Opleiding:	Open Universiteit, faculteit Betawetenschappen
	Masteropleiding Business Process Management & IT
Degree programme:	Open University of the Netherlands, Faculty Science
	Master of Science Business Process Management & IT
Course:	IM0602 BPMIT Graduation Assignment Preparation IM9806 Business Process Management and IT Graduation Assignment
Student:	Sefanja
Severin Identification number:	
Date:	7 June 2022
Thesis supervisor	dr. E. Roubtsova
Second reader	dr. B. Roelens
Version number:	1.3
Status:	final

Abstract

ArchiMate is an enterprise modelling language that is designed to support consistency checking. However, it does not provide guidelines on how to create consistent models. By taking inspiration from goal-oriented requirements engineering (GORE) methods such as KAOS and ExtREME, and by thoroughly analysing the semantics of the ArchiMate metamodel, we have formulated the semantic relationships between three ArchiMate viewpoints as consistency rules and guidelines. They clarify the undefined justification of sub-models by goals in KAOS. Based on these consistency rules and guidelines, a method to create consistent multi-view models in ArchiMate has been proposed. This method has been evaluated by modelling an insurance case. The resulting set of views complies with our consistency requirements. The method can be used by ArchiMate practitioners and inspire researchers to further develop formal languages for the automatic validation of ArchiMate models. As an aside, our research shows that the goal refinement relationship is missing in ArchiMate.

Key terms

ArchiMate, multi-view consistency, enterprise modelling, goal-oriented requirements engineering (GORE), KAOS, ExtREME.

Acknowledgements

I wish to extend my special thanks to my supervisor, dr. E. Roubtsova, who guided me throughout this project and whose articles and insights were a welcome source of inspiration. To express the feeling that this thesis is a team result, I have used the plural form.

Contents

1. Introduction

1.1. Background

An enterprise model is an abstract representation of an organization. It consists of a set of related views, where each view describes the organization from a different perspective and in its own language. It is essential that these views are consistent, or else these different perspectives are in fact different truths. This would undermine the purpose of the enterprise model, namely to describe the organization as a coherent whole.

ArchiMate (The Open Group, 2019) is an enterprise modelling language. An ArchiMate model can, and typically will, consist of multiple views. Thorough study of the ArchiMate specification and guiding literature shows that ArchiMate does not offer any guidelines or rules that ensure consistency between views.

Our goal is to discover what elements from other multi-view modelling approaches can be applied to ensure consistency between ArchiMate views and how the modelling tool Archi (Beauvoir, Sarrodie, & The Open Group, 2021) can help to ensure consistency.

1.2. ArchiMate modelling language

ArchiMate is a visual enterprise architecture modelling language based on an abstract syntax. For the definition of architecture, ArchiMate refers the reader to the TOGAF framework which defines architecture as 'the structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time' (The Open Group, 2018).

In this work we conform to ArchiMate terminology. Table 1 lists the ArchiMate terms that are used most often in this work.

Table 1. ArchiMate terms used most often in this work. From ArchiMate 3.1 Specification*, by The Open Group, 2019.*

Term	Definition
Architecture view	A representation of a system from the perspective of a related set of concerns.
Architecture viewpoint	A specification of the conventions for a particular kind of architecture view.
Aspect	Classification of elements based on layer-independent characteristics related to the concerns of different stakeholders. Used for positioning elements in the ArchiMate metamodel.
Concept	Either an element, a relationship, or a relationship connector.
Element	Basic unit in the ArchiMate metamodel. Used to define and describe the constituent parts of Enterprise Architectures and their unique set of characteristics.
Layer	An abstraction of the ArchiMate framework at which an enterprise can be modelled.
Model	A collection of concepts in the context of the ArchiMate language structure.
Relationship	A connection between a source and target concept. Classified as structural, dependency, dynamic, or other.

Note that the *concept* and its specializations are defined ambiguously: they can be part of a model or the ArchiMate metamodel. To distinguish between the two, we will use *concept type* to refer to a member of the metamodel and *concept instance* to refer to a member of a particular model. We will abbreviate this to *concept* if the meaning can be derived from the context.

In general, an element belongs to a certain cell of the ArchiMate framework; for example, a cell of intersection of *business* and *passive structure*, or *application* and *behaviour*, see Figure 1. Relationships can occur in any cell and between cells since they are 'overloaded': their exact meanings depend on the elements they connect.

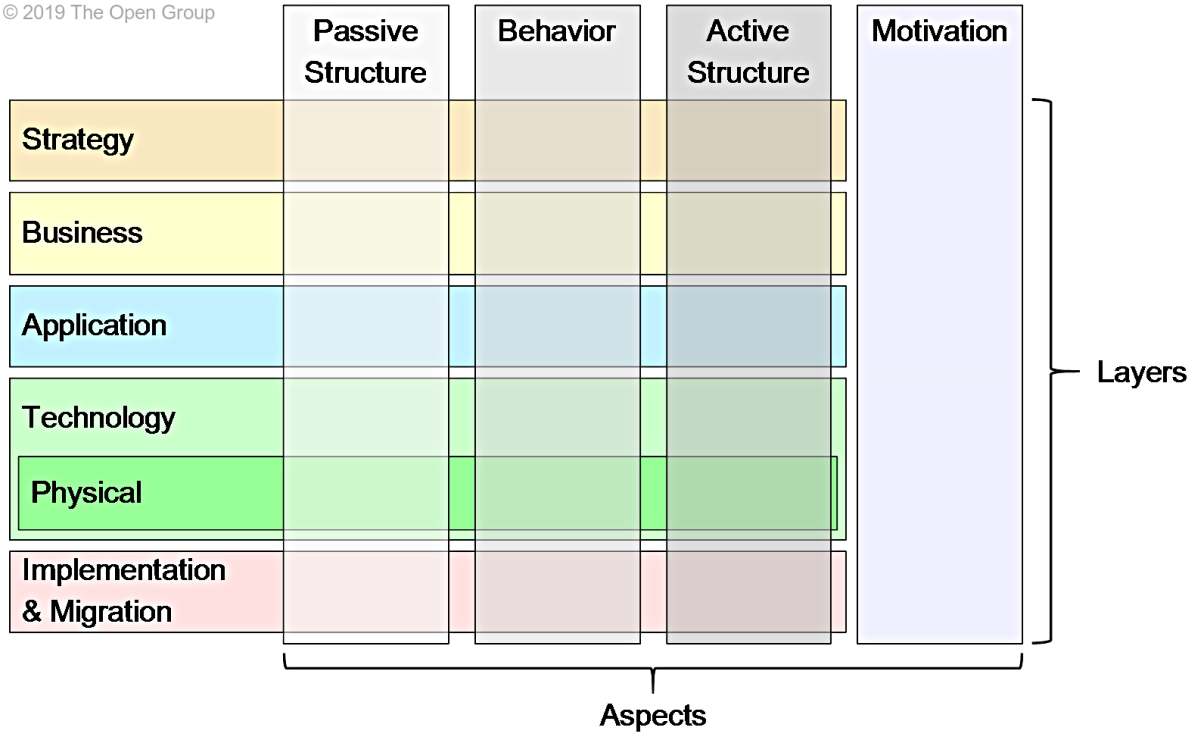


Figure 1. ArchiMate full framework. From ArchiMate 3.1 Specification*, by The Open Group, 2019.*

1.3. Problem statement

We investigate the sources of inconsistencies in ArchiMate. To illustrate this, we create a collection of ArchiMate views in Archi (Beauvoir, Sarrodie, & The Open Group, 2021), based on a case called 'Preparation of a document by several participants' (Roubtsova, Interactive Modeling and Simulation in Business System Design, 2016) with the following description:

> Let us consider a system that controls a preparation of a document (a proposal a paper or a report) by several participants. One of the participants usually plays the coordinator role. The coordinator is responsible for submitting the document. There is a deadline for the document submission. The coordinator creates the parts of the document and chooses participants. Each part is assigned to a participant. A part has its own deadline before the deadline of the document and should be submitted by the participant, so that the coordinator has the time to combine parts and submit the document. If a participant misses the deadline of his part, the coordinator sends a reminder to the delaying participant. The coordinator can change the deadline or assign the part to another participant. Only the coordinator can cancel the preparation of the document. (pp. 117-118)

The goal model of the case is shown in Figure 2.

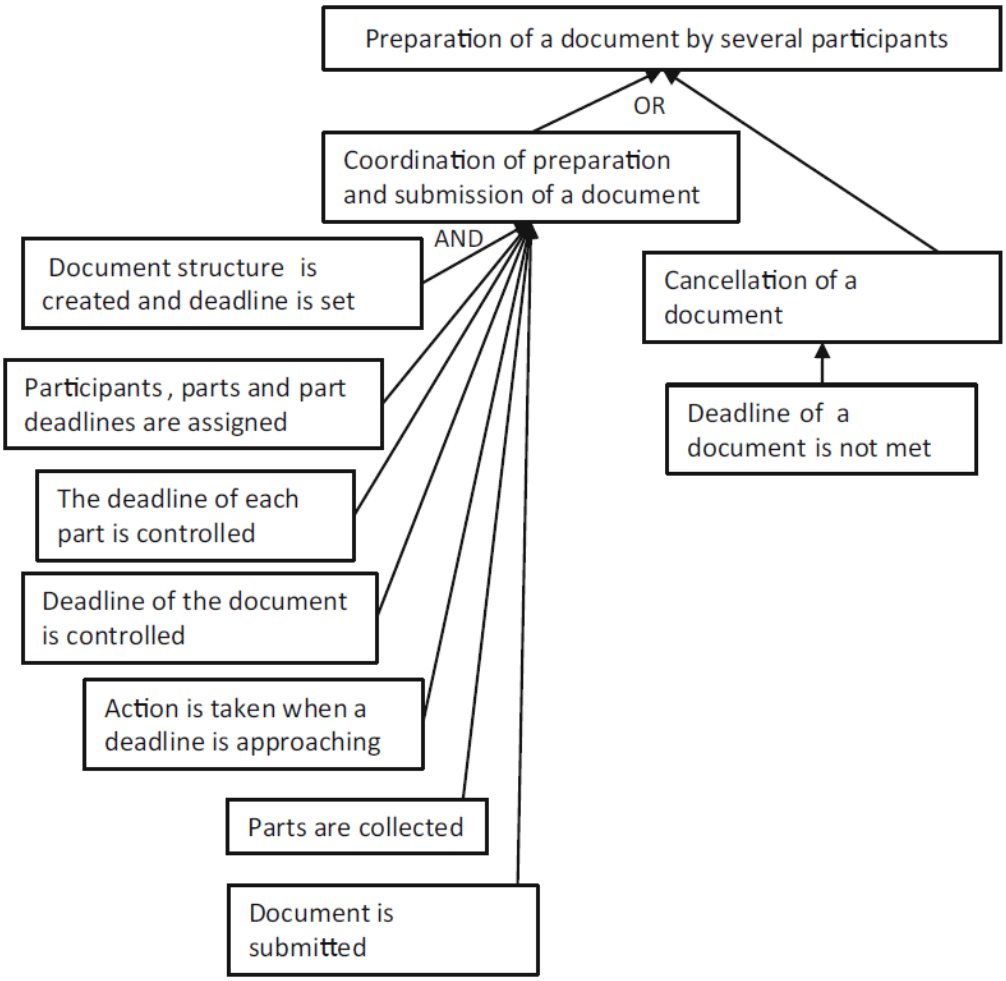


Figure 2. Goals and requirements for the case Preparation of a document by several participants. From Interactive Modeling and Simulation in Business System Design*, by E. Roubtsova, 2016, Springer International Publishing.*

Figure 3 shows the goal model translated to ArchiMate following the example in Figure 2. In a later section we will discuss the correct use of elements and relationships in ArchiMate. The focus point for now is consistency between views.

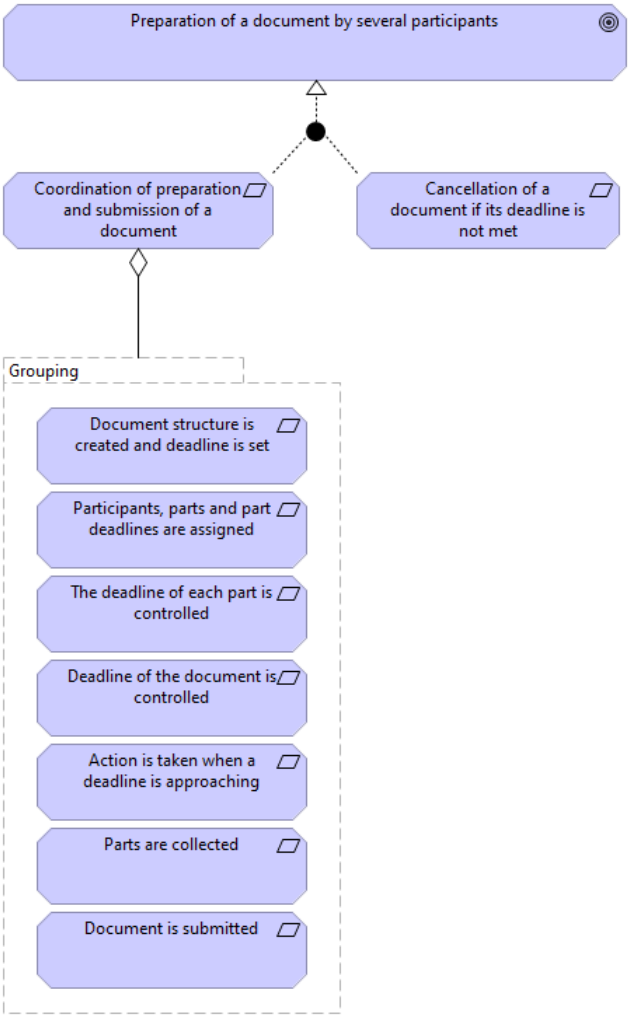


Figure 3. A goal realization view

Figure 4 shows the same goal view but expanded to include the structural view on the left, which in turn is expanded into a behavioural view on the right.

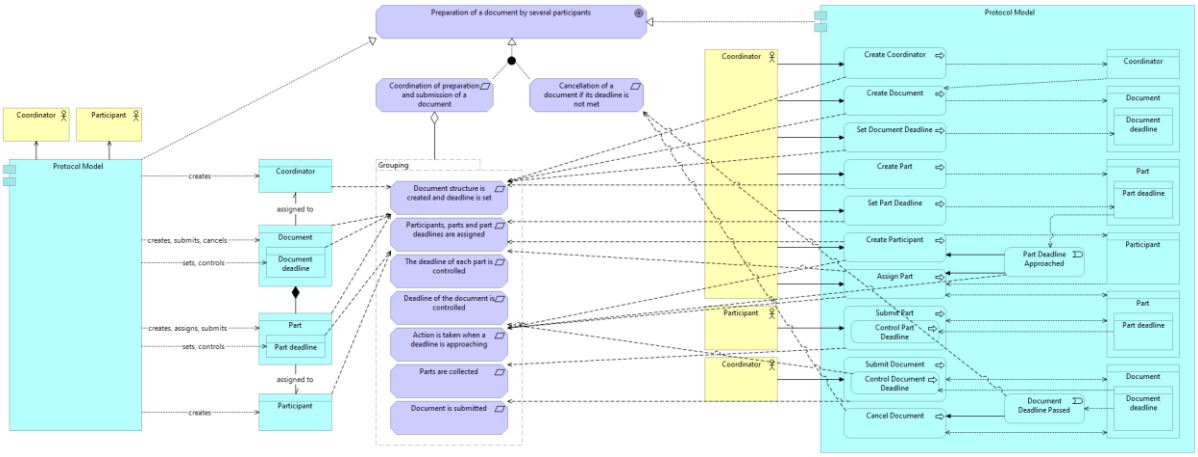


Figure 4. A goal view expanded with structural and behavioural elements

As one can see, such a complete overview is hard to read, even with few elements. We therefore need to split them up into multiple views, as shown in Figure 5.

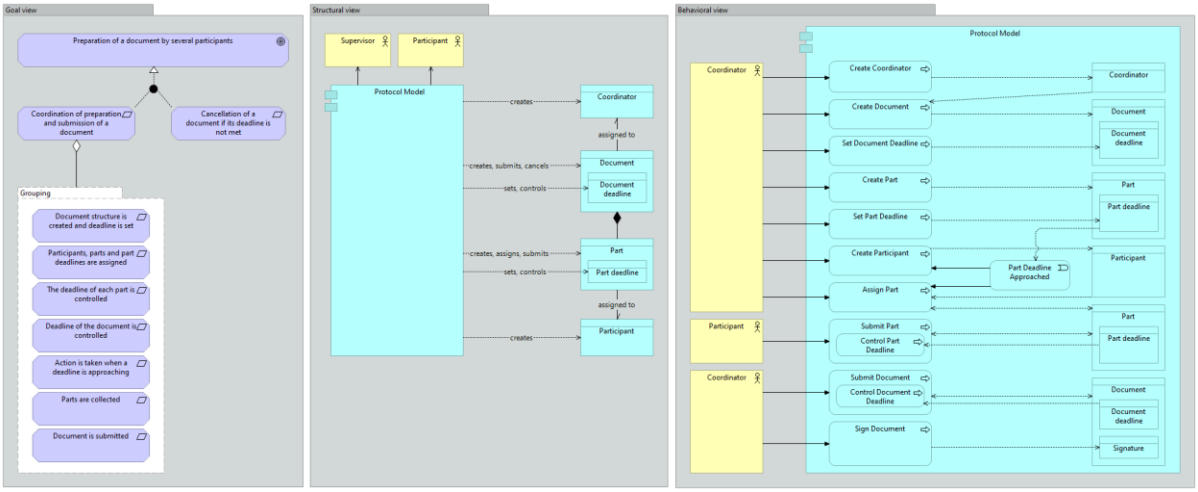


Figure 5. A goal, structural and behavioural view, side by side

But by splitting up the views we have created a new problem: we cannot readily see how these views are related, which makes it difficult to check their consistency. Using Figure 5 as our source of inspiration, we can identify several sources of inconsistency:

- In the structural view the synonym *Supervisor* has been used instead of *Coordinator*.
- The structural view contains a relation to *Document* labelled *cancels*, but this activity is not modelled in the behavioural view.
- The behavioural view contains the element *Sign document*, which cannot be justified by the requirements in the goal model.
- In the structural view, *Part deadline* has been misspelled as *Part daedline*.
- The behavioural view contains the structural element *Signature* which cannot be found in the structural view.

