Adaptive to change? Compositional modelling of Healthcare Insurances using Protocol Modelling

Reducing the impact of coverage changes

Author	Jaco Verheul
Student Number	835971801
Date	13 September 2011

Adaptive to change? Compositional modelling of Healthcare Insurances using Protocol Modelling

Reducing the impact of coverage changes

Open Universiteit Nederland, faculteiten Managementwetenschappen en Informatica
Masteropleiding Business Process Management and IT
Course Code T89317 Afstudeertraject Business Process Management and IT

Author Jaco Verheul
Student Number 835971801
Date 13 September 2011

Graduation Committee
Chair Dr. E.E. Roubtsova
Second Reader Prof. Dr. Ir. Stef Joosten
Supervisor Dr. E.E. Roubtsova

Abstract

This report presents the results of research on application of Protocol Modelling in the domain of healthcare insurance claims processing in the Dutch context.

Protocol Modelling is claimed to enable "Evolvable Behaviour Modelling" (McNeile & Roubtsova, 2009). The goal of this research is experimental validation of this claim on the factual material available in the domain of healthcare insurances and identification of semantic constructs that reduce the impact of changes on a protocol model.

The research method is based on the analysis of the literature on product evolution and configuration management in general, transactional properties of Protocol Modelling in particular and application of all findings for building an executable Protocol Model of healthcare insurance applications. The initial version of the Protocol Models is based on the requirements of the Base Insurance from 1 January 2006 and then the model is extended with all changes in the rules of the Base Insurance introduced in the last six years from 2006 to 2011.
Nine use cases are defined that cover most of the functionality of the Base Insurance. All these changes are exposed to the model. Their impact on the model structure and behaviour is accessed and classified on the basis of the model structure that clearly reflects the problem domain.

All the changes fall into four categories:

1. Change in covered care procedures. Changes in covered care procedures occur each and every year.
   Two subclasses are defined:
   a. Coverage Added. A care procedure that was uncovered previously, has become a covered procedure.
   b. Coverage Removed: A care procedure that was covered previously, has become uncovered.
2. Condition Changed: A care procedure that was covered before, is still covered but the conditions for coverage have become more restrictive or less restrictive.
   Changes in conditions occur in most years. Most changes concern changes in age limits.
3. Change in benefit calculation. The algorithm to calculate the benefit amount of a claim has changed. Change in benefit calculation happens once: when the mandatory deductible is introduced in 2008. In later years only the deductible mount is increased.
4. Other Change: Changes that do not belong to one of the three categories above. The only change of this type is the introduction of a personal budget for visual aids. This is outside the scope of the model as this change does not impact claims processing.

For each class of the changes, the impact on the model is assessed. All change types can be implemented by changing the model configuration only, except for the introduction of the new concept of a mandatory deductible. The initial model is extended to support mandatory deductibles by adding new behaviours and attributes utilizing the composition semantics of protocol modelling. Only the ProcessClaim callback code had to be modified, to handle the additional step of deductable calculation.

The conclusion of this study is that all necessary elements of flexibility for a healthcare insurance model, namely, flexibility in addition and removal of covered care procedures, flexibility in conditional coverage and flexible deductible, are supported by the protocol

model by such means as Parameterization, User Exits, Composition, Derived Attributes and States. The first two options can also be achieved by other modelling techniques. The third option of CSP parallel composition leverages the composition semantics of Protocol Modelling. Composition, derived states and attributes enable the reuse of model elements.

The semantic constructs of protocol modelling can best be applied in a model of a healthcare insurance by:

- Using parameterization in the definition of Product, Coverages and Conditions.
- Using User Exists to enable different price- and benefit calculation algorithms per coverage object.
- Using Composition to include different combinations of conditions in coverage objects.
- Using derived attributes to abstract over a range of history records, like deductable consumption.

It is expected that the results of this research also apply to other healthcare insurances as the developed model has abstracted from the Base Insurance by applying generalization. It might also apply to other types of insurance that are similar to healthcare insurance.

Contents

1. Introduction

This report presents the results of research on the application of Protocol Modelling in the domain of healthcare insurances in the Dutch context. The developed model supports the processing of submitted healthcare insurance claims.
Protocol Modelling is claimed to enable "Evolvable Behaviour Modelling" (McNeile & Roubtsova, 2009). The goal of this research is to validate this claim in the domain of healthcare insurances.

An evolvable model of healthcare claims processing must be able to cope with different changes:

- Changes in coverage of healthcare costs. These changes are often prescribed by law.
- Differences between customers. This is especially true when modelling the behaviour of an application sold as a product. The application and hence also the model, must be flexible enough to handle differences between customers.

Therefore the research concentrates on the semantic constructs that reduce the impact of changes on a model of healthcare claims processing.

The research question is:
Which semantic constructs reduce the impact of changes on a protocol model of a healthcare insurance?

This research question is detailed in the following sub questions:

1. Which flexibility is needed for a healthcare insurance model? In other words: which types of changes occur in the healthcare insurance domain?
2. Which semantic constructs of Protocol Modelling support the needed flexibility?
3. How can the semantic construct best be applied?

The research has the form of a case-study:

- After describing the theoretical framework, a model of the Base Insurance as of 1 January 2006 is created. At that time, the Base Insurance started in the Netherlands.
- After creating the model, all changes in the rules of the Base Insurance since 2006 until and including 2011 are exposed to the model. Their impact is assessed and classified.
- The results of this assessment are discussed. The initial model is enhanced with necessary structural changes. In the concluding chapter, the research questions are answered.

The Dutch Base Insurance (Basisverzekering) is taken as a use case because many of the customers of Oracle Health Insurance are in the Netherlands. Also the rules of the Base Insurance are clearly defined. All Dutch inhabitants are required to enrol into the Base Insurance.

The report is constructed as follows:

1. Chapter 2 "Background" presents some background information.
2. Chapter 3 "Research Framework" describes the actions taken to answer the research question. It also presents the phases of the research project and the research design.
3. Chapter 4 "Theoretical Framework" presents the results of the literature review. On the one hand it outlines the flexibility needed for product software and explains why this flexibility is also needed in a model of a software product. On the other hand, the semantics of Protocol Modelling are reviewed. The last section shows how the required flexibility can be constructed using Protocol Modelling.

4. Chapter 5 “Design Protocol Model for Base Insurance 2006” uses the result of the literature review to create an initial model for the Base Insurance as of 1 January 2006.
5. Chapter 6 “Results and analysis” classifies all changes in the rules of the Base Insurance since 2006. The impact of all change types is determined and assessed.
6. Chapter 7 “Model Enhancements” extends the initial model with the support of a Mandatory Deductible as introduced in 2008. The impact of this structural change on existing elements of the initial model is described.
7. Chapter 8 “Conclusions and Discussions” presents conclusions and answers the research questions. The validity of the results is discussed. Suggestions for additional research are presented.

2. Background

The Oracle Health Insurance (OHI) division currently develops a new application to process healthcare insurance claims. This application is called *OHI Next*.

OHI Next is a replacement of an existing 20-year old application with the following characteristics:

1. Originally developed with a rule-based architecture to support several regional healthcare insurance companies. Lots of configuration changes can be made by setting parameters. This allows for differences between customers and law changes.
2. Developed and sold as a product. Does not contain customer specific modules.
3. Developed as a monolith.
4. Starting from 2006, multilingual and multi-country support has been added to the application. Since then, the first customers abroad have implemented the application successfully.

Typical for the domain of healthcare insurances is the large influence of the law. These laws differ substantially between countries. This restricts the number of countries the existing application can serve: only the countries for which the healthcare systems are more or less similar to the Dutch system.
The wish to have customers in major markets like US, Germany and France is one of the drivers to develop the new application.

In the remainder of the research, the development of the new application does not play a role anymore, nor are the results of this research used for or dependent on the development of the new application.
The initial release of the new application only supports the United States market, whereas the developed model is for the Dutch Base Insurance.
Only chapter 4 outlines which roles a model could play during the development and implementation of product software.

The author is a member of the development team of OHI Next.

3. Research Framework

This chapter describes the actions taken to answer the research questions. The research framework is shown in figure 1. The arrows show how the results of an action are used in subsequent actions.

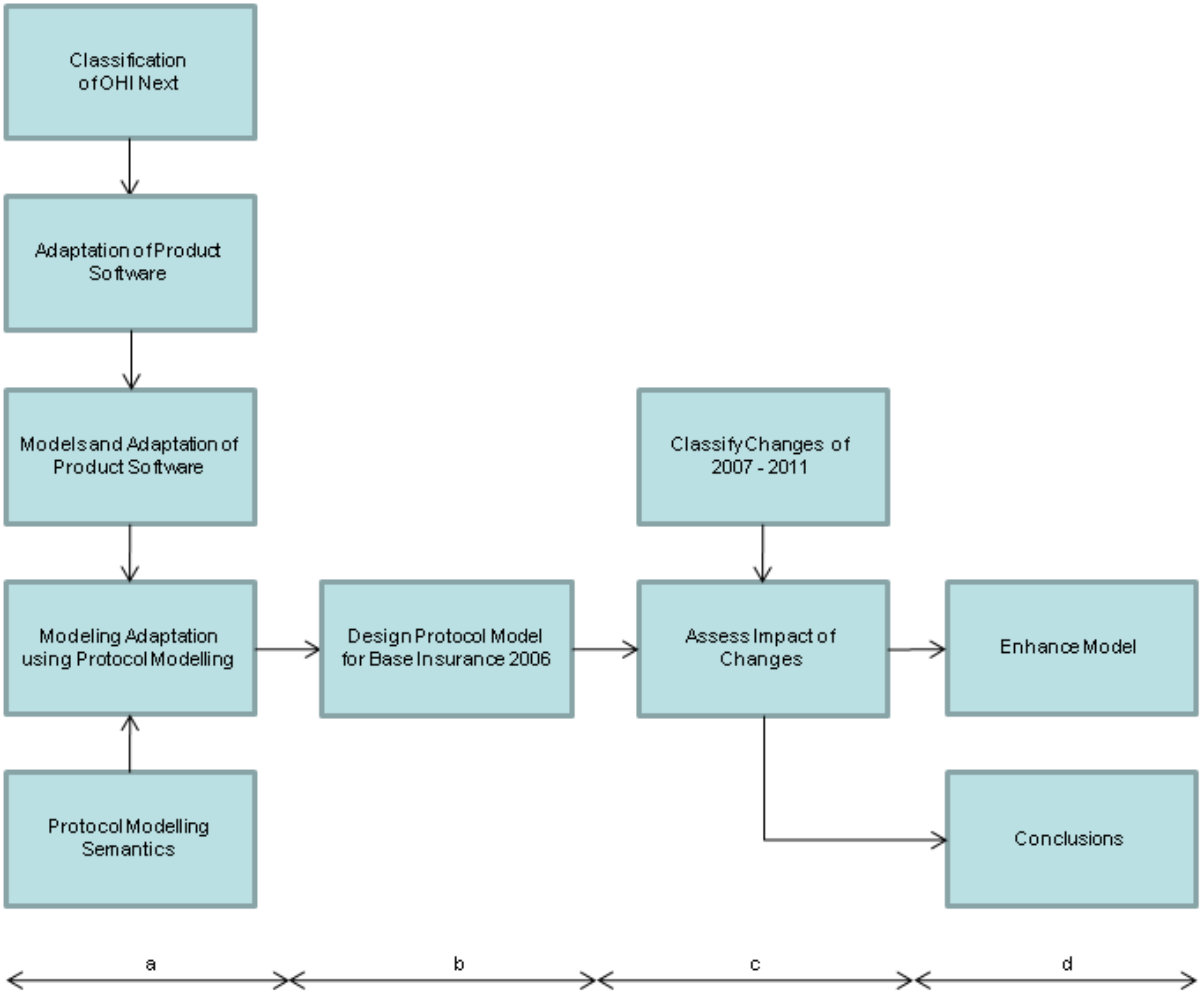


Figure 1 Research Framework

The actions are executed in four phases:

a) "Literature Review" contains five actions. Each action is described in its own section in chapter 4 "Theoretical Framework".
    a. Classification of the OHI Next application. Determines the characteristics of the class of applications OHI Next belongs to.
    b. Adaptation of Product Software. Describes various options to make software products adaptable.
    c. Models and Adaptation of Product Software. Describes the role of models during development and adaptation of product software
    d. Protocol Modelling Semantics. Gives a summary of the semantics of Protocol Modelling.
    e. Modelling Adaptation using Protocol Modelling. Shows how techniques from Adaptation of Product Software can be implemented using Protocol Modelling.

b) "Design Protocol Model". A model of the Dutch Base Insurance (Basisverzekering) is constructed. The Base Insurance was introduced in the Netherlands in 2006. So the constructed model reflects the rules as they were at 1 January 2006. The model is described in detail in chapter 5 "Design Protocol Model for Base Insurance 2006".

c) "Assess Impact of Changes". Yearly, the rules of the Base Insurance change. In this phase, all changes from 2007 until 2011 are classified. Their impact on the model is assessed. The results of the assessment are described in chapter 6 "Results and analysis".
d) "Results". The results of c) are used to enhance the model and to give an answer to the research questions. See chapters 7 "Model Enhancements" and 8 "Conclusions and Discussions".

3.1. Research Design

The research has the form of a case study: only the Dutch Base Insurance is studied. This is caused by the explorative nature of the research:

- No reference models of the healthcare insurance domain were found.
- No guidelines exist how to achieve flexible and adaptable protocol models, even though some papers talk about evolvable models (McNeile & Roubtsova, 2009).

The Base Insurance is a healthcare insurance for which the coverage is prescribed by Dutch law. All inhabitants of the Netherlands are required to enrol for the Base Insurance.
The situation of 1 January 2006 is taken as starting point. At that time, the Base Insurance was first introduced in the Netherlands.
The Base Insurance is taken as a research case because:

- The coverage rules are prescribed by Dutch law and implemented by all insurance companies in the Netherlands.
- The Base Insurance contains aspects common to other healthcare insurances. It is expected that results for the Base Insurance will also be valid for other healthcare insurances.
- The coverage rules of the Base Insurance are publicly available on the Internet, including all changes from 2007 until 2011. So using the Base Insurance as a case saves analysis time.

4. Theoretical Framework

This chapter presents the results of the literature review. Each section covers one of the actions shown in chapter 3 “Figure 1 Research Framework”

4.1. Classification of OHI Next

This section classifies the OHI Next application to get insight in its characteristics.
Milicev (Milicev, 2009) gives the following definition of *Information Systems* (IS):
An information system is a computer-based system primarily dealing with large amounts of data that are structured, stored, transferred, processed, and presented ... to accomplish a specific purpose for the users.
According to this definition, OHI Next is an Information System. Milicev gives characteristics of Information Systems, divided in three categories:

1. Domain related:
    a. Inherent complex functionality.
    b. IS concepts based on domain concepts.
    c. Instantiation of large number of objects.
    d. Evolution of functionality.
2. Usability:
    a. Interactive
    b. Gives relevant information at correct time and place
    c. Security.
    d. Ease of use.
3. Deployment related:
    a. Large amounts of data of different type.
    b. Scalable.
    c. Persistent.
    d. Concurrent access.
    e. Distributed processing.

All these characteristics apply to OHI Next.

Wortmann and Kusters define *Enterprise Information Systems* (EIS) as:
An Enterprise Information System is an information system which supports human activities in organisations, even if these activities are performed by stakeholders outside the traditional organisational boundaries.
An EIS gives users the following capabilities:

1. Storage of data and mutations.
2. Functionality suited to their jobs.
3. User support, so that a user can use the functionality.
4. Authorisation.

Al these characteristics also apply to OHI Next.

Enterprise Resource Planning (ERP) systems are a subclass of EIS. According to Wortmann and Kusters, ERP systems are increasingly positioned as “transaction processing backbones”. This is exactly the role of OHI Next: processing claims that are delivered electronically or on paper.
Information Systems for businesses are called *Business Information System.*

Roubtsova, Wedemeijer, Lemmen and McNeile (2009) define *Service Providing Business Processes* (SPBP) as:

A Service Providing Business Process (SPBP) is an interactive process that transforms the requests of users and the information presented in rules, law regulations or databases of official organizations into a physical product or a document.
Based on the information present in the rules, a SPBP can accept or refuse a request. This is exactly what OHI Next does: a submitted claim is either paid (partly) or rejected. In our case, the rules are the rules of the Base Insurance:

To summarize:

- OHI Next is an Information System.
- OHI Next is an Enterprise Information System that functions as a transaction processing backbone.
- OHI Next supports Service Providing Business Processes.

4.2. Adaptation of Product Software

OHI Next is sold as a standard application to different healthcare insurance companies worldwide. The application has not been developed for a specific customer, but for a market. Being a standard application for a worldwide market, gives additional requirements. Those requirements are described in this section.

1. Section "Product Software" gives a definition of *product software.*
2. Section "Flexibility" describes which flexibility is needed.
3. Section "Adaptation" describes terminology and classifications dealing with adaptation of product software.
4. Section "Adaptation of OHI Next" relates the previous sections to the case.

4.2.1. Product Software

In accordance to Xu and Brinkkemper (Xu & Brinkkemper, 2005), instead of standard application the term *product software* is used with the following definition:
"*A software product is defined as a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market".*
The term *product* indicates the difference with custom software.