To support consistency checking, the ArchiMate metamodel is based on an abstract syntax (Lankhorst, Proper, & Jonkers, 2009): the same element can appear on multiple views, but there is only one instance of it in the model. This allows modelling tools to perform consistency checks between views. For example, they can check if every element in one view is also present in another view.

In addition, ArchiMate specifies relationships that may exist between the different cells of its framework. For example, relations can be made between elements belonging to the business layer and those belonging to the application layer of the framework. This allows an analyst or tool to check, for example, if every application component supports at least one business process.

Although ArchiMate has some properties designed to support consistency checking, the standard does not provide any guidelines to ensure consistency. It does not give directions on how to build consistent views or on what kind of consistency rules to use. Also, ArchiMate does not have a language in which to express consistency rules. To illustrate this point, Babkin and Ponomarev (2017) had to transform their ArchiMate model to the language of the MIT Alloy Analyzer system before they could verify the model, using their own consistency rules.

There are other multi-view modelling approaches that do offer rules or guidelines to deal with consistency.

A first example is the **4+1** architectural view model (Kruchten, 1995). This model identifies five views and describes the relations between those views. Each view can have its own modelling language and style. To support consistency, related semantic points between the views are identified. In 4+1, a special place is reserved for the view that contains the scenarios which are 'in some sense an abstraction of the most important requirements' (Kruchten, 1995). The scenarios view is used to drive and validate the other views. Perhaps we could use the motivation elements in ArchiMate in a similar manner.

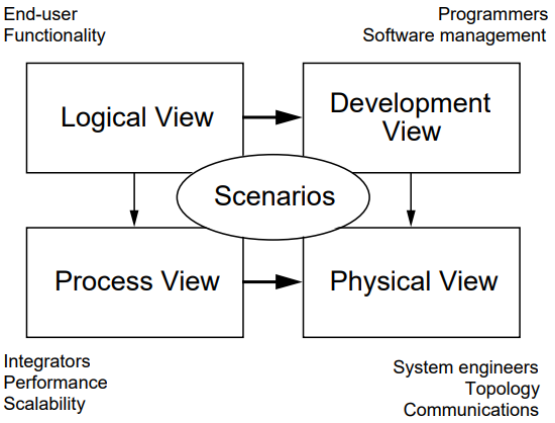


Figure 6. The "4+1" model. From Architectural blueprints—the "4+ 1" view model of software architecture*, by P. Kruchten, 1995, IEEE software, 12(6).*

A second example is the **4EM** (For Enterprise Modelling) method (Sandkuhl, Stirna, Persson, & Wißotzki, 2014). This method defines sub-models and the allowed relations between them, see Figure 7. It also gives guidelines to integrate these sub-models, for example: 'Every process must be related to at least one ARM role, which is responsible for or performs that process.' We should be able to reuse these guidelines in ArchiMate. In general, Sandkuhl et al. (2014) state that 'models should complement each other' and that 'inter-model links should establish a clear line of reasoning'. The goal model in 4EM has a similar function as the scenarios view in 4+1 as it 'describes essentially the reason, or motivation, for components in the other sub-models' (Sandkuhl, Stirna, Persson, & Wißotzki, 2014).

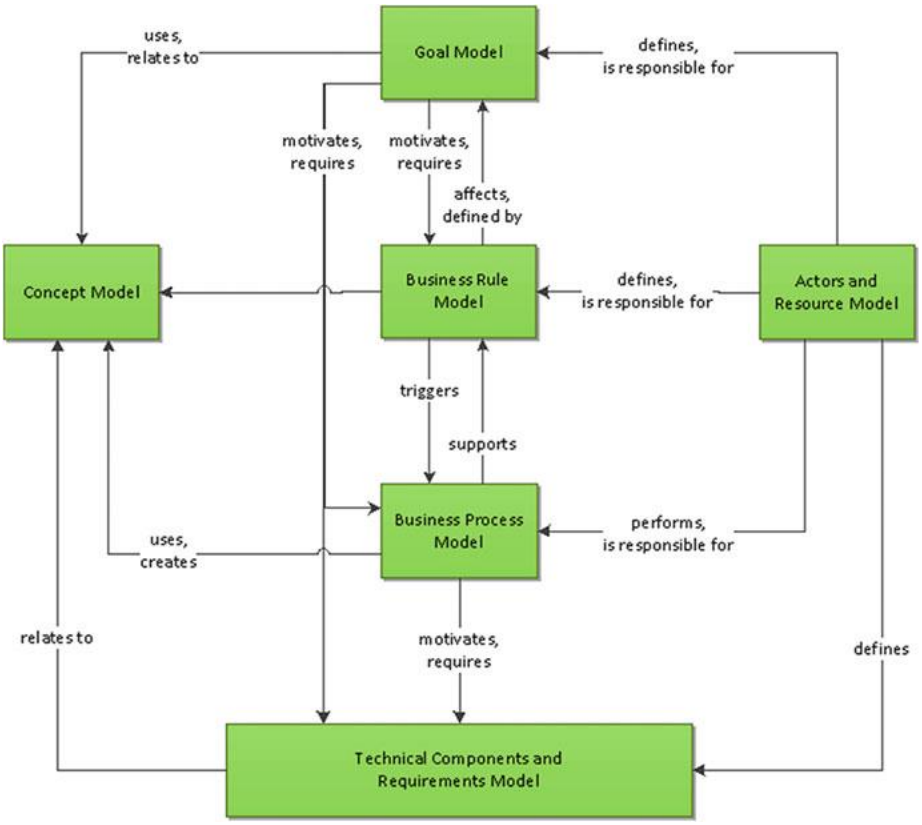


Figure 7. Sub-models of the 4EM approach and their relationships. From Enterprise modeling*, by K. Sandkuhl, J. Stirna, A. Persson and M. Wißotzki, 2014, Springer.*

A third example is the **ExtREME** (EXecuTable Requirements Management and Evolution) method (Roubtsova, Interactive Modeling and Simulation in Business System Design, 2016). This method defines two models: a goal model and a protocol model. The protocol model contains both structural and behavioural elements. Managing consistency between the two models is part of the core of this method. This is done by first refining the goal model to requirements such that they only contain countable or comparable nouns (the concepts). The protocol model is 'generated' from the requirements by transforming nouns into concepts and verbs into events, assisted by human interpretation of the natural language in which the requirements are expressed. Figure 8 shows an example of a protocol model that realizes the goal model in Figure 2. It seems that this method of goal refinement and relating requirements to other parts of the architecture should be applicable in ArchiMate models.

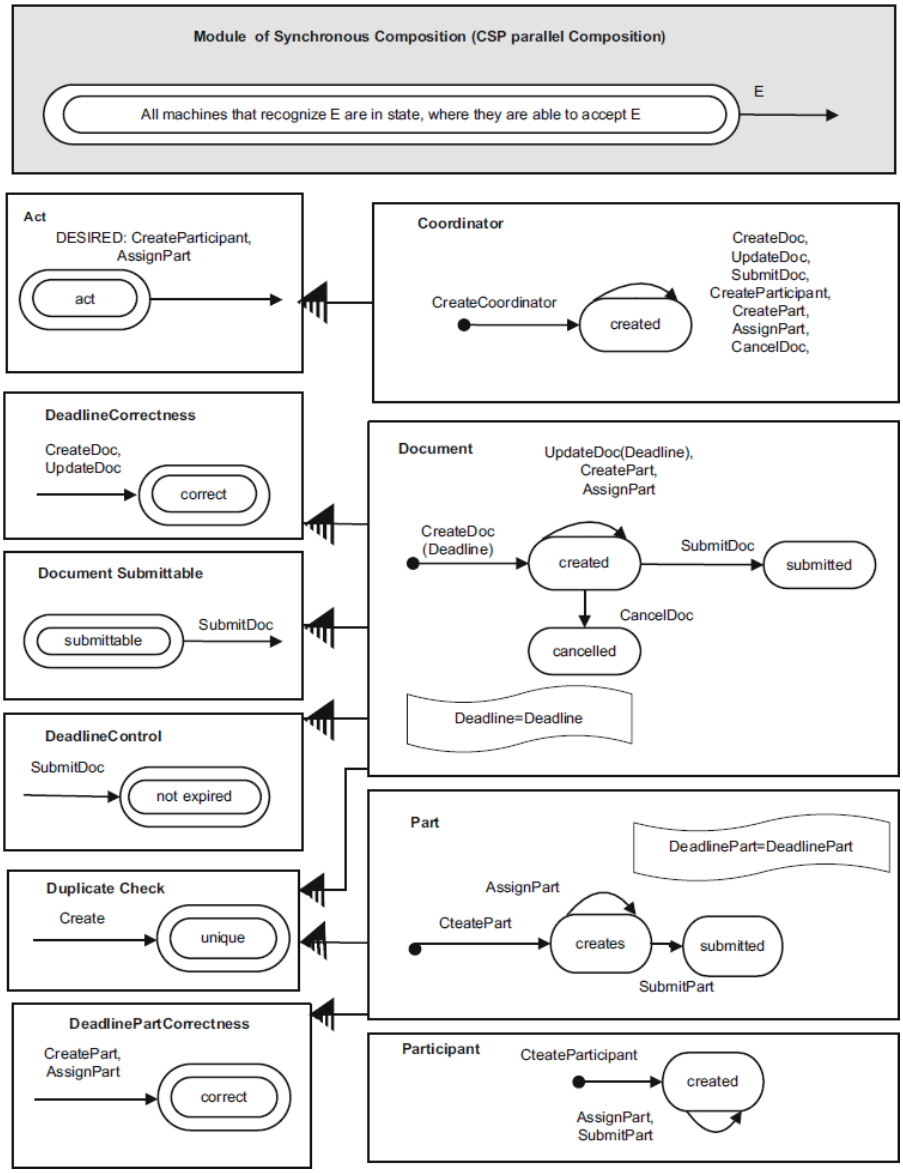


Figure 8. An ExtREME protocol model. From Interactive Modeling and Simulation in Business System Design*, by E. Roubtsova, 2016, Springer International Publishing.*

1.4. Research objective and questions

When creating a multi-view model in ArchiMate, we want all views of the model to be consistent. Our goal is to formulate a method that ensures consistency between views.

Consistency is always in reference to something. In the case of an ArchiMate model, what should be the reference point for consistency? In 4+1, 4EM and ExtREME the reference point is the view that contains the goals. This would also make sense for ArchiMate models because enterprise architecture is supposed to facilitate decision making. Although the motivation extension in ArchiMate was missing in its initial version (Cardoso, Almeida, & Guizzardi, 2010), after its addition Cardoso et al. (2010) regard the extension to be more 'sophisticated' in comparison to other enterprise modelling approaches. Positioning the goal view at the centre of an ArchiMate model would help explain the 'why' of an architecture, its contribution to the goals of an organization. In the words of the ArchiMate specification (The Open Group, 2019): 'The purpose of the motivation elements is to model the motivation behind the core elements in an Enterprise Architecture.'

If we choose the goal view as our reference point for consistency, we may turn to the modelling approaches that have inspired the motivation extension in ArchiMate and see if they contain guidelines to ensure consistency between views.

Combining all the above leads to the following research questions:

> How can we define goal-oriented multi-view consistency as a set of concrete rules and guidelines for ArchiMate models?
>
> What could be a method for use in Archi that produces goal-oriented ArchiMate models with multi-view consistency?

1.5. Motivation/relevance

ArchiMate is an enterprise architecture modelling language. Many different definitions of enterprise architecture (EA) exist, most of them founded in theory. Kotusev (2019) formulates an evidence-based definition of EA:

> EA is a collection of special documents (EA artifacts) describing various aspects of an organization from an integrated business and IT perspective intended to bridge the communication gap between business and IT stakeholders, facilitate information systems planning and thereby improve business and IT alignment.

A way to bridge the communication gap is to use multiple perspectives, each tailored to a particular group of stakeholders. KAOS, a multi-view modelling approach, states this explicitly (Respect-IT, 2007):

> Many companies have noticed that users and IT analysts most often do not understand each other very well. KAOS provides the right connection between the two worlds: users quickly feel confident with goal and responsibility models; analysts like the object and operation models. (p. 9)

To bridge the gap using a multi-view strategy, it is essential that the different views are consistent. Without multi-view consistency the proverbial bridge cannot exist. Consistency in this context means that there are rules for the unique transformation of elements found in one set of views, with its own semantics, into elements of another set of views, having different semantics.

2. Theoretical framework

2.1. Research approach

Before we can formulate a method for goal-oriented multi-view consistency in ArchiMate we first need to answer three questions, as formulated in Table 2. This table also contains the search queries for each question.

The rest of the research approach described here resembles the approach described by Webster & Watson (2002). If a search query results in many hits, we will briefly scan the title and abstract for relevance. For the relevant articles found, we will go backward to see which earlier articles they cite that may be relevant to our research. Next, we will do a forward search using the Web of Science to identify articles that cite the ones already identified. We know that our backward and forward searches are complete when no new concepts are found.

Table 2. Literature search queries

Question	Query	Comment
How does ArchiMate work?	Query 1: • Keywords: ArchiMate • Databases: ACM Digital Library, IEEE Digital Library, Electronic Journals Service (EBSCO), SpringerLink • Author: Lankhorst	Marc Lankhorst was a key developer of ArchiMate. Articles by other authors will be identified during the backward and forward search.
Which approaches have inspired ArchiMate's motivation extension and how do they work?	Query 2: • Keywords: ArchiMate AND GORE • Databases: ACM Digital Library, IEEE Digital Library, Electronic Journals Service (EBSCO), SpringerLink	GORE is an acronym for goal-oriented requirements engineering.
How do the identified approaches ensure multi-view consistency?	Query 3: • Backward search through the literature identified in the other queries	

2.2. Implementation

Table 3 lists the literature search results for each query identified in Table 2.

Table 3. Literature search results

Query	Hits	Relevant hits + backward + forward	Literature used	Total used
1	6	3 + 2 + 0	• (Lankhorst, Proper, & Jonkers, 2009) (The Open Group, 2019)	2
2	28	3 + 4 + 0	• (Engelsman, Quartel, Jonkers, & van Sinderen, 2011) • (Respect-IT, 2007) • (The Object Management Group, 2015) (Yu & Mylopoulos, 1994)	4
3	2	2 + 0 + 2	• (Dijkman, Quartel, & Van Sinderen, 2006) (Nuseibeh, Kramer, & Finkelstein, 1994)	2

2.3. Results and discussion of the literature review

In section 1.2-1.4 the ArchiMate language has been described and the need for goal-orientated modelling has been identified.

The key concepts of GORE (goal-oriented requirements engineering), such as *goal* and *requirement*, were not part of the initial version of ArchiMate, but have been added later as part of its motivation extension. The precursor to this extension of ArchiMate is ARMOR (Engelsman, Quartel, Jonkers, & van Sinderen, 2011). The metamodel in Figure 9 shows how ARMOR extends the ArchiMate metamodel. Concepts that belong to the ArchiMate metamodel have no fill colour. An open arrow denotes a specialization relationship.