4.2.2. Flexibility

Product software will be used by different customers. An important aspect of product software is therefore how and to which extends *differences between customers* are supported. It is not likely that business processes will execute exactly identical at different customers. So the product needs to be adapted to the situation for a specific customer.
The product should also be able to handle changes caused by the *passing of time*. For healthcare insurances, these changes often have to do with *changes in the law.*
So an important requirement is *flexibility*: the ability to adapt the product to different and changing circumstances.

4.2.3. Adaptation

Adaptation of product software is described in the literature with different terminology. Authors also differ in their classification of adaptation possibilities. See the table below.
Note: classifications of adaptation in the literature often refer to Enterprise Resource Planning (ERP) applications. ERP applications form an important subcategory of application software, for which much research is done. The results of the research to application adaptation also apply to OHI Next.

Table 1 Adaptation of product software: classification and terminology

Author	Terminology
(Carney, 1997).	Carney talks about *adaptation* and describes the use of *glue code, wrappers* and *bridges* to wire components together.
(Morisio & Torchiani, 2002)	Morisio and Torchiano talk about *customization* and give the following classification: • Adapt program code. • Program using a set of delivered API's and a scripting language. • Defining macros. • Parameterization.
(Ahituv, Neumann & Zviran, 2002).	Ahituv, Neumann and Zivran indicate that flexibility is an important aspect of ERP systems. They define four ways to achieve flexibility: • Setting of parameters. • Changing program source • Adding modules • Connectivity to other systems. Flexibility according to them can give companies competitive advantage, by defining a unique customization of an ERP system.
(Yang, Bhuta, Boehm & Port, 2005)	Yang, Bhuta, Boehm and Port describe *tailoring options*: • GUI operation • Setting of parameters. • Programming of scripts.
(Brehm, Heinzl & Markus, 2000)	Brehm, Heinzl and Markus give the following overview of ERP *tailoring options*: • Configuration: setting of parameters and control data. • Bolt-ons: A predefined ERP configuration for a particular industry. • Screen masks: new entry screens • Extended reporting: extra data output possibilities • Workflow programming: Create different workflows. • User exits: Program extensions using a predefined interface. • ERP programming: adding new modules using the ERP systems programming language. • Interface development: programming interfaces with other systems. • Package code modification: Change the ERP source code itself. The options are given in ascending order of 'impact': how much is the system changed and how much effort is needed?

To summarize:

- Flexibility can be built into a software component in multiple ways.
- The classification of Brehm, Heinzl and Markus is the most elaborate.

4.2.4. Adaptation of OHI Next

OHI Next uses a number of adaptation possibilities (classified according to Brehm, Heinzl and Markus):

- Configuration: setting of parameters and control data. This option is used extensively and forms the basis of OHI Next's flexibility. Coverage rules can be freely defined and supplied with parameters.

- User exits: at certain fixed locations in the system, a piece of custom logic can be called. Custom logic has a predefined interface (input/output definition). Custom logic is used in situations to that are too specific to create configurable parameters for it. An example is the matching of a claim to a required authorization. Customers do this in very different ways. They can implement their algorithm in a piece of custom logic.

4.3. Models and Adaptation of Product Software

This section describes the role of models during development and adaptation of product software.

1. Section "Models during development and adaptation of product software" concentrates on the usage of models in combination with the development and adaptation of product software.
2. Section "Models and OHI Next" draws conclusions and relates the findings to the case.

More generic background information about the usage of models can be found in Appendix 1: Usage of Models".

4.3.1. Models during development and adaptation of product software

In relationship with product software, models can be viewed from two perspectives:

1. From the perspective of the supplier.
2. From the perspective of the customer using the product.

4.3.1.1. Supplier Perspective

Also in the development of product software one can use models, model-driven prototyping and model-driven engineering.
Product software differs from custom software by a greater degree of flexibility. An executable model of product software therefore also needs to have a greater degree of flexibility. In other words: *the adaptation capabilities of the software must also exist in the executable model.*

4.3.1.2. Customer perspective

Wortmann and Kusters describe the use of *enterprise models*. According to them, these models also play a role in the selection, implementation and use of enterprise information systems (EIS). The comparison of enterprise models with models of an EIS assists in the selection of a suitable package.
It is therefore important that the supplier of the EIS provide models. That often comes in the form of a *reference model*. A reference model describes the processes and structure of an EIS and is a description of best practices. A reference model also documents the total functionality of an EIS.
During implementation, the real world must be modelled in the language of the EIS. The implementation decisions taken should be recorded (documentation) and communicated to stakeholders.

4.3.2. Models and OHI Next

This section relates the information of the preceding sections to the case. The development of applications within OHI is partly financed by a launching customer. This launching customer had to be found at the time that the software does not yet exist. Currently, the sales process makes use of electronic presentation material in combination with demonstrations using the old system (with a different structure).
An executable model helps to visualize a new application to customers.

Within OHI, applications are designed by functional analysts. They discuss the requirements with customers and store them in designs. The analysts have very detailed functional knowledge. Construction and technical design is done by Java developers, without functional knowledge.
The transfer of specifications of analysts to developers will benefit from an executable model. The developers can use this model while constructing the software.

The design is delivered electronically to the launching customer for approval in the form of a structural model with textual explanation. It is noticed that representatives of the customer need much additional explanation before they can understand a delivered design. This leads to major delays in the approval process.
An executable model helps explaining designs to users.

During the sales cycle of OHI applications to new customers, often a fit-gap analysis is executed: where fits the software with the requirements, and where is a gap encountered?
An executable model supports to discover and resolve gaps.

The characteristics of OHI are comparable to the characteristics of plan-driven development:

- The requirements are relatively stable.
- The team is big (30+).
- High reliability is required. Errors in the application have direct financial implications.

OHI certainly suffers from a very large problem-implementation gap: the functionality is very complex. The technical architecture uses a unique combination of techniques that exists almost nowhere in the world.
Taking into account the preferences of the current development team, Model-Driven Development is a bridge too far. Model-Driven prototyping with the main goal to improve communication internally and externally is more realistic.

The executable model (prototype) must have the same adaptation possibilities as the product to be developed.
For OHI Next this means:

- Configuration: Changing parameters and control data.
- User exits: define a piece of custom logic to be called from fixed places in the application.

4.4. Protocol Modelling Semantics

This section presents a summary of the semantics of protocol modelling. Aspects used for the construction of the model are emphasized. The information in this chapter is based on (McNeile and Roubtsova, 2009) and (McNeile and Simons, 2006). More background information can be found in these documents.

Protocol modelling concepts are explained using examples of the insurance domain. See "Appendix 4: Explanation of graphical symbols" for an explanation of the symbols used in diagrams.

Protocol Modelling already was in use in the research community, so it was the preferred choice when starting this research. A complete comparison of Protocol Modelling semantics with UML State Machine semantics is outside the scope of this research project, but "Appendix 5: UML Behaviour Models" explains why Protocol Machines are better suited than UML state machines to describe the behaviour of Business Information Systems.

4.4.1. Protocol and event

The meaning of *protocol* in protocol modelling is comparable to the meaning in UML Protocol State Machines (PSM) (OMG (2009-2). UML PSM's model the allowed sequences of operations.
Protocol modelling supports the modelling of allowed sequences of *events*.
For example:

- Event "Submit Claim" is only possible after the event "Create Policy".
- At most four events "Claim hour Nutritional Counselling" are allowed during the lifetime of a policy. (The Base Insurance only covers four hours of Nutritional Counselling).

So a protocol defines *the set of all possible allowed sequences of events*.

Definition of event[1]:
An "event" is the data representation of an occurrence of interest in the real world business domain.
(McNeile and Simons, 2006).

An event has *attributes* and *values*. For example, the event "Create Policy" contains attributes Person, Product, Policy Number and Start Date. An attribute is either a reference attribute (like Person), or a value attribute (like Policy Number). A reference attribute refers to another object[2]. A value attribute has a scalar value like a String or Date.

4.4.2. Protocol machine

Protocol modelling uses state transitions machines, *protocol machine* or *machine* for short.
The behaviour of a protocol machine is defined as follows:

- A protocol machine has an alphabet: the set of events that the machine "understands".
- Events not in the alphabet are ignored.
- Events in the alphabet are either accepted or refused.
- After the processing of an event, the machine can transition to a new state.

[1] This definition is loosely: the difference between event and event instance is ignored.
[2] More precise: a reference attribute refers to a behaviour. See section "Composition".

See diagram below for a graphical representation of the Person protocol machine.

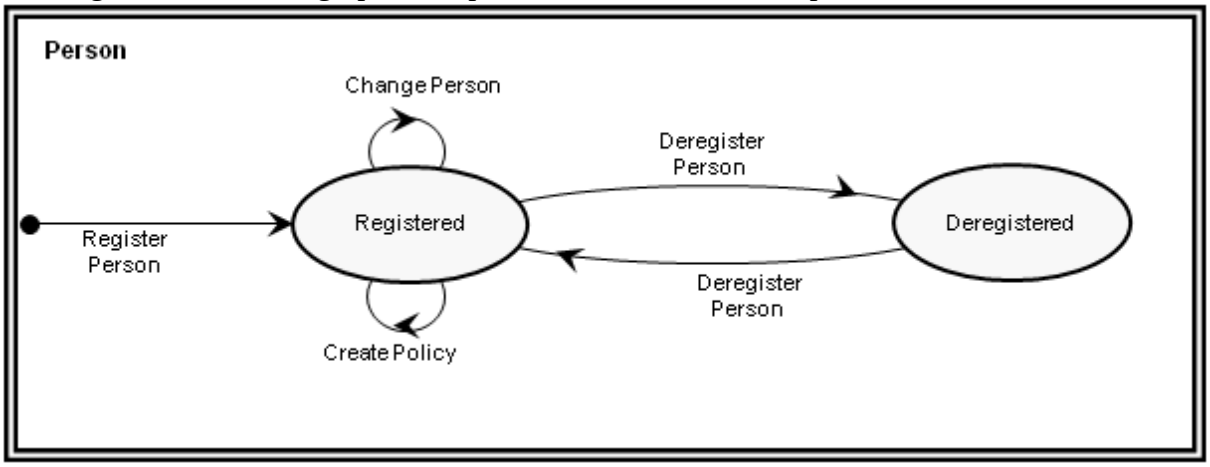


Figure 2 Protocol Machine Person

- The alphabet of this machine consists of the events Register Person, Change Person, Create Policy, and Deregister Person.
- Event Submit Claim is not a part of the alphabet, so the machine ignores this event.
- If a person is in the state Registered, the event Deregister Person is accepted. The new state of the machine will become Deregistered.
- If a person is in the state Deregistered, the event Deregister Person is refused. The state of the machine will not change.

An event is said to be *in context*, if the protocol machine is in a state that accepts the event.

4.4.3. Object modelling

In protocol modelling, an object consists of:

- Attributes. A set of attribute names and attribute values. Similar to events, both reference- and value attributes are possible.
- Behaviour. An object contains one or more protocol machines.

For example:

- The object Person has attributes Name and Date of Birth.
- The object Person contains a protocol machine with states Registered and Deregistered and the alphabet Register Person, Change Person, Create Policy and Deregister Person.

As said above, the state of an object can change when an event occurs. An event can also result in the change of attribute values. Using the ModelScope tool, an attribute change can be implemented in two ways:

1. If the name of an event attribute matches the name of an object attribute, the object attribute gets the value of the event attribute. This is called Name Co-incidence Data Transfer.
2. The event is handled by a Java callback. This is needed for situations where plain name matching is not sufficient.

4.4.4. Composition

Multiple Protocol Machines can be *composed* into a Protocol System. A Protocol System is itself a Protocol Machine with the following behaviour:

- The alphabet of the Protocol System is the union of the alphabets of the composing machines.
- Events not in the alphabet of the Protocol System are ignored.

- Events presented to the Protocol System, are presented to the composing machines.
- Events in the alphabet of the machine are refused if refused by one of the machines.
- In all other cases, the event is accepted.

These characteristics enable composition of behaviour. For example, an insurance company only wants to deregister a person if all outstanding bills are paid. One can add the Protocol Machine Debtor to the Person Object. The diagram below shows the definition of the Debtor machine:

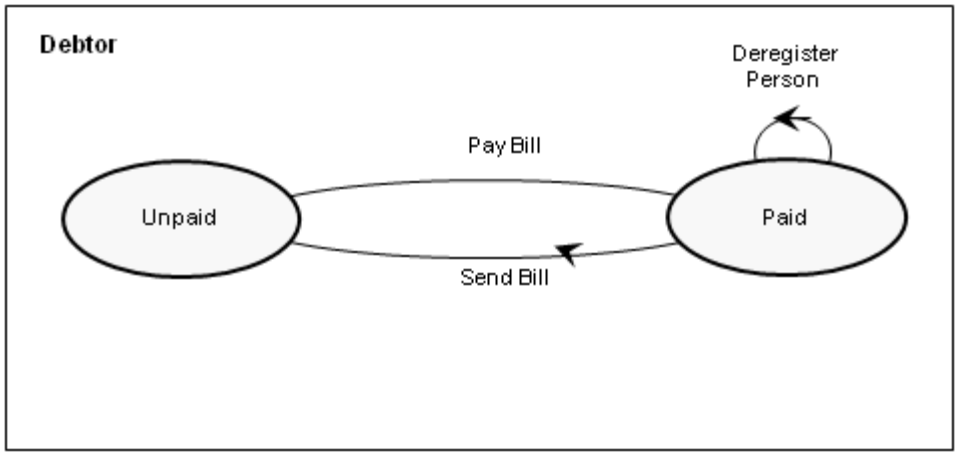


Figure 3 Protocol machine Debtor

The Person object now has a Protocol System composed of the Protocol Machines Person and Debtor. Debtor will refuse the event Deregister Person if in the state Unpaid. According to the composition rules, Person will also refuse the event, so deregistering a person is only possible when the bill is paid.

In protocol modelling, Debtor is called a *Behaviour*. Similar to an object, a Behaviour has attributes and behaviour. However, an object can be instantiated itself, whereas a Behaviour is only instantiated as part of object instantiation.
Composition of behaviours and objects can happen as follows:

- An object can contain multiple behaviours.
- A behaviour can also contain multiple behaviours.

The behaviours are also called mixin's. (McNeile and Simons, 2004).

In theoretical papers like (McNeile and Simons, 2006), the terminology of Protocol Machines and Systems is used. In practical documents like the Modellers' Guide (Metamaxim), one talks about Objects and Behaviours.
These concepts relate as follows:

- A behaviour maps to a Protocol Machine.
- A behaviour containing other behaviours maps to a Protocol System.
- An Object is a Behaviour that can be instantiated. So all Objects are Behaviours, but not all Behaviours are Objects.

4.4.5. Derived Attributes and States

Both an attribute and state can be *derived*. For derived attributes and states, the value is not stored, but computed when needed. Computation in the ModelScope tool is implemented in a Java callback.
Derived attributes and states enable abstraction over a dataset.
For example: the state of the Protocol Machine Debtor can be derived as follows: if the sum of all unpaid bills is greater than zero, the state is Unpaid, otherwise Paid.

When using derived states, the state transition is not triggered directly by an event. Derived states are depicted as follows:

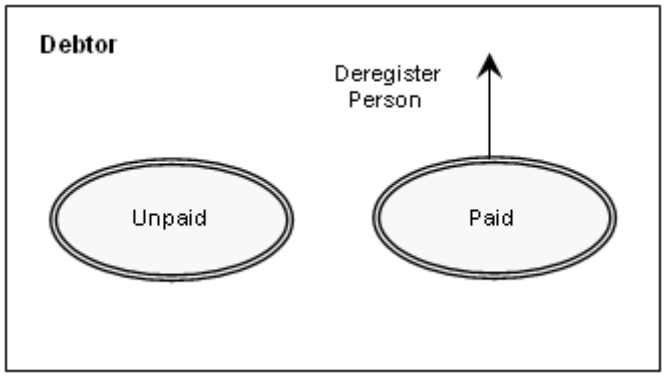


Figure 4 Protocol machine Debtor with derived state

4.4.6. Sub Events

An event is a data representation of an occurrence of interest in reality. Sometimes it is needed to generate multiple model events from the real-world event. These generated model events are called *Sub Events*. See figure 5 from (McNeile and Simons, 2006).

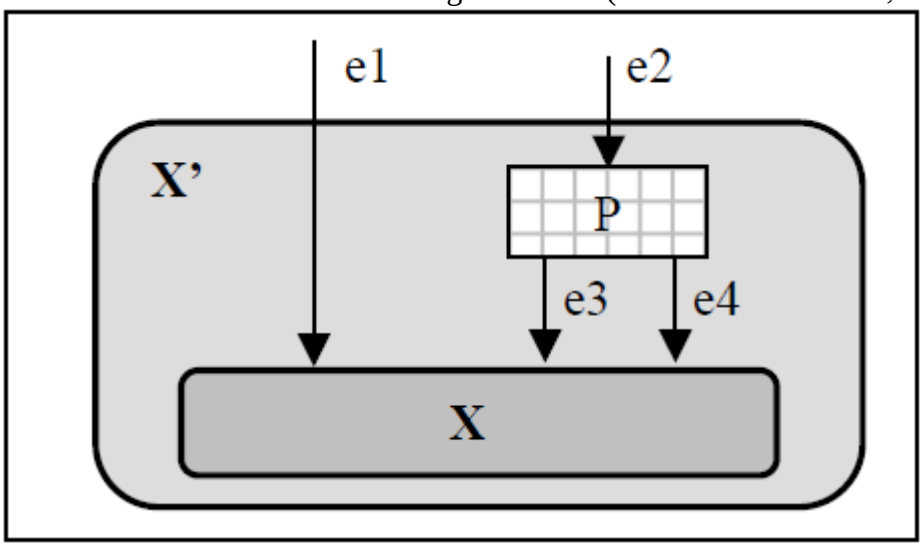


Figure 5 Model extension

Existing model X is extended into X'. Real-world events of type e2 are processes by X'. Process P generates Sub Events e3 and e4 out of e2 and presents them to model X. Event e1 is presented directly to X.
X' is called an *extended model*.