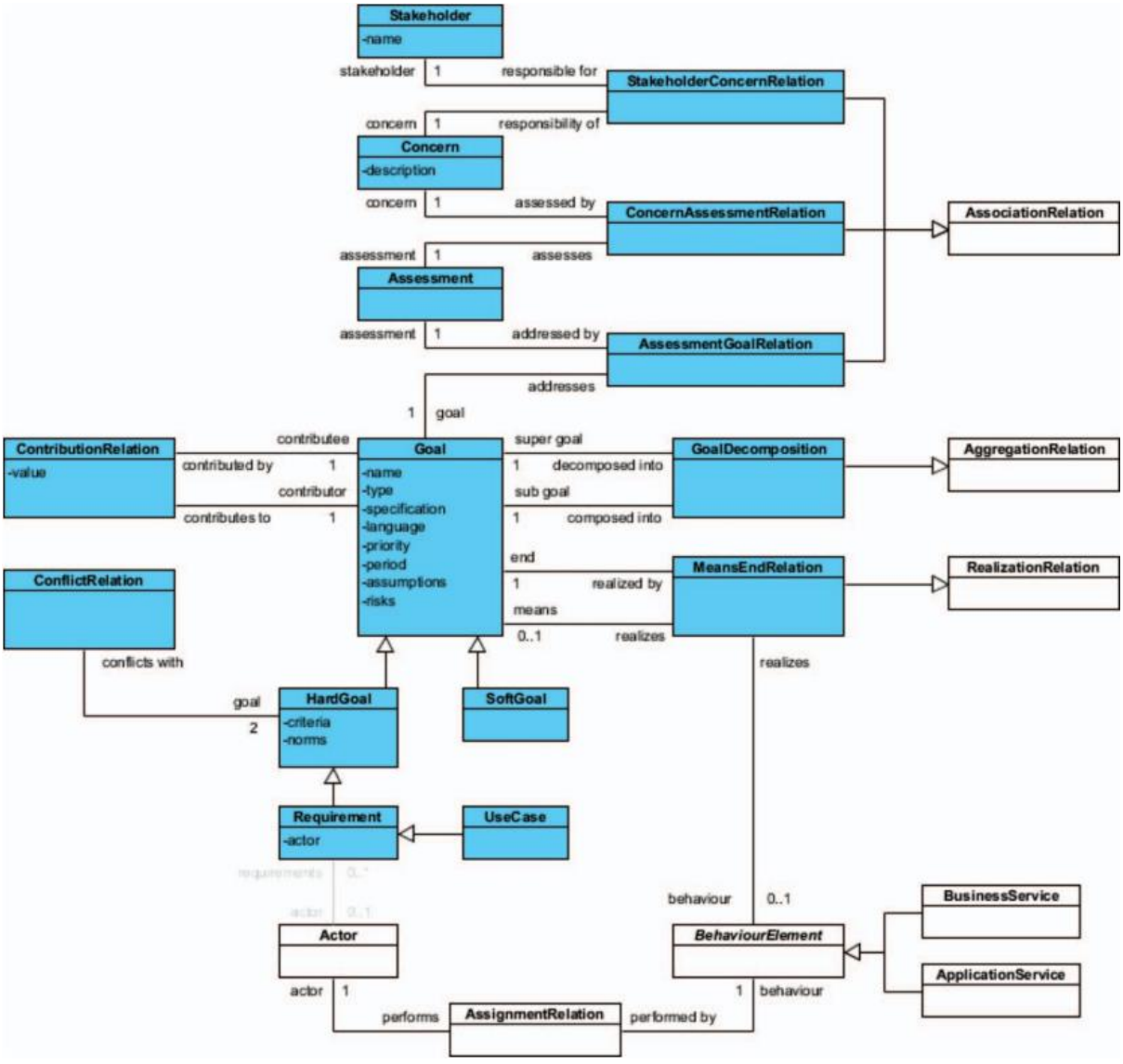


Figure 9. ARMOR metamodel. From Extending enterprise architecture modelling with business goals and requirements*, by W. Engelsman, D. Quartel, H. Jonkers & M. van Sinderen, 2011.*

ARMOR has been inspired by BMM (Business Motivation Model) (The Object Management Group, 2015), i* (Yu & Mylopoulos, 1994) and KAOS (Respect-IT, 2007). Among these approaches only KAOS contains multi-view consistency rules and guidelines.

KAOS is a method for requirements engineering that originated from a cooperation between two universities in 1990. It is a multi-view modelling approach. Figure 10 shows the KAOS metamodel, with four different types of interrelated viewpoints. Although KAOS is a GORE method, it includes notions beyond the scope of goal modelling, such as 'objects' and 'operations'. This makes KAOS a suitable candidate for our purpose, namely to find a method for creating consistent goal-oriented multi-view models in ArchiMate.

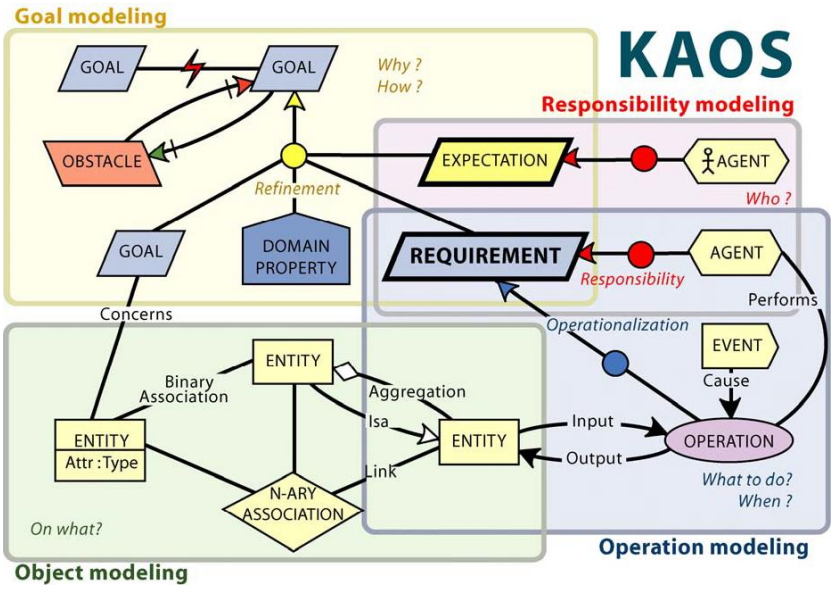


Figure 10. KAOS metamodel. From A KAOS Tutorial, V1.0*, by Respect-IT, 2017.*

KAOS and ArchiMate use slightly different terms and definitions to denote models, views, and viewpoints. In KAOS the word *model* can refer to both an abstract model (independent from representation in diagrams and documents) and a set of interrelated diagrams (such as 'the goal model'). In ArchiMate the word *model* is only used to refer to an abstract model. The notion of a set of interrelated diagrams in KAOS can be compared to a set of views in ArchiMate that share the same viewpoint. For ArchiMate terms and their definitions, see Table 1 in section 1.2.

ARMOR does not provide a method for multi-view consistency but references an existing framework with partly the same authors (Dijkman, Quartel, & Van Sinderen, 2006). The authors distinguish between three levels of expressiveness of viewpoint relations and three levels of conceptual support, see Figure 11. Because concepts in ArchiMate cannot be generalized, only specialized, there is no room for improvement on the *conceptual support* axis. As can be seen in Figure 11, ArchiMate supports relations between viewpoints, but does not offer consistency rules or guidelines. A guideline is defined informally, without automated support, while a rule is defined formally and can be automated.

The goal of our research is to identify a set of consistency rules and guidelines and a method for producing ArchiMate models that conform to those rules and guidelines.

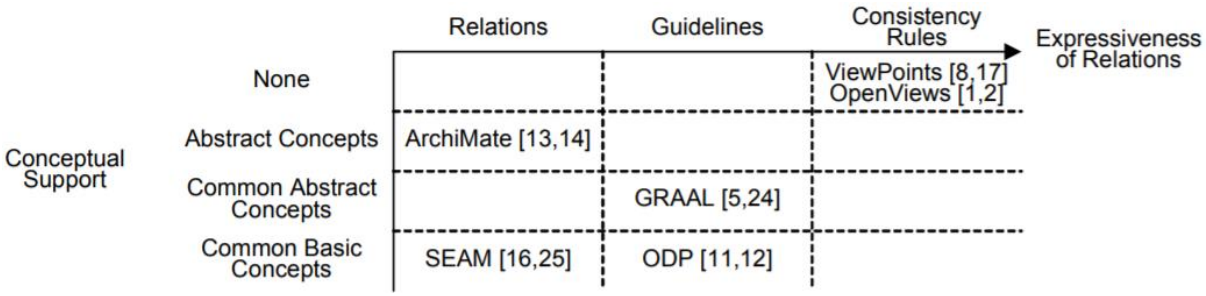


Figure 11. Existing frameworks and their support for consistency checks. From Consistency in multi-viewpoint architectural design of enterprise information systems*, by by R.M. Dijkman, D.A.C Quartel & M.J. van Sinderen, 2006.*

3. Method

The overall research objective is to formulate a goal-oriented method for ArchiMate that ensures multi-view consistency. To validate this method, we need to define consistency as a concrete set of rules and guidelines. Therefore, we have defined three objectives:

1. Formulate multi-view consistency requirements as a set of concrete rules and guidelines
2. Formulate a method for creating consistent multi-view models
3. Validate the method by modelling a case and checking the model against the consistency requirements

3.1. Conceptual design

For the first two objectives, we used an explorative approach because we wanted to discover which ideas identified in the relevant literature would be suited to be incorporated into our consistency requirements and method, without having a preconceived notion about those ideas.

For the third objective, we used a case study to test whether the formulated method would produce a consistent model for that case.

3.2. Technical design

To achieve the first two objectives, we performed an in-depth reading of the literature on ArchiMate, ARMOR, KAOS and ExtREME, identified in the Theoretical framework. We translated the KAOS metamodel to ArchiMate to use KAOS's key ideas and completeness criteria for the formulation of our consistency requirements and method. We enriched the requirements and method with ideas from ExtREME.

To achieve the third objective, we selected a case that is suitable for testing the method. In particular, the case needed to have the following properties (see section 4.1):

- Has three different goal refinement patterns: OR refinement, AND refinement, and milestone refinement
- Has only achieve goals
- Does not have contradicting goals

We selected the insurance case from Roubtsova (2016, pp. 50-58) because its goal model has already been validated by a protocol model, ensuring us that it does not contain contradicting goals. Using this case, we created a model in ArchiMate, following our own method to the letter. To validate the model, and thereby the method, we counted the number of consistency requirement violations.

3.3. Data analysis

- Data for the first objective is analysed by reading the mentioned literature and translating each KAOS concept to ArchiMate.
- Data for the second objective is analysed by iteratively formulating a method, constructing ad-hoc models based on that method and validating their conformity to the consistency requirements, until the method reliably produces models that conform to all consistency requirements.

- Data for the third objective is analysed by constructing an ArchiMate model based on the selected case, following our own method to the letter, and then validating that model against the consistency requirements, by counting the number of requirement violations. The internal ArchiMate model of the case study and all views have been collected in the Archi tool and are available to inspection.

3.4. Reflection w.r.t. validity, reliability, and ethical aspects

For the first two objectives, KAOS needed to be translated to ArchiMate. Since the translation depends on subjective interpretation, we included the argumentation for the translation of each concept in an appendix. This allows the reader to check our translation. To improve the internal validity of the translation, we consulted ARMOR, a KAOS-based predecessor of the motivation extension of ArchiMate.

To improve the reliability and *internal validity* of the third objective we used two types of triangulation: *researcher triangulation* and *methodological triangulation* through the adoption of different complementary methods (Quintão, Andrade, & Almeida, 2020). We used researcher triangulation by having two researchers create their own views, derived from the same goal view. We used methodological triangulation by performing both a manual and a tool-based generation of elements from the goal view (Roubtsova & Severin, 2022).

The research result can be argued to be *externally valid*, but only within certain boundaries. Since our method was not developed for the selected case in particular, we expect that it can be generalized to similar cases. Those cases must have non-contradicting goals with only the goal refinement patterns mentioned. Most of the steps in our method are repeatable. Therefore, if the method is applied to a goal view, it should create predictable outcomes (object and lifecycle views), with only small variations.

4. Formulating a method for multi-view consistency in ArchiMate

4.1. Identifying consistency requirements

To transpose KAOS's viewpoints (or 'models') and viewpoint relationships to ArchiMate, we first translate between their metamodels. We look to ARMOR for inspiration. KAOS's metamodel is shown in Figure 10. Figure 12 shows a literal translation to ArchiMate, with a layout that resembles that of KAOS.

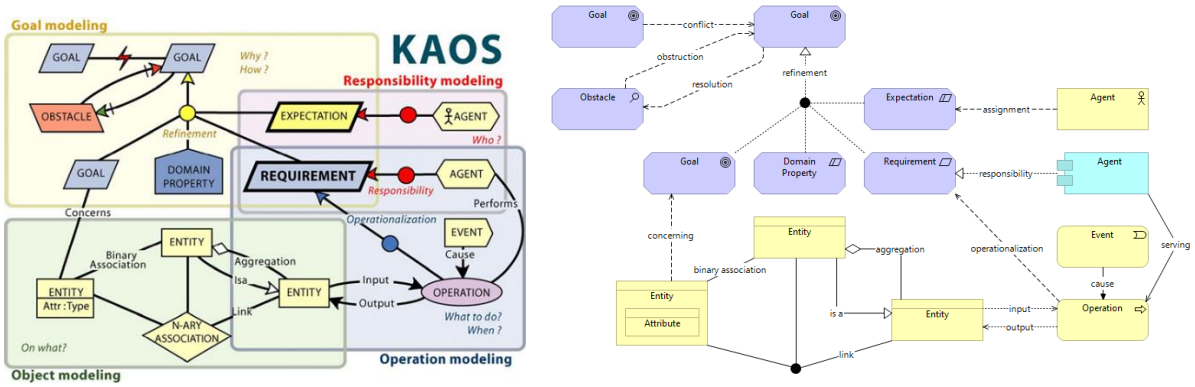


Figure 12. Literal translation of the KAOS metamodel (left) to ArchiMate (right)

What is not obvious from the translation in Figure 12 is that a KAOS agent is translated to both an active and a passive structure element in ArchiMate. This is done because in KAOS an agent specializes an object. Thus, an agent can be the input or output of an operation. This is not the case in ArchiMate: an access relationship cannot target an active structure element. Therefore, when we consider the active part of an agent, we will translate to an active structure element. If we consider the passive part of an agent, we will translate to a passive structure element. See Figure 13.

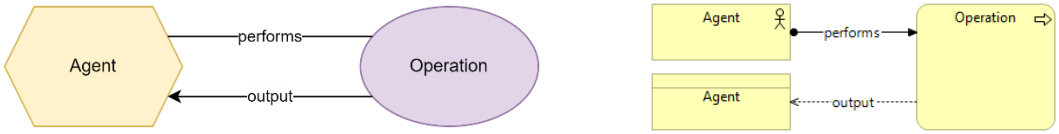


Figure 13. Translation of a KAOS agent (left) to both an ArchiMate business actor and business object (right)

A pattern emerges from the metamodel in Figure 12, which lends credence to our translation of the KAOS metamodel to ArchiMate:

- The upper left elements (the goal model in KAOS) all belong to the motivation extension in ArchiMate.
- The upper right elements (the responsibility model in KAOS) are all active structure elements in ArchiMate: things that display actual behaviour.
- The lower left elements (the object model in KAOS) are all passive structure elements in ArchiMate: things on which behaviour is performed.
- The lower right elements (the operation model in KAOS) all belong to the behaviour aspect of ArchiMate.

The translation of each concept in the KAOS metamodel to ArchiMate is listed side by side in Table 4. The motivation for each translation is provided in Appendix 2.