4.4.7. Actors

ModelScope has the concept of *Actors*. An actor represents a group of users of the model. An actor can only see a subset of objects and events of the model.
For example:

- The actor Member can only operate on Policies and Claims.
- The actor Functional Management can only operate on Products and Coverages.

With actors, one can define different views on the same model.

4.4.8. Observations

This section contains some observations regarding the applicability of protocol models for creating an executable model of healthcare insurances.

Protocol Machines are suitable for modelling the behaviour of "transactional business systems". (McNeile & Simons, 2006). In transactional business systems, an event often has impact on multiple objects. Each of these objects may decide to refuse the event. The composition semantics of Protocol Modelling support this, because each object independently may decide to refuse the event. The event is then rejected as a whole. This leads to better encapsulation.
An example from the domain of OHI Next: *a submitted claim relates to an insured person. An insured person has its own state machine. If the state machine of the insured person is in the state Late Payment, then the claim is refused.*

Besides that, Protocol Modelling has derived states. The state of a protocol machine is calculated on demand by the state-function. Derived states "... *increase the expressive power to describe action sequencing protocols that depend on the values of stored data".*

4.5. Modelling Adaptation using Protocol Modelling

This section describes the modelling of product adaptation with protocol modelling. Only the adaptation options used by OHI Next are discussed. See section 4.2.4 "Adaptation of OHI Next".
Code snippets shown in this section originate from the ModelScope tool.

4.5.1. Parameterization

Business rules often have parameters. An example makes this clear: to cut costs, a healthcare insurance company wants to process claims automatically. Claims over 1000 Euro however, should be checked manually. The threshold value of 1000 Euro should be adjustable.

This can be modelled as follows: a parameter object contains the adjustable limit value. See code snippet below:

```
OBJECT Parameter
NAME Name
ATTRIBUTES Name: String,
           Check Limit: Currency
STATES Active
TRANSITIONS @new*Create Parameter=Active,
            Active*Change Parameter = Active
```

The Claim object is extended with a Manual Check behaviour. See picture below:

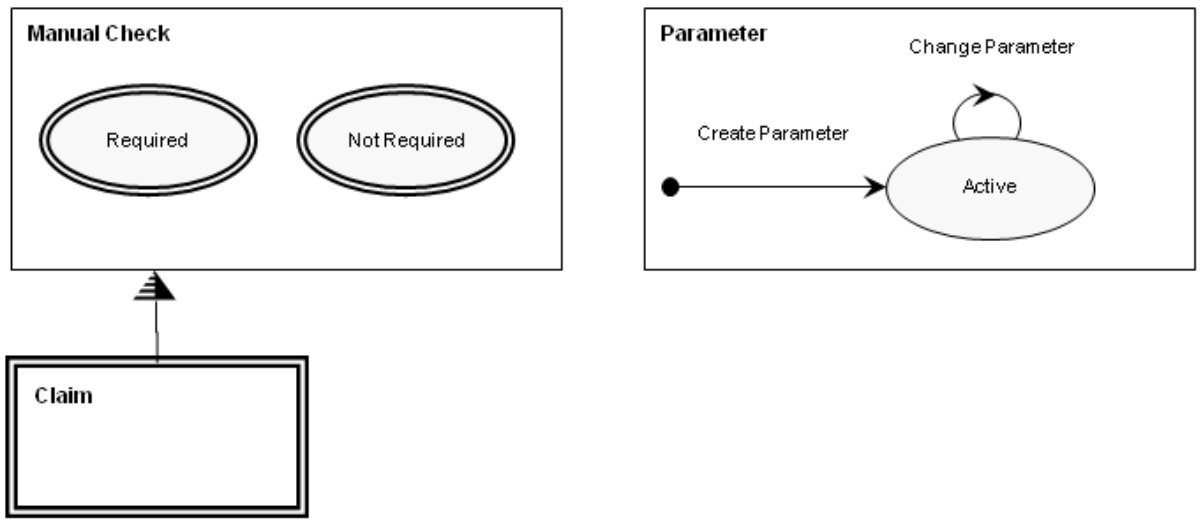


Figure 6 Claim includes Manual Check

The Manual Check behaviour determines whether a manual check of the claim is required. The state of this behaviour is derived with the following callback that compares the claim amount with the Check Limit:

```
public String getState() {
      //Get Active instance of Parameter object
      Instance[] parameters = this.selectInState("Parameter", "Active");
      //Get value of threshold.
      int checkLimit = parameters[0].getCurrency("Check Limit");
      //Get claim amount
      int amount = this.getCurrency("Amount");
      //And compare them.
      return amount >= checkLimit ? "Required" : "Not required";
}
```

So for claims with an amount exceeding the Check Limit, the included behaviour Manual Check will automatically get the state Required. Note that the Manual Check behaviour can be reused in other objects, as long as those objects have an Amount attribute.

To summarize:

- It's straightforward to model parameterization of business rules using Protocol Modelling.
- Parameter objects can be added with parameter values stored as attributes.
- Parameter objects can be retrieved and used in callbacks.

4.5.2. User Exits

With a User Exit it is possible to call a custom piece of code from an application. This possibility must be built into the product beforehand. At predefined spots in the application logic a user exit can be used.
The interface between the application and the user exit is predefined, i.e. the custom code must comply with the predefined input and output contract. See picture below:

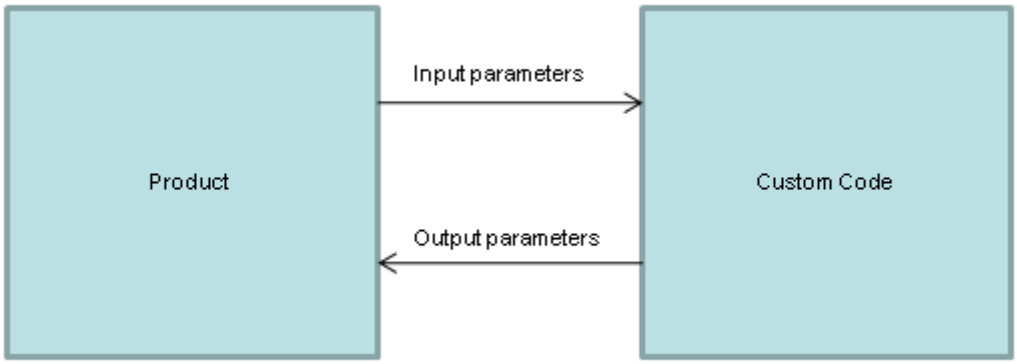


Figure 7 Product with User Exit

OHI Next uses User Exits quite often. An example is fraud-detection: detection of costs that are claimed more than once. Algorithms for fraud-detection are hard to prebuild, because they tend to differ per customer. They are also subject to frequent changes.
So instead of delivering prebuilt fraud detection code, at certain spots in the claims processing User Exits are defined. The User Exit for fraud detection has a claim as input. The custom code should return a list of zero or more duplicate claims.

4.5.2.1. User Exits using Protocol Modelling

It is fairly straightforward to model a User Exit using protocol modelling. See picture below:

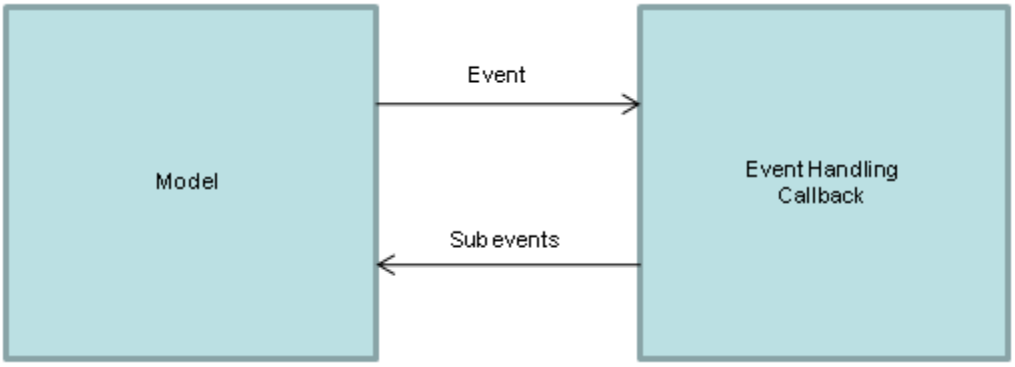


Figure 8 Protocol Model with User Exit

At certain predefined spots in the model, the model submits an event with event arguments. The custom code of the User Exit handles the event and can operate upon the arguments of the event. The custom code may generate model events (Subevents) to update the state of the model.