Table 4. Translation between KAOS and ArchiMate concepts

KAOS	Definition (Respect-IT, 2007)	ArchiMate	Definition (The Open Group, 2019)
Agent	Active Object* (=processor) performing operations* to achieve goals*. Agents can be the software being considered as a whole or parts of it. Agents can also come from the environment* of the software being studied; human agents are in the environment*.	Agent of the software being studied: Application Component and Business Object Agent in the environment: Business Actor and Business Object	An application component represents an encapsulation of application functionality aligned to implementation structure, which is modular and replaceable. A business actor represents a business entity that is capable of performing behavior. A business object represents a concept used within a particular business domain.
Assignment	*Not defined in the glossary.* Agents in the environment are assigned to expectations.	Influence relationship	The influence relationship represents that an element affects the implementation or achievement of some motivation element.
Association	Object*, the definition of which relies on other objects linked by the association.	Association relationship	An association relationship represents an unspecified relationship, or one that is not represented by another ArchiMate relationship.
Attribute	*Not defined in the glossary.* Entities may have attributes whose values define a set of states the entity can transition to.	Business Object (which is composed by another business object)	*See above.*
Cause	*Not defined in the glossary. A relationship between an event and an operation.*	Triggering relationship	The triggering relationship represents a temporal or causal relationship between elements.
Concern	*Not defined in the glossary.* Concerns relationship is used to link a requirement to the objects that are needed for it to be satisfied.	Influence relationship	*See above.*
Conflict	Goals* are conflicting if under some boundary condition the goals cannot be achieved altogether.	Influence relationship	*See above.*
Domain property	Descriptive assertion about objects* in the environment* of the software. It may be a domain invariant or a hypothesis. A domain invariant is a property known to hold in every state of some domain object, e.g., a physical law, regulation, … A hypothesis is a property about some domain object supposed to hold.	Constraint	A constraint represents a factor that limits the realization of goals.
Entity	Autonomous object*, that is, the definition of which does not rely on other objects.	Business Object	*See above.*
Event	Instantaneous object* (that is, an object alive in one state only) which triggers operations* performed by agents*.	Business Event	A business event represents an organizational state change.
Expectation	Goal* assigned to an agent* in the environment*.	Constraint	*See above.*
Input/output	*Not defined in the glossary.*	Access relationship	The access relationship represents the ability of behavior and active structure

			elements to observe or act upon passive structure elements.
Goal	Prescriptive assertion capturing some objective to be met by cooperation of agents*; it prescribes a set of desired behaviours. Requirements* and expectations* are goals.	Goal	A goal represents a high-level statement of intent, direction, or desired end state for an organization and its stakeholders.
Object	Thing of interest in the composite system* being modelled whose instances can be distinctly identified and may evolve from state to state. Agents, events, entities and associations are objects.	Structure element	*Not formally defined.* Any core element that is not a behavior element.
Obstacle	Condition (other than a goal) whose satisfaction may prevent some goal(s)* from being achieved; it defines a set of undesired behaviours.	Assessment	An assessment represents the result of an analysis of the state of affairs of the enterprise with respect to some driver.
Operation	Specifies state transitions of objects* that are input and/or output of the operation. Operations are performed by agents*.	Process	A process represents a sequence of behaviors that achieves a specific result.
Refinement	Relationship linking a goal* to other goals that are called its subgoals. Each subgoal contributes to the satisfaction of the goal* it refines. The conjunction of all the subgoals must be a sufficient condition entailing the goal* they refine.	Realization relationship	The realization relationship represents that an entity plays a critical role in the creation, achievement, sustenance, or operation of a more abstract entity.
Requirement	Goal* assigned to an agent* of the software being studied.	Requirement	A requirement represents a statement of need defining a property that applies to a specific system as described by the architecture.
Responsibility	Relationship between an agent* and a requirement*. Holds when an agent* is assigned the responsibility of achieving the linked requirement*.	Realization relationship	*See above.*

Note that the resulting metamodel in Figure 12 is not compliant with ArchiMate since the standard does not permit realization relationships between goals and between requirements. We are of the opinion that in a next version of the standard, ArchiMate should allow realization relationships between these elements for the following reasons:

- Other relationships cannot express refinement alternatives as shown in Figure 14. It is possible to mimic the graphical presentation in ArchiMate using, for example, influence relationships and and-junctions, but the semantics would not express refinement alternatives. For that we need the relationship to imply KAOS's 'sufficient condition' (see Table 4, *refinement*), which is included in ArchiMate's description of the realization relationship: 'The interpretation of a realization relationship is that the *whole or part* of the source element realizes the *whole of* the target element' (The Open Group, 2019).
- The specification explains that the 'realization relationship indicates that more abstract entities ("what" or "logical") are realized by means of more tangible entities ("how" or "physical")' (The Open Group, 2019). The standard allows this relationship between application components because it recognizes that one can distinguish between more

abstract (or 'logical') and more tangible (or 'physical') application components. In our opinion the same reasoning applies to goals and goal-related elements.

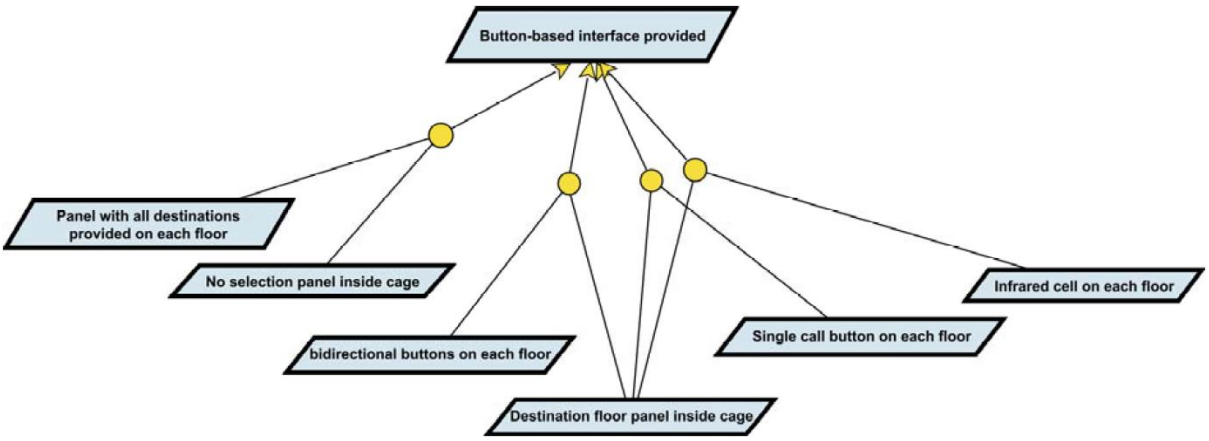


Figure 14. Goal refinement alternatives, from A KAOS Tutorial, V1.0*, by Respect-IT, 2007.*

Table 5 lists the changes that would need to be made to the specification. In this table each letter denotes a certain type of relationship: specialization, composition, aggregation, realization, influence, and association. The changes to the table are displayed as bold and underlined.

Table 5. Suggested modifications to ArchiMate's relationship table

from → ↓ to	Goal	Outcome	Principle	Requirement	Constraint
Goal	scg **r** n o	r n o	r n o	r n o	r n o
Outcome	n o	scg **r** n o	r n o	r n o	r n o
Principle	n o	n o	scg **r** n o	r n o	r n o
Requirement	n o	n o	n o	scg **r** n o	scg **r** n o
Constraint	n o	n o	n o	scg **r** n o	scg **r** n o

See Appendix 1 for the changes we made to Archi to allow for these extra relationships.

Literal translations often do not sound right to a native speaker, in this case to a 'speaker' of ArchiMate. We therefore adapt the literal translation of the KAOS metamodel to a more natural version, using native ArchiMate terms such as *business object* and *application component* instead of *entity* and *agent*. The result is shown in Figure 15.

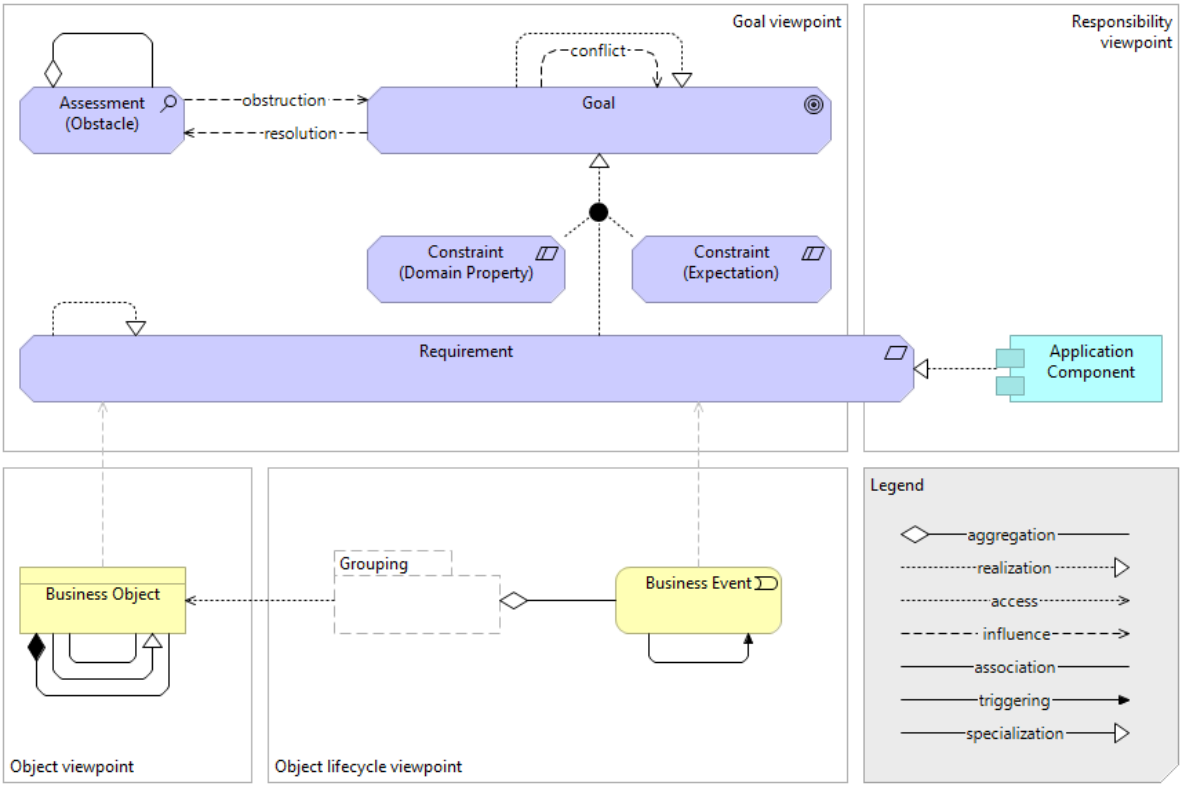


Figure 15. A more ArchiMate-native version of the KAOS metamodel

Note that we have also made a few substantive changes to the metamodel:

- We have dropped the *operation*. The reason for this is that we are not interested in modelling the inner workings of the application component. We therefore only model events that are external to the application component and that have meaning from a business perspective.
- We aggregate *business events* into *groupings* (object lifecycles), to resemble ExtREME: 'Each business concept has its own life cycle: it is created, goes through decisions and specific states and may be deleted' (Roubtsova, Interactive Modeling and Simulation in Business System Design, 2016).
- We do not relate *application components* to *business events* since these relationships should be derived via the *requirements*.
- We have dropped the agent in the environment. Adding external actors to the goal model as independent elements decreases the readability of the goal view. The external actors can simply be mentioned in the descriptions of expectations.
- The greyed-out relationships do not appear in the model. Instead, the analyst uses numberings and a visual layout on his private view to express these relationships, which will be demonstrated when we model the insurance case.

Table 6 lists the KAOS key ideas and completeness criteria, which are candidates for reuse in ArchiMate.

Table 6. Our ArchiMate viewpoints mapped to KAOS key ideas and completeness criteria

Viewpoint	KAOS key idea (Respect-IT, 2007)	KAOS completeness criterion (Respect-IT, 2007)
Goal viewpoint	**Key idea 1**: First build a requirements model **Key idea 2**: Justify your requirements by linking them to higher-level goals	**Completeness criterion 1**: A goal model is said to be complete with respect to the refinement relationship 'if and only if' every leaf goal is either an expectation, a domain property (DomProp) or a requirement. **Completeness criterion 2**: A goal model is complete with respect to the responsibility relationship 'if and only if' every requirement is placed under the responsibility of one and only one agent (either explicitly or implicitly if the requirement refines another one which has been placed under the responsibility of some agent).
Responsibility viewpoint	**Key idea 4**: Build a responsibility model	
Object viewpoint	**Key idea 5**: Build a consistent and complete glossary of all the problem-related terms you use to write the requirements	
Object lifecycle viewpoint	**Key idea 6**: Describe how the agents need to behave in order to satisfy the requirements, they are responsible for	**Completeness criterion 3**: To be complete, a process diagram must specify (i) the agents who perform the operations (ii) the input and output data for each operation. **Completeness criterion 4**: To be complete, a process diagram must specify when operations are to be executed. **Completeness criterion 5**: All operations are to be justified by the existence of some requirements (through the use of operationalization links).

KAOS states that 'refinement is no longer necessary as soon as a goal has been placed under the responsibility of a single agent' (Respect-IT, 2007). This single-agent criterion is what distinguishes a goal from a requirement in KAOS. However, such a requirement can be formulated in a way that is still too vague for automation by software, and for consistency with other views. We therefore combine KAOS's single-agent criterion with ExtREME's countability criterion: 'If the refinement of a goal is finished, then the leaves of the goal refinement tree represent requirements and they are expressed using the countable and (or) comparable concepts and rely on business domain knowledge' (Roubtsova, Interactive Modeling and Simulation in Business System Design, 2016). In our method, both criteria must be met. ExtREME's countability criterion enables us to derive business objects, and their lifecycles, from requirements.

In our opinion, the greatest challenge in keeping views consistent lies between three viewpoints: the goal, object, and object lifecycle viewpoint, because the semantics of other viewpoints also includes goal, object and object lifecycle but for different objects. We have therefore selected a case with only one application component, so that we can ignore the responsibility viewpoint. This case also has no obstructions, expectations, or domain properties, since there is no need for a complex goal model to demonstrate the challenges associated with multi-view consistency. KAOS distinguishes between *achieve*, *maintain*, *cease*, and *avoid goals*. We only use *achieve goals*. KAOS distinguishes between different types of goal refinement, such as milestone-driven and case-driven (Respect-IT, 2007). Inspired by KAOS, we distinguish between three patterns of goal refinement: OR refinement, AND refinement, and milestone refinement. These patterns are displayed in Figure 16, using classical operators to express temporal logic: ◊ (eventually) and □ (always in the future). Note that there is a difference between OR refinement and refinement alternatives. The latter would be expressed using multiple junctions (see Figure 29).

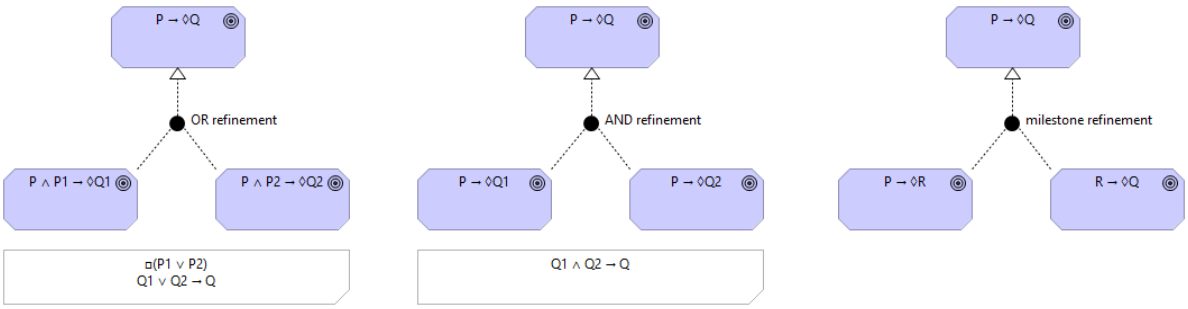


Figure 16. Three goal refinement patterns

Using these viewpoints and goal refinement patterns as our basis, we can now define viewpoint relationships, defined as a set of rules and guidelines. Rules are formally defined and can be automated. We distinguish between the following viewpoint relationships (see Figure 17):

1. Between goal and responsibility viewpoint
2. Between goal and object viewpoint
3. Between object and object lifecycle viewpoint
4. Between goal and object lifecycle viewpoint

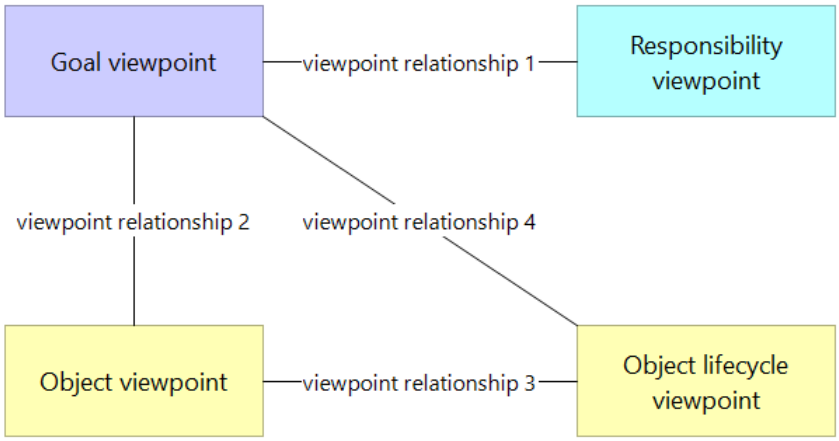


Figure 17. The viewpoints of our method and their relationships

The goal view is the reference view with respect to consistency. It contains a finite set of requirements $r \in R$. The names of these requirements are descriptions in natural language of desired states. From these descriptions, we can derive state information (hold by objects) and state changes (events). While analysing requirements we must consider (or 'undo') some properties of natural language, such as phrase-level contractions, in which one or more whole words within a phrase are omitted because they have the same form and meaning as another part.

Let R be a set of requirements, ordered by their goal refinement tree. Each requirement $r \in R$ is a relation (G, N, V), where $G \subset N \times V$ is a subset of all possible combinations of normalized noun phrases N and verb phrases V that occur in the requirement's description. We say that $R(n, v)$ if $(n, v) \in G$. For example, $R(\text{'Customer','is registered'})$.

A noun or verb phrase is normalized by transforming it to singular form, deleting adjectives, substituting synonyms, etcetera. For example, $normalize(\text{'Registered customers'}) = \text{'Customer'}$ or $normalize(\text{'a client'}) = \text{'Customer'}$.

The first requirement in the refinement tree that is related to a certain normalized noun phrase $n \in N$ is considered to contain an associated creation verb (e.g., 'created', 'registered', etcetera), even if not explicitly mentioned by the requirement's description.

Viewpoint relationship 1 is defined by the following rules:

> **1.1**: Each top-level requirement is realized by exactly one application component. A requirement is top-level if it does not refine another requirement.