4.5.3. Reduce impact of changes

This paragraph describes some semantic construct that might reduce the impact of changes on the model.

Composition enables the break-down of larger machines into smaller ones. Composition thus leads to:

- Smaller machines which are easier to understand.
- Smaller machines which can be reused more often.

Let's give an example:
In certain cases a claim must be adjudicated manually. This aspect can be modelled in a separate machine Manual Adjudication with states Required, Not Required and Done. This machine can then be included by all objects for which manual adjudication might be applicable.

By decomposing a model in smaller units, where every unit has only one role, the impact of changes will be reduced, because each feature is implemented in only one place. Only one change to the model is needed when the feature changes.

Derived attributes enable the abstraction over implementation details. A derived attribute can be considered a contract definition, where implementation is hidden from calling client objects.
Changes in the implementation do not influence the clients, as long as the derived attribute behaves the same.

The same applies to *derived states*.

5. Design Protocol Model for Base Insurance 2006

This chapter describes the design and construction of a protocol model of the Base Insurance (situation as per 1 January 2006).

5.1. Requirements

Because the domain is restricted to the Base Insurance, it is easy to formulate the requirements:

- The model should be able to calculate the benefits for submitted claims according to the coverage rules of the Base Insurance.
- The model should show the results to the insured members (*benefit specification).*

The Dutch government determines the coverage for the Base Insurance. The functional requirement can be derived from the Zorgverzekeringswet (Law for Health Insurance)
See "Appendix 2: The Dutch Base Insurance" for a list of coverages of the Base Insurance.
Non-functional requirements are out-of-scope.

5.1.1. Actors

The following actors play a role in the model:

- Members (Persons that have a Policy) can submit claims. The model processes the claims and calculates the covered amount according to the rules of the Base Insurance. The calculation results are presented to the members in the form of a Benefit Specification.
- Relation Management is responsible for (de)registering Persons and creating Policies.
- Functional Management configures the model by setting up coverages rules.

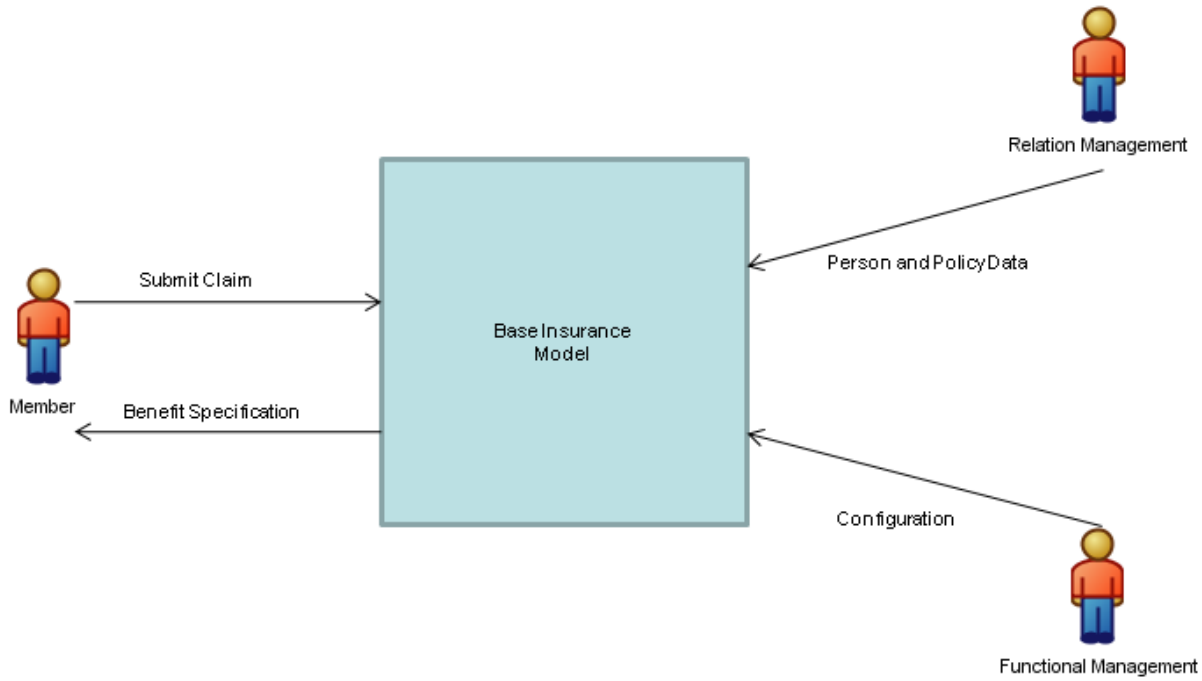


Figure 9 Actors

The following objects are involved in the processing of a claim:

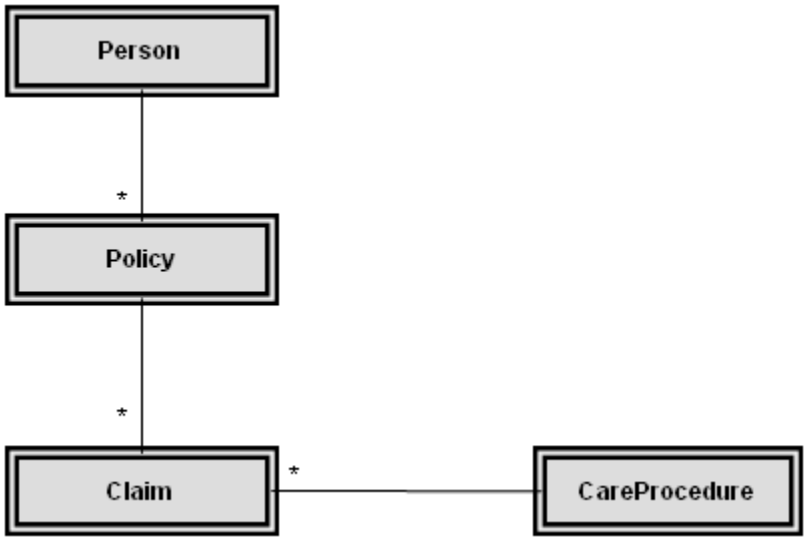


Figure 10 High Level Object Model

- A *person* can have zero or more policies.
- *Claims* are submitted within the context of a policy.
- A claim concerns the declaration of the cost of a medical treatment. The medical treatment is defined as a *Care Procedure.*

Claims processing is triggered by submitting of a claim. The ultimate result is to calculate the covered amount (benefit) and possible other results. Those results are in the claim object. The member receives a benefit specification with the results of the calculation.

5.1.2. Use Cases

From the total list of coverages defined in "Appendix 2: The Dutch Base Insurance", nine use cases are derived. See table below. Each use case concerns a different aspect. A model that implements all these use cases supports the Base Insurance almost completely.

Table 2 Base Insurance Use Cases

Use Case	Description	Aspect
1	Not Covered (Alternative Medicine)	Not Covered.
2	Covered 100% (General Practitioner Care)	Completely covered.
3	Covered 100% with Age Limit (Dental Care)	Completely covered depending on age of member.
4	Covered 100% up to maximum number (Nutritional Counselling)	Completely covered up to maximum number of units.
5	Covered with Co-payment (Inpatient Delivery)	Deduction of co-payment per unit.
6	Coverage of Treatment (Physiotherapy)	Completely covered, starting from a treatment.
7	Cover to Maximum Number of Units with Co-payment (Maternity Care)	Co-payment per unit and maximum number of units.
8	Cover specific treatments (IVF)	Completely covered for specific treatments.
9	Cover Partly (Prostheses)	Covered percentage.

The aspects can be divided in two groups:

1. Aspects that determine whether a claim is covered or not. Coverage can be unconditional, or subject to conditions like age and earlier treatments consumed.
2. Aspects that determine how much is covered: fully, partly, or with co-payment.

5.2. Assumptions

This chapter describes the assumptions used for the design and the construction of the protocol model.