Viewpoint relationship 2 is defined by the following rules and guidelines:

> **2.1**: There is an object $o \in O$ for each normalized noun phrase $n \in N$.
>
> **2.2**: Two objects $o_1, o_2 \in O$ have a relationship if their corresponding noun phrases $n_1, n_2 \in N$ are related to the same verb phrase $v \in V$. Thus $R(n_1, v) \wedge R(n_2, v) \vdash R(o_1, o_2)$.
>
> **2.3**: Noun phrases are not only connected by verb phrases, but they can also be connected by prepositions. In such cases, the two objects $o_1, o_2 \in O$ in question also have a relationship.
>
> **2.4**: The object view contains specialization relationships for keywords such as 'each' and 'one of'.
>
> **2.5**: The object view may contain extra objects and relationships, except extra autonomous objects. An object is considered autonomous if it does not compose another object. It is left up to the analyst to decide whether a specialized object must be regarded as autonomous (and thus must have an associated lifecycle, see rule 3.1).

Viewpoint relationship 3 is defined by the following rules:

> **3.1**: The object lifecycle view contains a grouping for each autonomous object in the object view.
>
> **3.2**: Each grouping is named after its autonomous object and follows this pattern: 'The lifecycle of a(n) <object name>'.
>
> **3.3**: An event can be expressed as a set of tuples $\{(o_1, v), (o_2, v), \ldots | o_1, o_2, \ldots \in O \wedge v \in V\}$. An event is aggregated by a grouping if and only if one of the event's objects $o_1, o_2, \ldots \in O$ is equal to that grouping's related object or one of its components (or specializations).

Viewpoint relationship 4 is defined by the following rules and guidelines:

> **4.1**: The object lifecycle view contains an event for each verb phrase $v \in V$.
>
> **4.2**: Each event's name contains the identification of the related requirement $r \in R$, the verb phrase $v \in V$ and all related noun phrases $n_1, n_2, \ldots \in N$.

4.3: There are cycles and splits for plurals and keywords such as ‘each’ and ‘one of’.

4.4: Events that are aggregated by the same grouping have the same ordering as their corresponding requirements in the goal refinement tree.

4.5: Adjacent events are connected via an or-junction if they are aggregated by the same grouping and if the shortest path between their corresponding requirements contains an OR refinement.

4.6: Adjacent events are connected via an and-junction if they are aggregated by the same grouping and if the shortest path between their corresponding requirements contains an AND refinement.

4.7: Adjacent events are connected by a triggering relationship if they are aggregated by the same grouping and if the shortest path between their corresponding requirements contains a milestone refinement.

4.8: The object lifecycle view may contain extra relationships and junctions, but not extra events, other than those defined by another rule or guideline.

4.2. Formulating a method for multi-view consistency

The following method should produce a set of views in Archi that are compliant with the consistency requirements defined in section 4.1. We assume that there is only one view for each viewpoint.

1. Creating the **goal view**
    a. Create a new view in Archi.
    b. Add elements and relationships until the view meets the following completeness criteria:
        i. Every leaf is either an expectation, a domain property, or a requirement (Respect-IT, 2007).
        ii. Every requirement is expressed using countable and/or comparable nouns (Roubtsova, Interactive Modeling and Simulation in Business System Design, 2016).
    c. Number each element.
    d. Perform a validation of the model in Archi and delete all unused elements and relationships from the abstract model.
2. Performing a **lexical analysis** of the goal view
    a. Create a new view in Archi. This view is only intended for the analyst, as a helper view to create object and object lifecycle views that are consistent with the goal view.
    b. Add all leaf requirements from the abstract model to this view and place them one below the other, in the same order as they appear in the goal view.
    c. Create a visual group next to each leaf requirement, to group all objects and events to be identified in the next step.
    d. Identify objects, events and relationships using the lexical analysis guidelines listed below. While doing so, aggregate events into groupings representing object lifecycles, where each grouping has an access relationship to one and one autonomous object only. An autonomous object is an object with an independent

lifecycle, meaning it is not a component (or specialization) of another object. To make the identification of creation events easier, grey out duplicate objects that have been identified in a previous requirement. Apply the following lexical analysis guidelines and document their use in a note, one next to each visual grouping:

i. Create an object for each normalized noun phrase.
ii. Create an event and/or object relationship for each verb phrase and preposition. Reuse the numbering of requirements for events.
iii. Transform plurals and keywords such as 'each', 'one of', etc. into object specializations and event cycles and splits.
iv. Have events access (via their groupings) all the objects they mention. If they mention non-autonomous objects, have them access the autonomous objects they are related to by a specialization or composition relationship. If an event accesses multiple autonomous objects, duplicate that event on the view, so it can be aggregated into multiple groupings.
v. Add creation events for autonomous objects.
vi. Add objects, events, and relationships for other reasons, such as domain knowledge, symmetry, parent goals, etc.

e. Validate the model in Archi and delete all unused elements and relationships from the abstract model.

3. Creating the **object view**
    a. Create a new view in Archi.
    b. Select all business objects from the model tree, put them on the view and rearrange them.
    c. Optional: add relationships and non-autonomous objects. If the need arises to add autonomous objects, return to step 1 to refine the goal view.
    d. Validate the model in Archi and delete all unused elements and relationships from the abstract model.
4. Creating the **object lifecycle view**
    a. Create a new view in Archi.
    b. In the lexical analysis view one instance of a grouping in the abstract model can occur multiple times in the view. For each set of identical groupings:
        i. Copy the set of identical groupings from the lexical analysis view, preserving their order.
        ii. Place all events that belong to the same set of identical groupings in one visual instance of that grouping and remove the other, identical groupings from the view.
    c. Within each grouping, add triggering relationships, such that there exists a path from the first event(s) to the last event(s). Use junctions if necessary.
        i. Add or-junctions in case of OR goal refinement.
        ii. Add and-junctions in case of AND goal refinement.
    d. Optional: add cycles. If the need arises to add events, return to step 1 to refine the goal view.

The method prescribes requirements and events to share their numberings. This serves two purposes: to allow stakeholders to check the derivation of events from requirements and to assist in the identifying duplicate instances of the same event in a view. These duplicate instances are used for the parallel composition of object lifecycles, meaning that an event can only fire if it can fire in all other lifecycles.

4.3. Testing our method on a case

We have selected an insurance case from Roubtsova (2016, pp. 50-58) to test our method for creating consistent goal-oriented multi-view models in ArchiMate:

> The main goal of any insurance business is to sell the insurance products. An insurance product covers possible costs of a product user in possible predefined situations. A health insurance covers the costs of medical procedures needed in case of health problems.
>
> An application for insurance business should support the composition of a product based on the covered medical procedures. The composition is usually done by the insurance company.
>
> An instance of an insurance product is called a policy. The system should support an act of buying of a policy by a customer (a person). After that, the customer becomes a policy-holder or a client.
>
> When a client undergoes a medical treatment, he/she pays for this treatment. In order to compensate the costs of the treatment, the client submits claims to the insurance company. Therefore, another goal of the insurance business is handling the claims submitted by clients. The handling should comply the rules fixed as a law by the government. The rules are assigned to medical procedures composed in the insurance product.

Figure 18 shows the goal model, refinement to requirements, belonging to the case description.

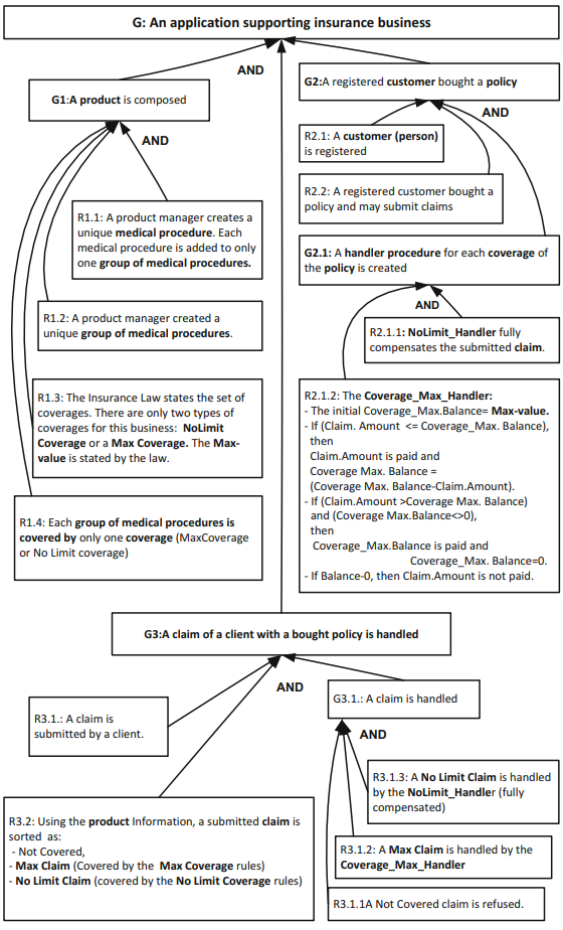


Figure 18. A goal model of the insurance case. From Interactive Modeling and Simulation in Business System Design*, by E. Roubtsova, 2016, Springer International Publishing.*

This goal model is already of high quality because it has been validated using an ExtREME protocol model, which is executable. We have therefore translated this model to ArchiMate with almost no changes, see Figure 19. This figure shows the three types of goal refinement patterns described earlier: OR refinement, AND refinement, and Milestone refinement. The goal view is complete with respect to the completeness criteria mentioned in step 1.b of the method.

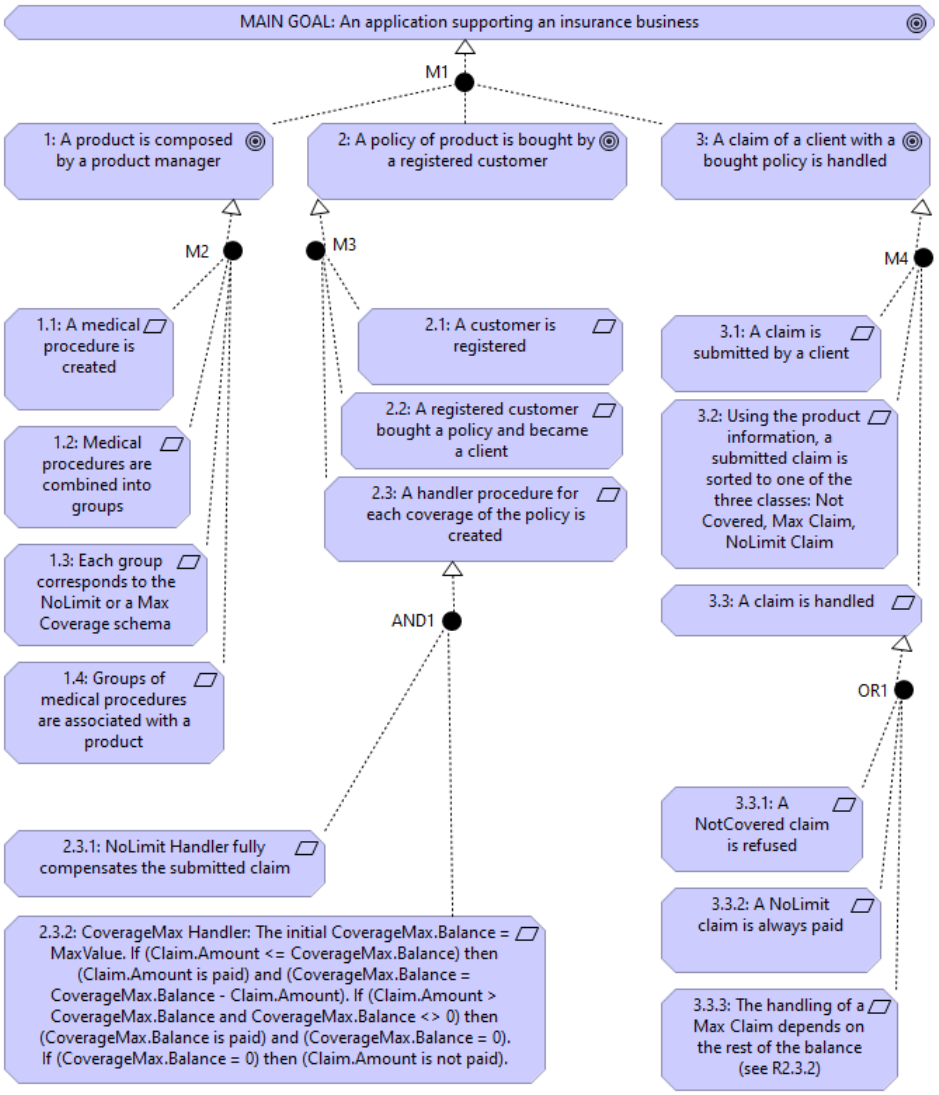


Figure 19. The goal view of the insurance case

The next step in our method is to perform a lexical analysis of the goal view. Figure 20 shows part of the initial setup of the lexical analysis view, resulting from executing steps 2.a, 2.b and 2.c.

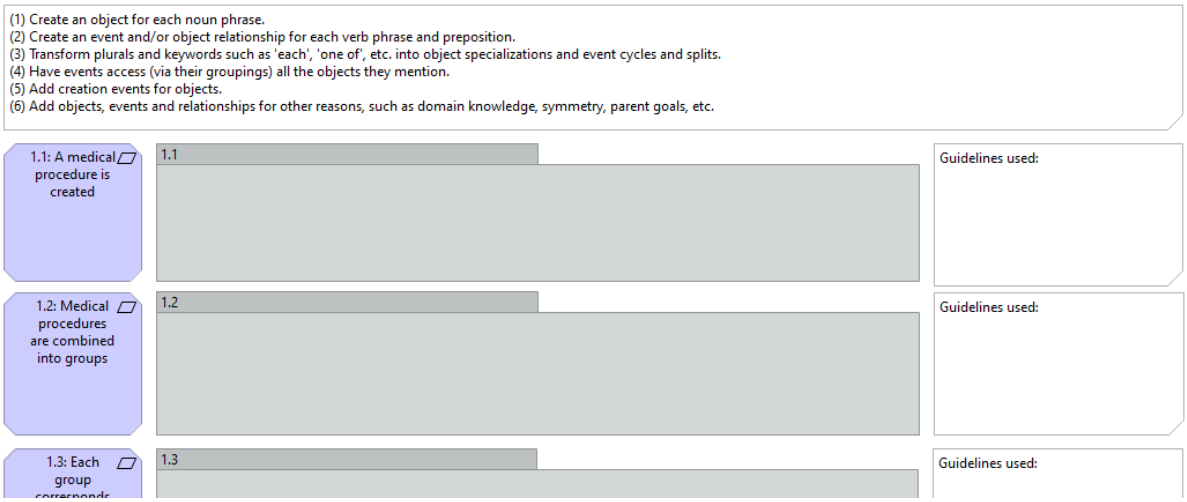


Figure 20. Initial setup of the lexical analysis view

The result of step 2.d is shown in Figure 21. A business object nested inside another object denotes a specialization relationship. Notice how:

- Events inherit their numbering from requirements. This allows stakeholders to easily recognize the consistency between the object lifecycle view and the goal view. This is not possible for business objects, because they are identified in multiple requirements.
- All autonomous objects have access relationships.
- Identical groupings appear multiple times in the view if they access multiple autonomous objects, see for example the grouping named *1. The lifecycle of a medical procedure*, which appears twice. Events in these identical groupings will be connected in the object lifecycle view, following the goal refinement patterns.
- Duplicate instances of the same business object are greyed out. Each autonomous business object that is not greyed out has a creation event next to it.

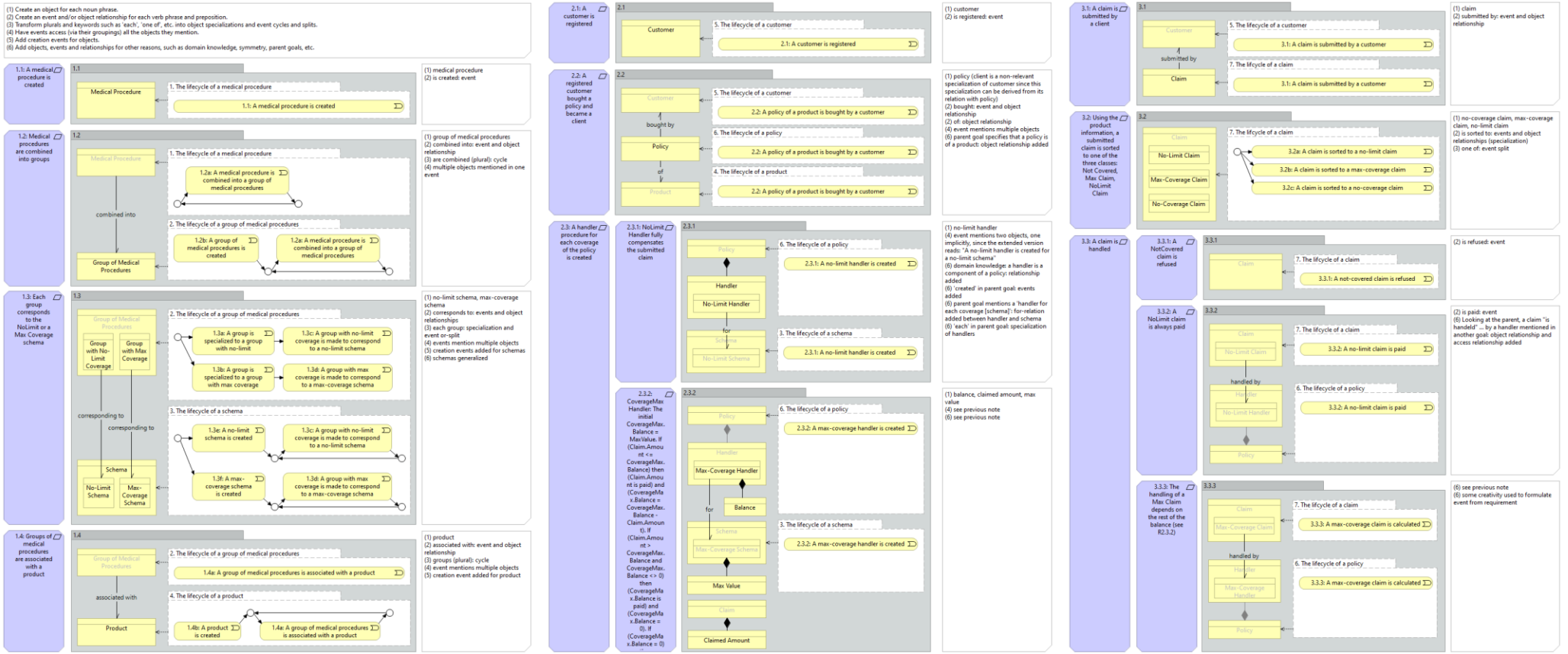


Figure 21. The complete lexical analysis view of the insurance case

In step 3 we create the object view. Figure 22 shows the process of creation: we select all business objects from the abstract model, place them on the view an rearrange them. The resulting view on the right is already an almost complete object view, which is purely the result of our lexical analysis in the previous step.

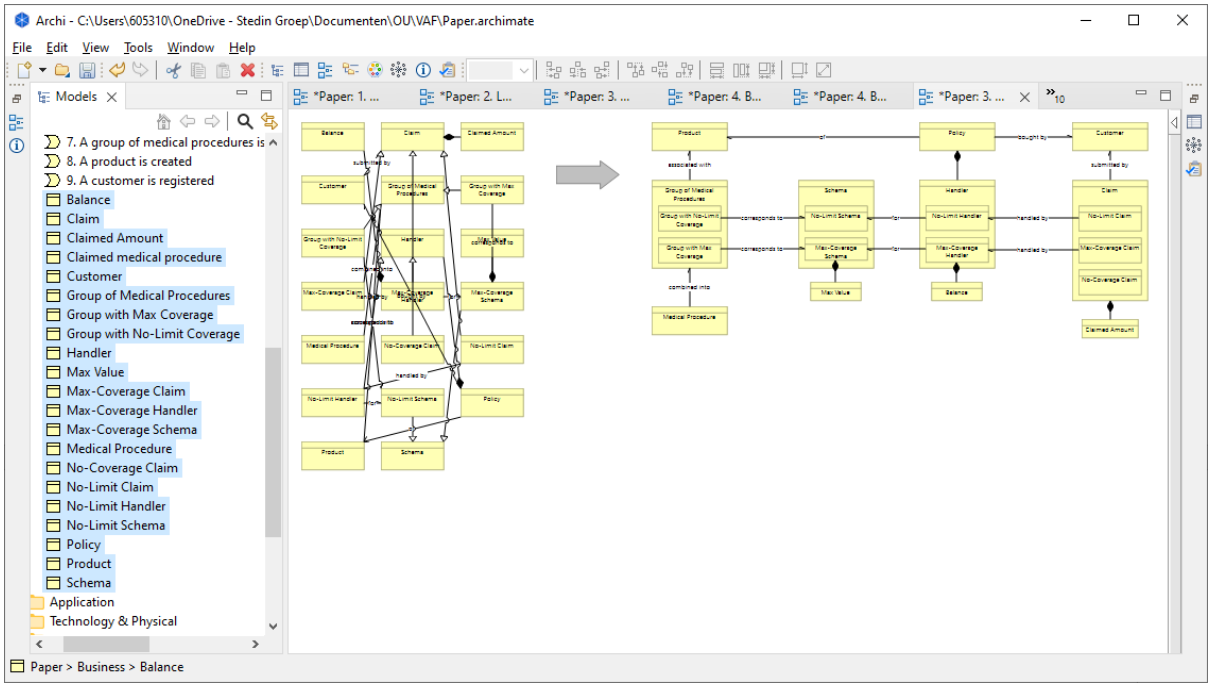


Figure 22. Construction of the object view of the insurance case

Figure 23 shows the final object view, in which only one (non-autonomous) object and one relationship have been added (green) in addition to those already identified during lexical analysis.

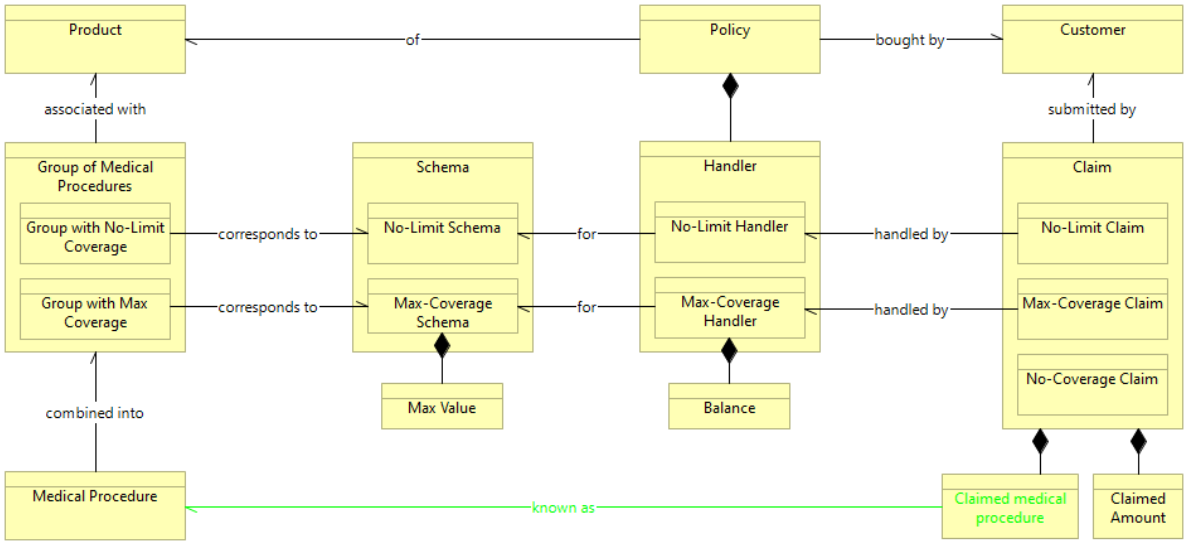


Figure 23. The final object view of the insurance case

In step 4 we construct the object lifecycle view. Figure 24 shows how this view is constructed. The upper part shows the identical groupings that have been copied over from the lexical analysis view.

The bottom part shows how identical groupings are deduplicated and how events are linked with triggering relationships (red).

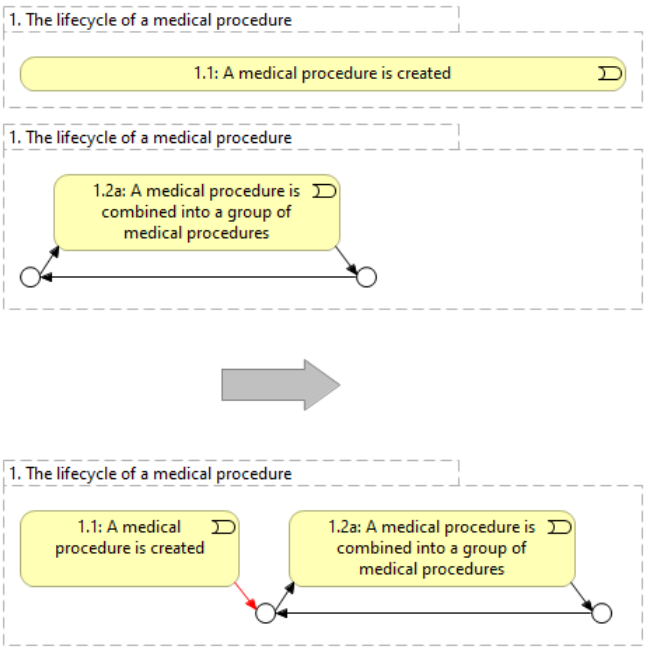


Figure 24. Construction of the object lifecycle view of the insurance case, showing the lifecycles of only two objects

Figure 25 shows the completed object lifecycle view. Triggering relationships are coloured red if they have been added based on goal refinement patterns. Additional events and relationships are coloured green. Note how events 2.3.1 and 3.3.2 are connected using two junctions. This corresponds to the refinement patterns that we encounter when traversing through the goal view, taking the shortest path from requirement 2.3.1 to requirement 3.3.2. The first junction between the two events is an and-join, which corresponds to the upward traversal through *AND1*. The second junction between the two events is an or-split, which corresponds to a downward traversal through *OR1*.

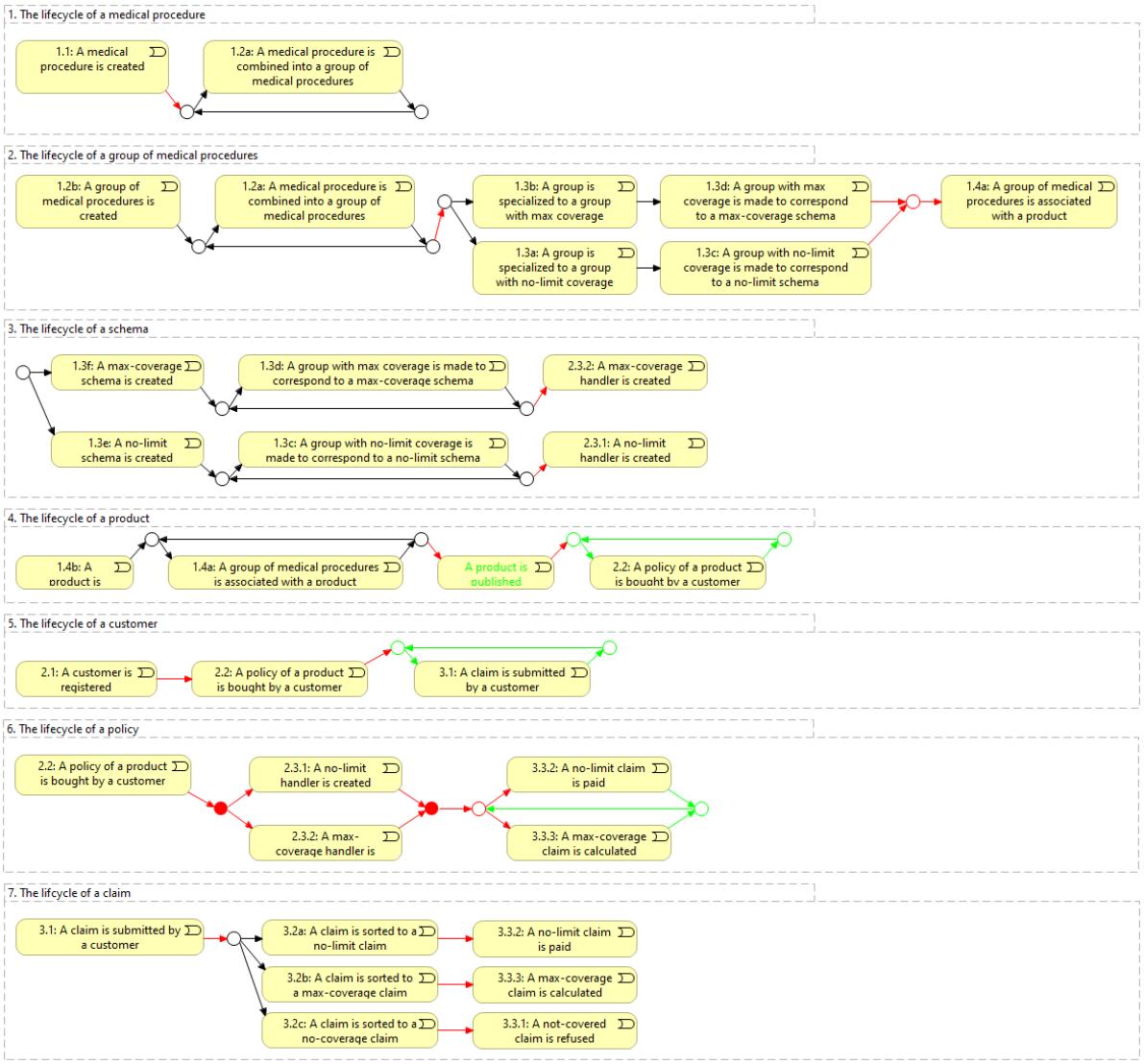


Figure 25. The completed object lifecycle view of the insurance case

Events are composed using CSP parallel composition: an event can fire in one lifecycle if it can fire in all lifecycles using this event. For example, a policy can only be bought (event 2.2 in lifecycle 6) after the customer is registered (event 2.1 in lifecycle 5) and the product is published (green event in lifecycle 4). Thus, the lifecycles of *Product*, *Customer*, and *Policy* are synchronised by the event *2.2: A policy of a product is bought by a customer*.

The green event (*A product is published*) is not derived from the goal view. This event has been added by the analyst to prevent customers from buying incomplete products. At this point we are violating consistency requirement 4.8. Step 4.d tells us to refine the goal model to restore consistency. Figure 26 shows the updated refinement of goal 1, to which requirement 1.5 has been added.

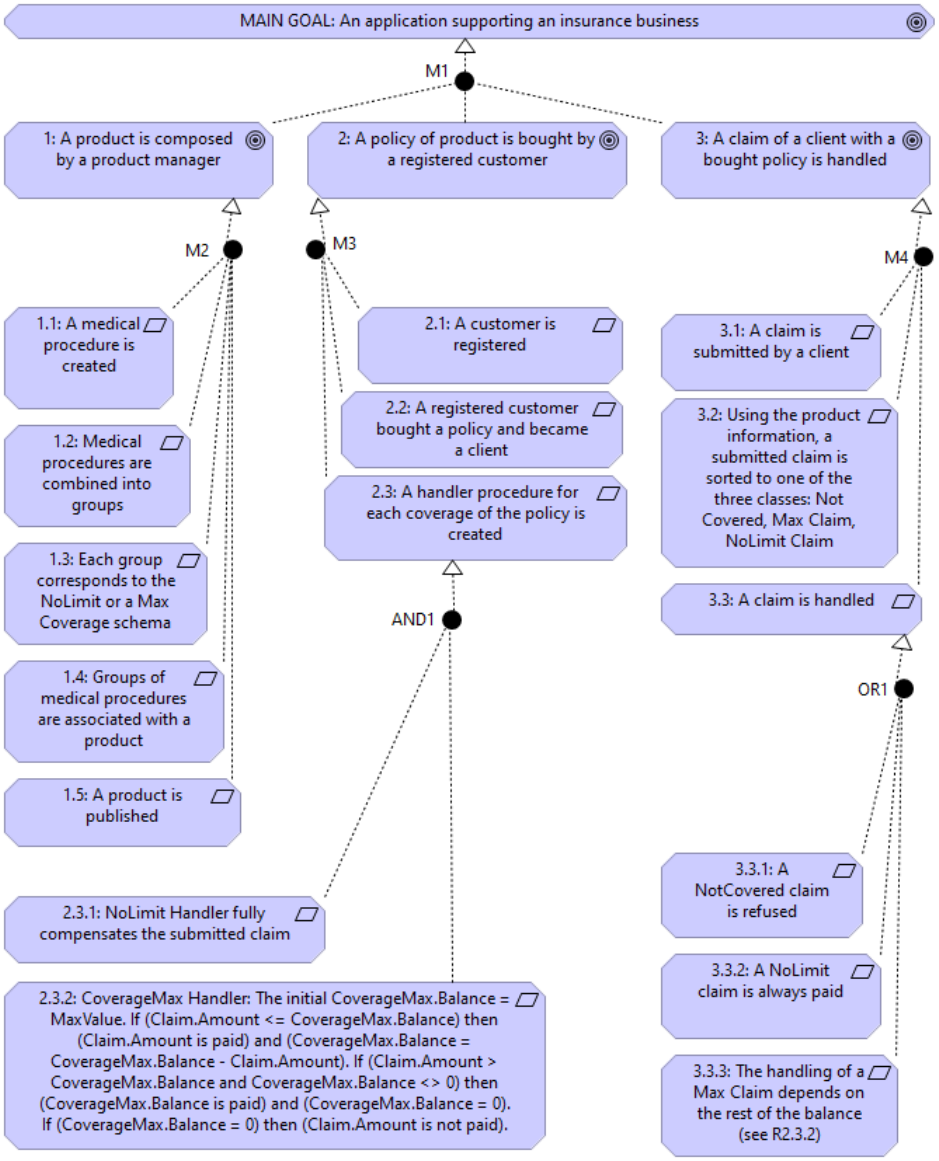


Figure 26. An updated version of the goal view

To determine the effectiveness of our method, the consistency between the goal, object and object lifecycle view has been checked by counting the number of violations of the consistency guidelines. We found that none of the consistency guidelines were violated. However, counting violations depends partly on the subjective interpretation of goal descriptions. Therefore, we also looked at the object and object lifecycle views of a second researcher who used the same method. The produced models turned out to be very similar, but we also discovered some differences. For example, one researcher identified *Registered Customer* as an object rather than just *Customer*. Another example is that one researcher generalized the *No-Limit Schema* and *Max-Coverage Schema* to *Schema*, while the other did not, which also led to differences in the object lifecycle views. For example, the generic object *Schema* reduces the number of lifecycles. We regard such differences as trivial and valid variants of perspective, within the latitude allowed by our guidelines.