5.2.1. Goal of the model

A model represents only a part of reality. Which part depends on the goal of the model. The goal of the model described in this report is to model the rules of the Dutch Healthcare Insurance 'Base Insurance'. The rules of the Base Insurance are described in "Appendix 2: The Dutch Base Insurance". This part of the model is elaborated in detail. Other aspects of healthcare insurance are modelled in less detail. This concerns for example the maintenance of data dealing with policies.

5.2.2. Abstraction and Parameters

Product software will be used by different customers. An important aspect of product software is to which extent and how the application supports *differences between clients*, because it is not likely that processes at different customers will execute exactly the same. The product has therefore to be adapted to a situation at a specific customer.
The product also needs adaptation for changes caused by the *passing of time*. Take for example the yearly changes in the *law* for the coverage of the Base Insurance.

So an important requirement is flexibility: the possibility to adapt the product to different and changing circumstances.
The required flexibility is taken into account during the design of the protocol model by applying *abstraction* and *parameterization*.

Abstraction: domain concepts are not directly translated into the model, but are generalized first. Take for example General Practitioner Care and Dental care, for which the costs are covered by the Base Insurance. The model uses the abstraction Coverage to model both General Practitioner Care and Dental Care. Abstraction lowers the chance that changes in the domain lead to structural changes in the model: "The abstractions that emerge during design are key to making a design flexible" (Gamma et al, 1995).

Parameters: boundary values and constants are not hard-coded in the model, but can be set using parameters. This concerns age boundaries and coverage percentages for examples.

5.2.3. Reusability

The use of abstraction and parameters as described in the previous section, leads to better reuse: constructions of the model can be used to model several insurance products using the same generalizations.

5.2.4. Time validity support

Many objects in the model are *time valid*: their data only applies during a certain period of time. Most commonly, time validity is modelled by adding start- and end date to objects. In the constructed model this time validity is in general not implemented to simplify the model.

5.2.5. Unique keys

Object instances are identified using one or more attributes. For example, a policy has a policy number. It's not possible to have multiple policies with the same policy number. The enforcement of uniqueness of identifying attributes is not implemented in the model. ModelScope does not have easy support to model unique keys.

5.2.6. Authorizations

For certain medical procedures, the member needs to get an approval from the insurance company before claiming the costs. Such an approval is called an *authorization*. Due to the limited amount of time available, authorizations are not included in the model.

5.3. Analysis of Benefit Rules

This chapter presents an analysis of the benefit rules of the Base Insurance. The analysis is based on the overview presented in "Appendix 2: The Dutch Base Insurance".

5.3.1. Products and Coverages

In addition to the Base Insurance, an Insurance Provider can offer additional supplementary insurances. Both Base Insurance and supplementary insurances are examples of insurance *products*.
A Product supplies benefits for certain medical treatments. For the Base Insurance, it is defined by law which treatments are covered. Examples are General Practitioner Care, Speech-training and Haemodialysis.
For supplementary insurances, an insurance provider can freely determine which costs are covered by the supplementary product.
So a Product p contains a set of *coverages Cp*:
$Cp=\{c_1, c_2, c_3...,c_n\}$

A Coverage consists of a set of medical treatments. For each medical treatment, a Care Procedure is defined. The coverage of General Practitioner Care contains the procedures (among others):

- 01/12000: Short consult.
- 01/12001: Long consult.
- 01/12002: Consult at home.

So a Coverage C can be defines as a set of Care Procedures CPc:
$C=(CPc)$ *where* $CPc=\{cp_1, cp_2, cp_3...,cp_n\}$

Product p covers the cost of care procedure cp_x if p contains a coverage c where pr_x is in the set of care procedures of c:
$Covered(p, cp_x) := \exists c \in Cp\ (\exists cp \in cpc\ (cp= cp_x))$

The sets of covered care procedures of the different coverages of a product are disjunct. A procedure cp_x is covered by exactly one coverage of a product p, or the care procedure is not covered at all.
$\forall c_1 \in Cp\ (Covered(c_1, cp_x) \rightarrow (\forall c_2 \in Cp\ (!Covered(c_2, cp_x) \vee c_2=c_1))$

5.3.2. Conditions

So a coverage covers the cost of a set of care procedures. Care procedure cp is covered by coverage c if cp is part of the set of covered procedures of c. This is an example of a *condition.*
A condition must be satisfied to get benefits for the cost of a care procedure.
For some coverage's, additional conditions apply. This applies for example to 'Occupational Therapy'. This coverage only covers a maximum of ten hours of treatment per year.
The Base Insurance has different types of conditions:

- A condition on the *Number of Units*. For example for Occupational Therapy.
- A condition on the *Treatment Number*. For example for IVF: only the second and third treatment is covered.

- A condition on the Age of the member. This applies Dental Care is only covered up to on age of 18.

It might look like more types of conditions exist in the Base Insurance. For example, for care is only covered with a chronic indication. In practice however, a different care procedure is defined for the same treatment. So a care procedure is defined for chronic and one for non-chronic indication. So a separate condition type is superfluous.
The definition of coverage C from previous section can be extended to:
C=(CPc, CN) where CN = {cn_1, cn_2, ... cn_n}, n>=1 is a set of conditions to be satisfied by cp_x for cp_x being covered. CN at least contains the condition that cp_x should be an element of CPc.

5.3.3. Benefit Calculation

When all conditions are satisfied, the benefit amount can be calculated. Different possibilities exist for the benefit calculation (bc):

- Cost is covered *fully*.
- A *percentage* of the cost is covered. This applies for example to prostheses, which are covered for 75 percent.
- The cost is covered after deduction of a *co-payment*. This applies for example to Maternity Care.

So the definition of coverage C can be extended to:
C=(CPc, CN, bc) where *bc ∈ BC and BC = {full, percentage, co-payment}*

5.3.4. Mathematical Model of Benefit Rules

To summarize the three preceding sections, a product p can be defined as a list of coverages:
Cp={c_1, c_2, c_3...,c_n} where c_n is a coverage.
Coverage C is defined as: *C=(CPc, CN, bc)* where

1. *CPc ={cp_1, cp_2, cp_3...,cp_n}* is a list of covered care procedures.
2. *CN =* {cn_1, cn_2, ... cn_n}, n>=1 is a set of conditions to be satisfied.
3. *bc ∈ BC and BC = {full, percentage, co-payment}*

5.3.5. Policies and claims

The main goal of the model is the modelling of coverage rules. Policies and claims are less detailed in the model.

A *policy* enables a member to claim coverage of costs covered by the product. A policy has a fixed duration of one year. Only medical costs of that year are covered.

Some simplifications have been made:

- In the model, a policy has only one member. In reality, the members of a whole family are enrolled on the policy.

Members can submit *claims* to get healthcare costs paid. A *claim* concerns the declaration of healthcare related costs in order to get compensation of these costs according to the rules of the policy.

Again: the model is simplified: in reality a claim has one or more claim lines, so several costs can be claimed together. The model does not have claim lines.

5.4. Model

Based on the goals and analysis described in the previous two chapters, the protocol model is constructed. This chapter describes the model.

5.4.1. Structure

The figure below depicts the structure of the most important parts of the model. It gives an overview of the relations between the objects. Some details are omitted for simplicity. These details are described in later sections.