5. Discussion, conclusions and recommendations

5.1. Discussion – reflection

Our method for consistent modelling combines key ideas and completeness criteria from KAOS with ideas from ExtREME on how to create protocol models from goal models, using lexical analysis and parallel composition of events. The result is a method that very much resembles KAOS but with an important addition: our method ensures consistency in a way that is not present in KAOS. A KAOS Tutorial (2007) notes:

> Many companies have noticed that users and IT analysts most often do not understand each other very well. KAOS provides the right connection between the two worlds: users quickly feel confident with goal and responsibility models; analysts like the object and operation models. (p. 9)

However, the tutorial gives the modeler quite some freedom in creating the operation model, merely requiring the modeler to 'justify' each operation by linking it to a requirement.

Our method restricts the ways in which a modeler can justify the existence of elements. This restriction should increase consistency, ensuring that users and IT analysts, when looking at their preferred views, are looking at different sides of the same proverbial medal. They recognise and use the same nouns in requirements and business objects, they use the same verbs in requirements and events, etcetera. Since we reuse the numbering of requirements in event names, the stakeholders can more easily verify the consistency between views.

I was lucky that my supervisor independently modelled the same case using roughly the same method but manually. This allowed me to compare the models. The differences between our models turned out to be remarkably small, which increases the reliability of the method. Still, the method has been applied to only one case, which is also a relatively simple one. Generalization to other cases should be possible, but only if those cases share the same characteristics: having only non-conflicting goals, with only the three refinement patterns mentioned.

While comparing ArchiMate to KAOS we accidentally discovered that ArchiMate lacks the goal refinement relationship. We therefore deviated from the ArchiMate standard and specified why and how it should be changed.

5.2. Conclusions

One of the essential requirements for the design of ArchiMate is that 'it must be possible to perform consistency checking of architectures' (Lankhorst, Proper, & Jonkers, 2009). Although ArchiMate supports consistency checking, it does not provide guidelines to create consistent multi-view models. Other modelling approaches that do provide such guidelines typically regard the goal view as the reference view for consistency. The goal view in ArchiMate can be modelled with elements from the motivation extension. This extension was inspired by several methods, of which KAOS is the only multi-view method. We therefore selected KAOS as our source of inspiration. We also took inspiration from ExtREME because multi-view consistency is part of the core of this approach.

Inspired by KAOS and ExtREME, we formulated consistency requirements and a method to produce consistent models in ArchiMate. We tested this method on an insurance case and found that the resulting model complies with the consistency requirements.

5.3. Recommendations for practice

Practitioners of ArchiMate can use our method as inspiration on how to create consistent goal-oriented models, and as a steppingstone to learn about goal-oriented requirements engineering (GORE) in general. Practitioners of KAOS can use our translation of its metamodel to translate their models to ArchiMate, for example to integrate them with existing ArchiMate models in the enterprise.

The authors of ArchiMate can use our research as justification to change the standard to allow for realization relationships between goals and between requirements.

5.4. Recommendations for further research

The method could be refined to apply to more complex cases. For example, we restricted ourselves to only achieve goals, while there are also maintain, cease, and avoid goals in KAOS. We also restricted ourselves to three goal refinement patterns.

We have restricted ourselves to the business layer of ArchiMate. The method could be extended to include goals of implementation and their transformation to implementation views (in the application and technology layer of ArchiMate).

Many requirements for consistency formulated in this work can be formalised in future work and used for automated consistency checks, which demands add-ins to the ArchiMate supported tools.

References

Babkin, E. A., & Ponomarev, N. O. (2017). Analysis of the consistency of enterprise architecture models using formal verification methods. *Business Informatics, (3)*, 30-40.

Beauvoir, P., Sarrodie, J.-B., & The Open Group. (2021). *Archi User Guide, Version 4.9.1.*

Cardoso, E., Almeida, J., & Guizzardi, R. (2010). On the support for the goal domain in enterprise modelling approaches. *14th IEEE International Enterprise Distributed Object Computing Conference Workshops*, 335-344.

Dijkman, R., Quartel, D., & Van Sinderen, M. (2006). Consistency in multi-viewpoint architectural design of enterprise information systems. *BETA publicatie : working papers; Vol. 188*.

Engelsman, W., Quartel, D., Jonkers, H., & van Sinderen, M. (2011). Extending enterprise architecture modelling with business goals and requirements. *Enterprise information systems, 5(1)*, 9-36.

Kotusev, S. (2019). Enterprise architecture and enterprise architecture artifacts: Questioning the old concept in light of new findings. *Journal of Information technology 34(2)*, 102-128.

Kruchten, P. (1995). Architectural blueprints—the “4+ 1” view model of software architecture. *IEEE software, 12(6)*.

Lankhorst, M., Proper, H., & Jonkers, H. (2009). The architecture of the archimate language. *Enterprise, business-process and information systems modeling*, 367-380.

Nuseibeh, B., Kramer, J., & Finkelstein, A. (1994). A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on software engineering, 20(10)*, 760-773.

Quartel, D., Engelsman, W., Jonkers, H., & Van Sinderen, M. (2009). A goal-oriented requirements modelling language for enterprise architecture. *IEEE International Enterprise Distributed Object Computing Conference*, 3-13.

Quintão, C., Andrade, P., & Almeida, F. (2020). How to Improve the Validity and Reliability of a Case Study Approach? *Journal of Interdisciplinary Studies in Education, 9(2)*, 264-275.

Respect-IT. (2007). *A KAOS Tutorial, V1.0.*

Roubtsova, E. (2016). *Interactive Modeling and Simulation in Business System Design.* Springer International Publishing.

Roubtsova, E., & Severin, S. (2022). Semantic Relations of Sub-Models in an Enterprise Model. *Accepted for BMSD 2022, the 12th International Symposium on Business Modeling and Software Design*.

Sandkuhl, K., Stirna, J., Persson, A., & Wißotzki, M. (2014). *Enterprise modeling.* Heidelberg: Springer.

The Object Management Group. (2015). *Business Motivation Model, Version 1.3.* https://www.omg.org/spec/BMM.

The Open Group. (2013). *ArchiMate® 2.1 Specification.*

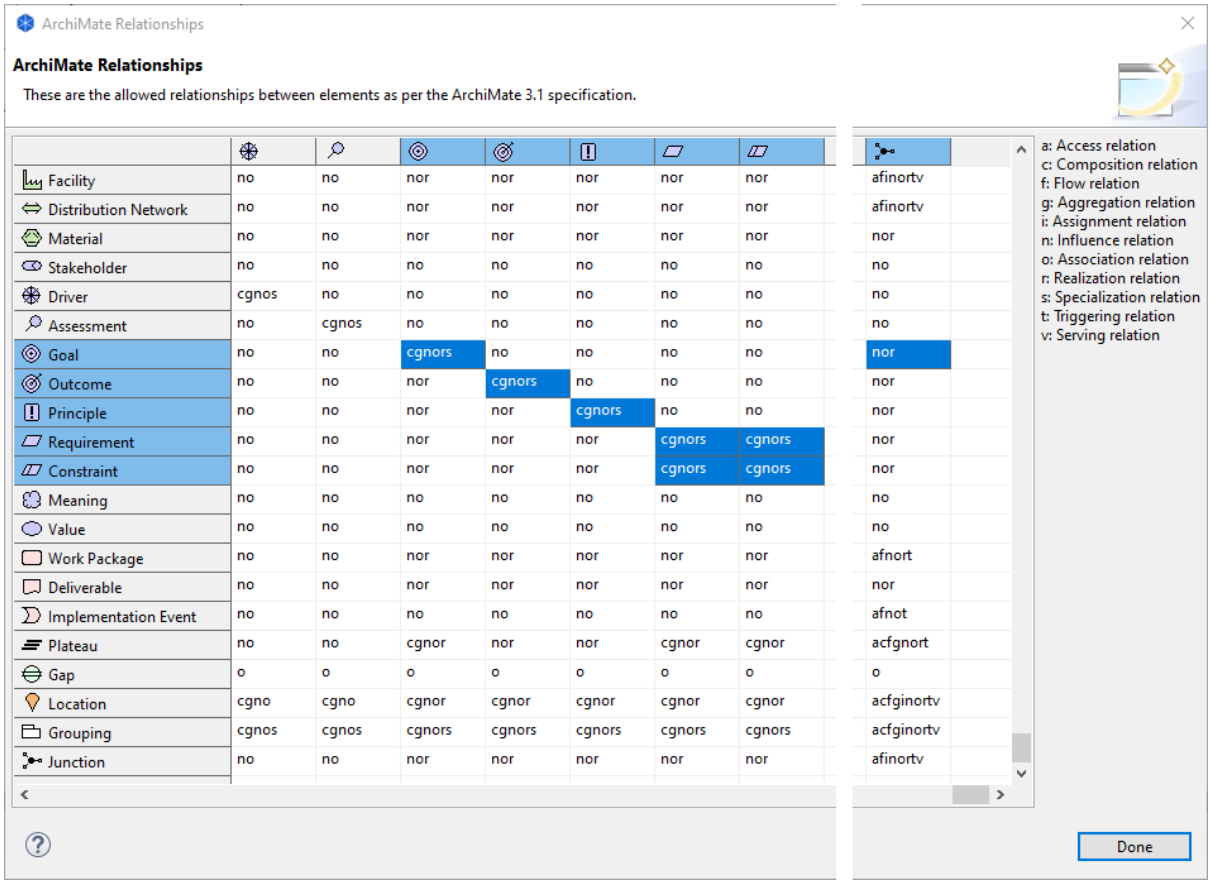
The Open Group. (2018). *the TOGAF® Standard, Version 9.2.*

The Open Group. (2019). *ArchiMate® 3.1 Specification.* Van Haren Publishing.

Webster, J., & Watson, R. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly, xiii-xxiii*.

Yu, E. S., & Mylopoulos, J. (1994). Understanding" why" in software process modelling, analysis, and design. *Proceedings of 16th international conference on software engineering*, 159-168.

Appendix 1: Customized relationships table in Archi



Facility	no	no	nor	nor	nor	nor	nor	afinortv
Distribution Network	no	no	nor	nor	nor	nor	nor	afinortv
Material	no	no	nor	nor	nor	nor	nor	nor
Stakeholder	no	no	no	no	no	no	no	no
Driver	cgnos	no	no	no	no	no	no	no
Assessment	no	cgnos	no	no	no	no	no	no
Goal	no	no	cgnors	no	no	no	no	nor
Outcome	no	no	nor	cgnors	no	no	no	nor
Principle	no	no	nor	nor	cgnors	no	no	nor
Requirement	no	no	nor	nor	nor	cgnors	cgnors	nor
Constraint	no	no	nor	nor	nor	cgnors	cgnors	nor
Meaning	no	no	no	no	no	no	no	no
Value	no	no	no	no	no	no	no	no
Work Package	no	no	nor	nor	nor	nor	nor	afnort
Deliverable	no	no	nor	nor	nor	nor	nor	nor
Implementation Event	no	no	no	no	no	no	no	afnot
Plateau	no	no	cgnor	nor	nor	cgnor	cgnor	acfgnort
Gap	o	o	o	o	o	o	o	o
Location	cgno	cgno	cgnor	cgnor	cgnor	cgnor	cgnor	acfginortv
Grouping	cgnos	cgnos	cgnors	cgnors	cgnors	cgnors	cgnors	acfginortv
Junction	no	no	nor	nor	nor	nor	nor	afinortv

Figure 27. An edited version of the relationships table in Archi that allows realization relationships between instances of goal modelling elements of the same type

Table 7. Changes to Archi's relationships.xml file to allow realization relationships between instances of goal modelling elements of the same type

Line	Original version	Edited version
1	`<?xml version="1.0" encoding="UTF-8"?>`	`<?xml version="1.0" encoding="UTF-8"?>`
2	`<relationships version="3.1">`	`<relationships version="3.1">`
	…	…
1434	`<source concept="Constraint">`	`<source concept="Constraint">`
	…	…
1457	`<target concept="Constraint" relations="cgnos" />`	`<target concept="Constraint" relations="cgnors" />`
	…	…
1483	`<target concept="Requirement" relations="cgnos" />`	`<target concept="Requirement" relations="cgnors" />`
	…	…
1497	`</source>`	`</source>`
	…	…
2149	`<source concept="Goal">`	`<source concept="Goal">`
	…	…
2183	`<target concept="Goal" relations="cgnos" />`	`<target concept="Goal" relations="cgnors" />`
	…	…
2186	`<target concept="Junction" relations="no" />`	`<target concept="Junction" relations="nor" />`
	…	…
2212	`</source>`	`</source>`
	…	…
2669	`<source concept="Outcome">`	`<source concept="Outcome">`
	…	…
2711	`<target concept="Outcome" relations="cgnos" />`	`<target concept="Outcome" relations="cgnors" />`
	…	…
2732	`</source>`	`</source>`
	…	…
2864	`<source concept="Principle">`	`<source concept="Principle">`
	…	…
2909	`<target concept="Principle" relations="cgnos" />`	`<target concept="Principle" relations="cgnors" />`
	…	…

```
2927        </source>                                                   </source>
            …                                                           …
3124        <source concept="Requirement">                              <source concept="Requirement">
                …                                                           …
3147            <target concept="Constraint" relations="cgnos" />           <target concept="Constraint" relations="cgnors" />
                …                                                           …
3173            <target concept="Requirement" relations="cgnos" />          <target concept="Requirement" relations="cgnors" />
                …                                                           …
3187        </source>                                                   </source>
            …                                                           …
4034    </relationships>                                            </relationships>
```

Appendix 2: Translation of each KAOS concept to ArchiMate

Goal – A goal in KAOS is defined as a 'prescriptive assertion capturing some objective to be met by cooperation of agents; it prescribes a set of desired behaviours' (Respect-IT, 2007). The most obvious ArchiMate concept to choose would be that of a goal, which is however defined slightly differently as a 'high-level statement of intent, direction, or desired end state for an organization and its stakeholders' (The Open Group, 2019).

Obstacle – An obstacle in KAOS is defined as a 'condition (other than a goal) whose satisfaction may prevent some goal(s) from being achieved; it defines a set of undesired behaviours' (Respect-IT, 2007). The authors of ARMOR write (Engelsman, Quartel, Jonkers, & van Sinderen, 2011):

> The obstruction of goals by obstacles are not modelled as part of a goal model in ARMOR. An obstacle is considered the result of the assessment of some stakeholder concern, like the assessment of an influencer as a threat or weakness in the BMM. The modelling of assessments should however be supported by ARMOR – not as part of the goal domain – but as part of the stakeholders domain.

An assessment in ArchiMate is defined as representing 'the result of an analysis of the state of affairs of the enterprise with respect to some driver' (The Open Group, 2019). The definition seems to have little in common with that of an obstacle in KAOS. However, the description following the definition makes the similarities clearer: 'An assessment may reveal strengths, weaknesses, opportunities, or threats for some area of interest. These need to be addressed by adjusting existing goals or setting new ones, which may trigger changes to the Enterprise Architecture' (The Open Group, 2019).

Requirement – A requirement in KAOS is defined as a 'goal assigned to an agent of the software being studied' (Respect-IT, 2007). Again, there is an obvious choice to make, namely for that of ArchiMate's requirement concept, which is defined as a 'statement of need defining a property that applies to a specific system as described by the architecture' (The Open Group, 2019). Like a goal, a requirement in KAOS can be refined, be obstructed by obstacles, and resolve obstacles.

Expectation – An expectation in KAOS is defined as a 'goal assigned to an agent in the environment' (Respect-IT, 2007). In ARMOR the 'concept of expectation is not supported explicitly, but can be modelled as a special type of requirement, i.e., one that can be assigned to an environment actor' (Quartel, Engelsman, Jonkers, & Van Sinderen, 2009). We would rather not use the requirement element in ArchiMate since an expectation does not represent a 'statement of need defining a property that applies to a specific system as described by the architecture' (The Open Group, 2019).

Luckily, ArchiMate contains a specialization of the requirement element, namely the constraint element, which is defined as a 'factor that limits the realization of goals' (The Open Group, 2019). An expectation is a kind of constraint because it prescribes an assumption that must be made by the system designers about the behaviour of an agent in the environment. The expectation thereby limits the ways in which goals can be realized.

Domain property – A domain property in KAOS is defined as follows:

> Descriptive assertion about objects in the environment of the software. It may be a domain invariant or a hypothesis. A domain invariant is a property known to hold in every state of

> some domain object, e.g., a physical law, regulation, … A hypothesis is a property about some domain object supposed to hold. (Respect-IT, 2007)

D. Quartel et al. write (Quartel, Engelsman, Jonkers, & Van Sinderen, 2009):

> The refinement of some goal may be based on certain assumptions about (elements in) the problem domain. i* and KAOS introduce the notions of assumption, belief and domain property for this purpose. Since it is considered useful to make such assumptions explicit, ARMOR supports the general notion of 'assumption'.

In ARMOR an assumption is represented as an attribute of the goal concept. However, ArchiMate's default notation does not support attributes.

The notions of *expectation* and *domain property* have similar definitions in KAOS. A domain property is defined as a 'descriptive assertion about objects in the environment …' (Respect-IT, 2007). In KAOS agents are (active) objects. An expectation is defined as a 'goal assigned to an agent in the environment' (Respect-IT, 2007).

Because an expectation could be said to be a special kind of domain property, we choose to translate the domain property the same as the expectation, namely as a constraint in ArchiMate. In this sense, a domain property is an assumption that the system designers must make about objects in the environment, again limiting the ways in which goals can be realized (see the definition of *constraint* in ArchiMate).

Agent – An agent in KAOS is defined as follows (Respect-IT, 2007):

> Active object (=processor) performing operations to achieve goals. Agents can be the software being considered as a whole or parts of it. Agents can also come from the environment of the software being studied; human agents are in the environment.

This definition distinguishes between internal and external agents, and assumes that we are designing software. Therefore, we should use an internal active structure element from ArchiMate's application layer to represent an internal agent. The obvious element for this purpose is an application component, which is defined as an 'encapsulation of application functionality aligned to implementation structure, which is modular and replaceable' (The Open Group, 2019). The external agent can be an internal active structure element from any of the ArchiMate layers, including a business actor, application component and device. An internal active structure element is defined as 'an entity that is capable of performing behavior'. In KAOS an agent is a specialization of an object and can be the input or output of an operation. This is not possible in ArchiMate. Therefore, an agent is also translated to a passive structure element in ArchiMate, whenever we need to consider its passive side.

Operation – An operation in KAOS is defined as something that 'specifies state transitions of objects that are input and/or output of the operation' (Respect-IT, 2007). Since these operations are performed by internal agents, we must use an appropriate element from the application layer, which in this case is the application process, defined as a 'sequence of application behaviors that achieves a specific result' (The Open Group, 2019).

Event – An event in KAOS is defined as an 'instantaneous object (that is, an object alive in one state only) which triggers operations performed by agents' (Respect-IT, 2007). Note that the word 'object' in KAOS does not imply passivity (as it does in ArchiMate), since agents are defined as 'active objects'. The KAOS Tutorial (Respect-IT, 2007) further specifies: 'Events can be external or produced by operations.' We therefore choose to translate this concept to the abstract ArchiMate concept of event, which can be a business event, application event, a technology event, or an implementation event. An event is defined as representing 'a state change' (The Open Group, 2019). The specification further explains that events can trigger behaviour, which nicely mirrors the triggering of operations by events in KAOS.

Entity – An entity in KAOS is defined as an 'autonomous object, that is, the definition of which does not rely on other objects' (Respect-IT, 2007). Entities are passive objects in KAOS. This corresponds to the notion of a passive structure element in ArchiMate, which is defined as 'an element on which behavior is performed' (The Open Group, 2019). A passive structure element is an abstract element in ArchiMate and can be a business object, contract, representation, data object, artifact, or material element.

Attribute – ArchiMate is not meant to be used for detailed information modelling. It has therefore no in-built support for attributes. Attributes are important in KAOS because their values define the states the entity can transition to. We can choose to represent an attribute as a passive structure element with a composition relation to its parent. We choose the composition relation because it expresses an existence dependency: the attribute cannot exist without its parent.

Refinement – A refinement in KAOS is defined as follows:

> Relationship linking a goal to other goals that are called its subgoals. Each subgoal contributes to the satisfaction of the goal it refines. The conjunction of all the subgoals must be a sufficient condition entailing the goal they refine. (Respect-IT, 2007)

In ARMOR the realization relationship is used to represent refinement. This relationship type expresses the sufficient condition in KAOS nicely since 'the interpretation of a realization relationship is that the *whole or part* of the source element realizes the *whole of* the target element' (The Open Group, 2019). However, the authors of ArchiMate have decided not to permit realization relationships between goals. They write that the 'refinement of goals into sub-goals is modelled using the aggregation relationship' and that 'to refine requirements into more detailed requirements … the aggregation relationship is used' (The Open Group, 2019). Version 2.1 of the ArchiMate specification (The Open Group, ArchiMate® 2.1 Specification, 2013) provides an example of goal refinement using the aggregation relationship, see Figure 28.

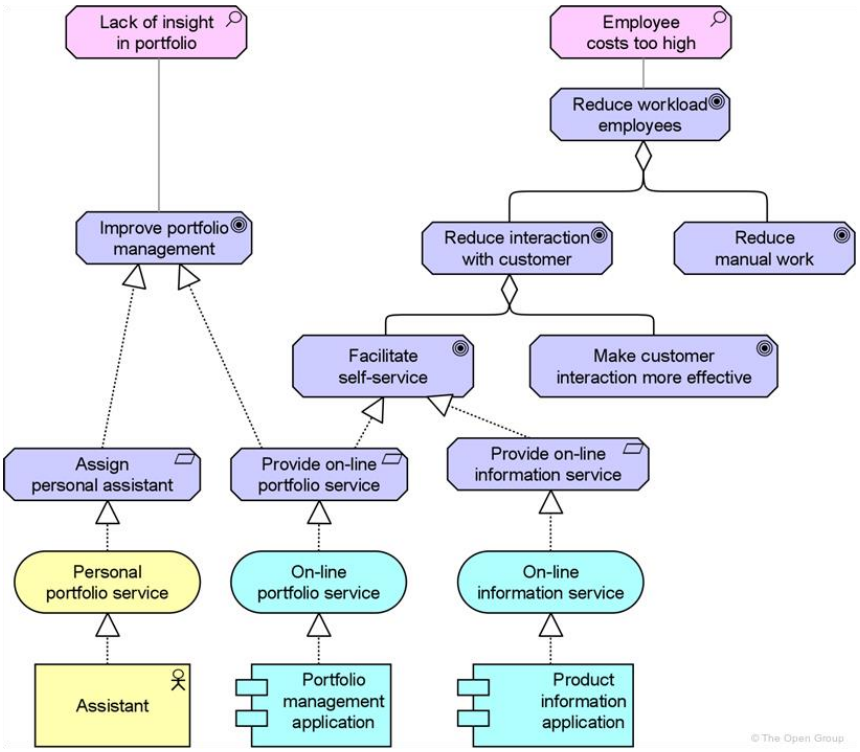


Figure 28. An example of goal refinement using aggregation. From ArchiMate 2.1 Specification*, by The Open Group, 2013.*

We have several reasons not to use the aggregation relationship to express goal refinement.

1) The aggregation relationship does not express the 'sufficient condition' in KAOS's definition of refinement.
2) The aggregation relationship cannot be used to express refinement alternatives as shown in Figure 29.
3) The aggregation relationship cannot be used to link between a mixed collection of motivation elements of different types as shown in Figure 30.

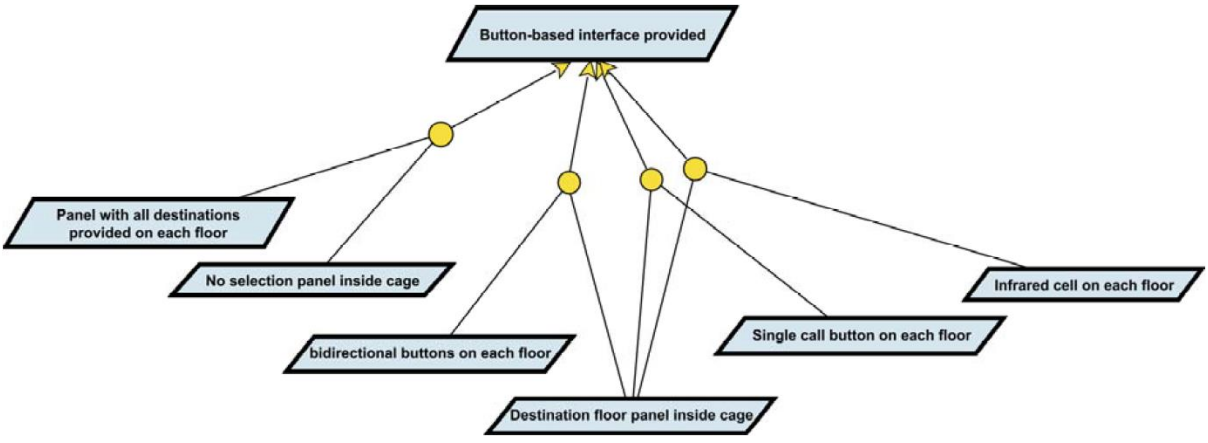


Figure 29. Goal refinement alternatives, from A KAOS Tutorial, V1.0*, by Respect-IT, 2007.*

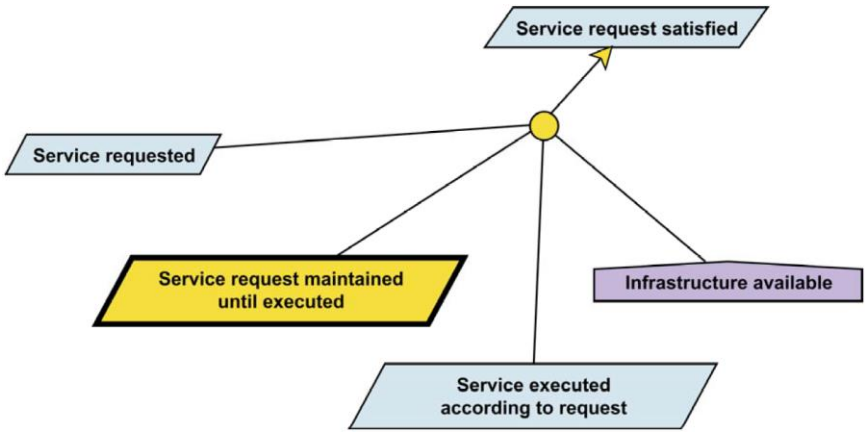


Figure 30. A generic goal refinement pattern, from A KAOS Tutorial, V1.0, by Respect-IT, 2007.

The only relationships in ArchiMate that are allowed between motivation elements and that can be routed through junctions, are the influence and association relationships. Since in general a stronger relationship is preferred over a weaker one, we choose the influence relationship over association.

Unfortunately, in contrast to the realization relationship, the influence relationship cannot be used to express sufficient condition in KAOS.

> The interpretation of a realization relationship is that the *whole or part* of the source element realizes the *whole of* the target element (see also Section 5.1.5). This means that if, for example, two internal behavior elements have a realization relationship to the same service, either of them can realize the complete service. If both internal behavior elements are needed to realize, the grouping element or an *and* junction (see Section 5.5.1) can be used. For weaker types of positive, neutral, or negative contribution to the realization of a motivation element, the influence relationship (see Section 5.2.3) should be used. (The Open Group, 2019)

This means that if we were to depict goal refinement using the influence relationship, as we have done in Figure 31, we are expressing that goals B and C can only influence goal A in conjunction, but not separately. While what we intend to express is that goals B and C in conjunction sufficiently realize goal A, whether or not they individually influence goal A.

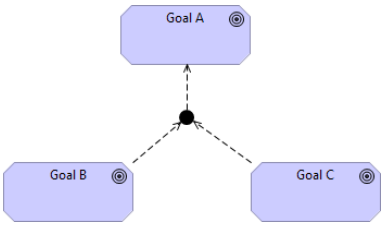


Figure 31. Goal refinement using influence relationships

Since none of the existing relationships in ArchiMate can be used to express goal refinement in a satisfactory manner, we must customize the language by creating a specialization of the influence relationship, which we shall name *refinement*, with the following definition: *The refinement relationship represents that an entity completely satisfies the element it refines*. Table 8 lists the permitted relations of this kind.

Table 8. Allowed refinement relationships

from → ↓ to	Goal	Outcome	Principle	Requirement	Constraint
Goal	refinement	refinement	refinement	refinement	refinement
Outcome		refinement	refinement	refinement	refinement
Principle			refinement	refinement	refinement
Requirement				refinement	refinement
Constraint				refinement	refinement

Notice however that, for example, between a requirement and a goal element, both a realization and a refinement relationship is now possible, with virtually no difference in semantics.

In my opinion, with this customization we have 'hacked' the ArchiMate language by introducing a relationship that is virtually identical to the already existing realization relationship, with the only difference that it allows relations between instances of goal modelling elements of the same type.

I am therefore of the opinion that ArchiMate should allow realization relationships between instances of goal modelling elements of the same type.

Note that we have chosen to translate a KAOS obstacle to an ArchiMate assessment and that ArchiMate does not allow realization relationships in combination with an assessment element. Luckily, we have little use for that relationship here since it is not necessary to express the 'sufficient condition' when refining obstacles. 'Obstacles can be refined the same way we do with goals, but while goals are generally 'AND-refined', obstacles are most often 'OR-refined' (Respect-IT, 2007).

Conflict – A conflict in KAOS is defined as follows: 'Goals are conflicting if under some boundary condition the goals cannot be achieved altogether' (Respect-IT, 2007). For this purpose, we can use the negative influence relation in ArchiMate. The influence relation is defined as representing 'that an element affects the implementation or achievement of some motivation element' (The Open Group, 2019). A negative influence relationship represents the fact that 'an element negatively influences – i.e., prevents or counteracts – such achievement'.

Obstruct – An obstacle is related to the goal it obstructs. In ArchiMate we can reuse the negative influence relation to represent obstruction.

Resolve – An obstacle can be related to a goal that resolves the obstacle. Again, we can use the negative influence relation from ArchiMate to represent that a goal or requirement prevents or counteracts the achievement of the obstacle.

Responsibility – A responsibility relationship in KAOS is defined as follows:

> Relationship between an agent and a requirement. Holds when an agent is assigned the responsibility of achieving the linked requirement. (Respect-IT, 2007)

Elsewhere, A KAOS Tutorial specifies (Respect-IT, 2007): '**Assignment** is used when several agents may be made responsible for some requirement or expectation, whereas **responsibility** is used when there's only one agent who is responsible for it' (p. 8). In ArchiMate we can use the influence relationship to express assignment, and the realization relationship to express responsibility. The realization relationship expresses full responsibility because the 'interpretation of a realization

relationship is that the *whole or part* of the source element realizes the *whole of* the target element' (The Open Group, 2019). The influence relationship expresses a weaker type of contribution to a motivation element.

ArchiMate also contains an assignment relationship to express responsibility, but this relationship is not allowed in combination with a motivation element.

Perform – An agent is linked to an operation with a performs-relation. The default relationship in ArchiMate to express this link is the assignment relationship, defined as the 'allocation of responsibility, performance of behavior, storage, or execution' (The Open Group, 2019). However, if the agent and operation belong to different layers, we can use the serving relationship.

Cause – An event is linked to an operation with a cause-relation to express that an event can start (or stop) an operation. The obvious choice is to use the triggering relationship in ArchiMate, which is defined as a 'temporal or causal relationship between elements' (The Open Group, 2019).

Operationalization – An operationalization relation in KAOS is defined as follows:

> Relationship linking a requirement to operations. Holds when each execution of the operations (possibly constrained to that intent) will entail the requirement. Makes the connection between expected properties (goals) and behaviours (operations). (Respect-IT, 2007)

The authors of ARMOR write: 'The realization relation of ArchiMate is used to represent refinement and to link a requirement to design artefacts, such as the services and processes that implement the requirement' (Quartel, Engelsman, Jonkers, & Van Sinderen, 2009). The interpretation of this relationship is that it realizes the *whole* of the target element. If we were to link both a service and a process to a requirement using a realization relationship, this would express that we can do without one or the other, since both completely realize the requirement. This is often not the case. We could solve this by using an *and* junction to express that *both* elements are needed to realize the requirement. But this is not practical, especially when the services and processes are modelled in different views. In KAOS there is also no need to express 'complete operationalization' in this way. We therefore opt for the weaker relationship, namely influence.

Input/output – An input/output relation in KAOS represents that an object is the input or output of an operation. The strongest relation between a behaviour element and a passive structure element is the access relationship, defined as representing 'the ability of behavior and active structure elements to observe or act upon passive structure elements' (The Open Group, 2019). The specification further explains that 'at the metamodel level, the direction of the relationship is always from an active structure element or a behavior element to a passive structure element, although the notation may point in the other direction to denote "read" access, and in both directions to denote read-write access'. We will use this notation to represent input and output.

Since events are defined as objects in KAOS, they too can be the output of an operation. In ArchiMate we have the triggering relationship for this.

Relations between entities – The notation in the object model in KAOS complies with UML class diagrams. ArchiMate too has relations that are based on UML. Therefore, we can quite easily choose

to represent the KAOS aggregation relation as an ArchiMate aggregation relation. The same applies to specialization and association.

Concern – A concern relation in KAOS is described as a relation that is 'used to link a requirement to the objects that are needed for it to be satisfied' (Respect-IT, 2007). The strongest possible relation in ArchiMate between a business object and a goal or requirement is realization. This relationship is too strong since the mere existence of the business object does not guarantee the satisfaction of the requirement or goal. We therefore opt for the weaker version, namely the influence relationship. See also our considerations above, for the translation of operationalization.