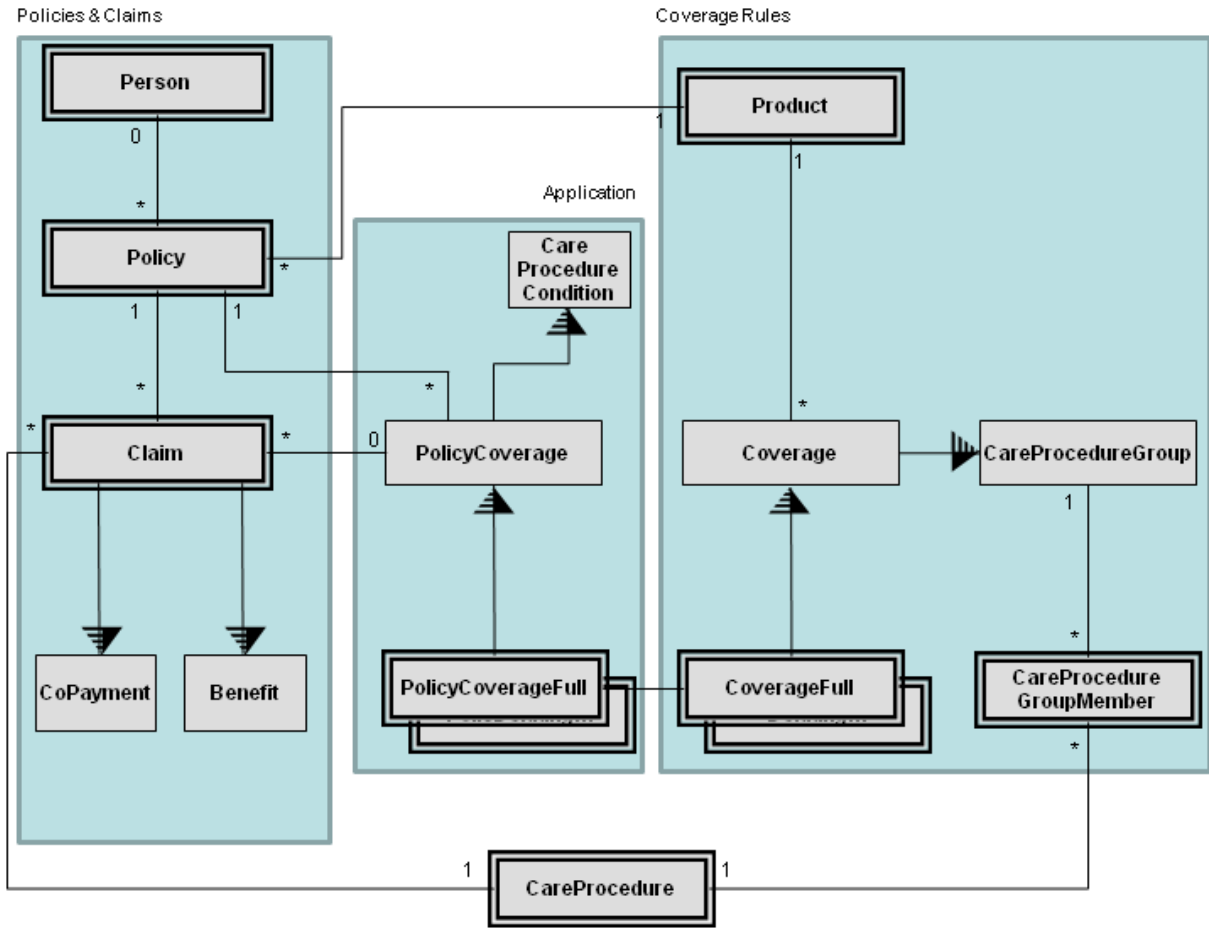


Figure 11 Main structure of healthcare insurance model

The next three sections each describe a part of the model presented in the figure above:

- Coverage rules contain the behaviours Product, Coverage, Care Procedure, Care Procedure Member, and Coverage.
- Policies and Claims are targeted towards the insured member. This part contains behaviours Person, Policy and Claim.
- The coverage rules are applied to submitted claims in the part that contains the behaviours Care Procedure Condition, Policy Coverage and Policy Coverage Full.

Multiple instances are shown for CoverageFull and PolicyCoverageFull because multiple different object types exists that include Coverage and PolicyCoverage. Details follow in next sections.

5.4.2. Modelling of coverage rules

This section describes the modelling of coverage rules. This part of the model is used by actor Functional Management.
The behaviours presented in this section mainly have data, they hardly have behaviour. Most objects only have the Active state[3]. All objects have a Create and a Change event to create new and change existing objects respectively.

The behaviours in this section only model the definition of coverage rules. The usage of the definition in the context of a policy is described in 5.4.4 "Modelling of the application of coverage rules".
The table below gives a short description of the objects and behaviours used in this part of the model. More details can be found in 15 "Appendix 6: Model Reference".

Table 3 Behaviours used to model Coverage Rules

Behaviour	Description	Includes
AgeLimit	This behaviour must be included by all coverages where the age of the member determines the coverage.	
BenefitCoPayment	This behaviour must be included by all coverages where a co-payment should be deducted.	
BenefitPercentage	This behaviour must be included by all coverages that only cover a percentage of the cost.	
CareProcedure	A care procedure is a definition of a medical treatment.	
CareProcedureGroup	A CareProcedureGroup models a set of care procedures.	
CareProcedureGroupMember	This object relates a care procedure to a care procedure group.	
Coverage	Coverage is a set of care procedures that are covered using the same rules.	CareProcedureGroup
CoverageAge	This object models a coverage with has an age condition: the claim is only covered when the age of the member at the claim date is within the specified limits.	Coverage AgeLimit
CoverageCoPayment	This object models a coverage for which a co-payment is deducted.	Coverage BenefitCoPayment
CoverageFull	This object models a coverage that covers the full price of the care procedure.	Coverage

[3] The state Inactive could be added to certain objects. An inactive object still exists, but cannot be used anymore. A product could be declared Inactive when no new policies should be created for that product for example. Making objects inactive is not part of the design of the model.

CoverageMaximumNumber	This object models a coverage that covers up to a maximum number of units.	Coverage MaximumNumberLimit
CoverageMaximumNumber-CoPayment	This object models a coverage which both a co-payment amount and a maximum	Coverage MaximumNumberLimit BenefitCoPayment
CoverageTreatment	This object models a coverage for certain treatments only.	Coverage TreatmentLimit
MaximumNumberLimit	This behaviour must be included by all coverages that cover up to a maximum number of units.	
Product	A *product* is a set of coverages of healthcare costs.	
TreatmentLimit	This behaviour must be included by all behaviours that only cover certain treatments.	

CoverageMaximumNumberCoPayment illustrates the mixin/multiple-inheritance capabilities of protocol modelling. The combination of maximum covered number of units and co-payment is implemented by including both the behaviours MaximumNumberLimit and BenefitCoPayment.

5.4.3. Modelling of policies and claims

This part of the model contains some simplifications compared to reality. For example, a policy has only one member, while in reality a whole family can be enrolled.

5.4.3.1. Object Person

A *person* is a human being known by the insurance company. A person may have (had) a policy. The diagram below shows the state transitions of Person:

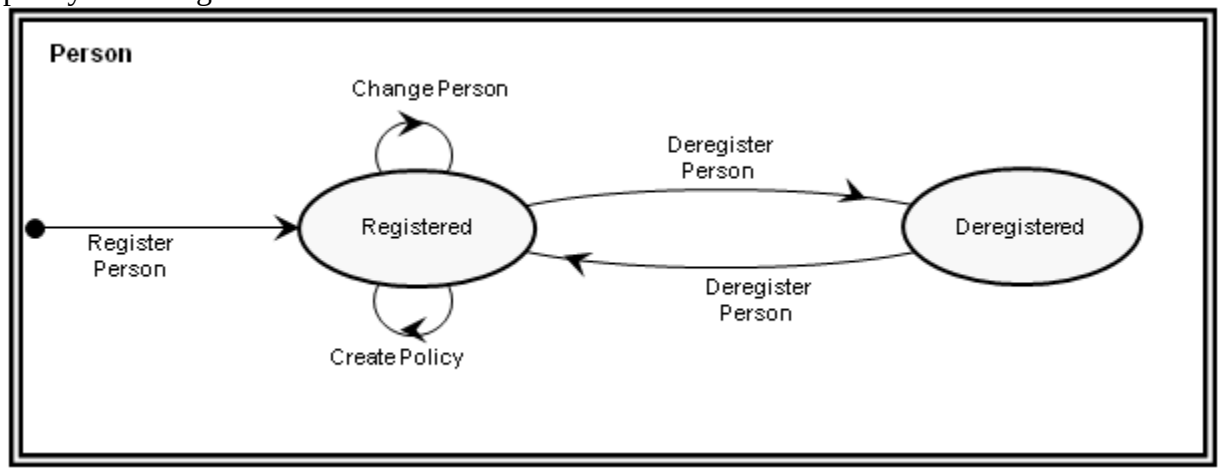
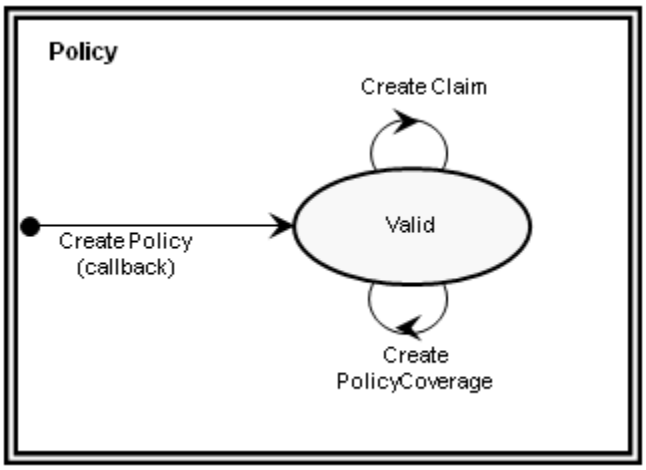


Figure 12 Person Protocol Machine

5.4.3.2. Object Policy

A *policy* grants a person the right to claim healthcare cost covered by the policy product. See the picture below for the Policy protocol machine:



5.4.3.3. Object Claim

A member submits a *claim* to the insurance company in order to receive benefits for healthcare costs according to the coverage rules of the policy product.
The diagram below shows the state transitions for Claim.

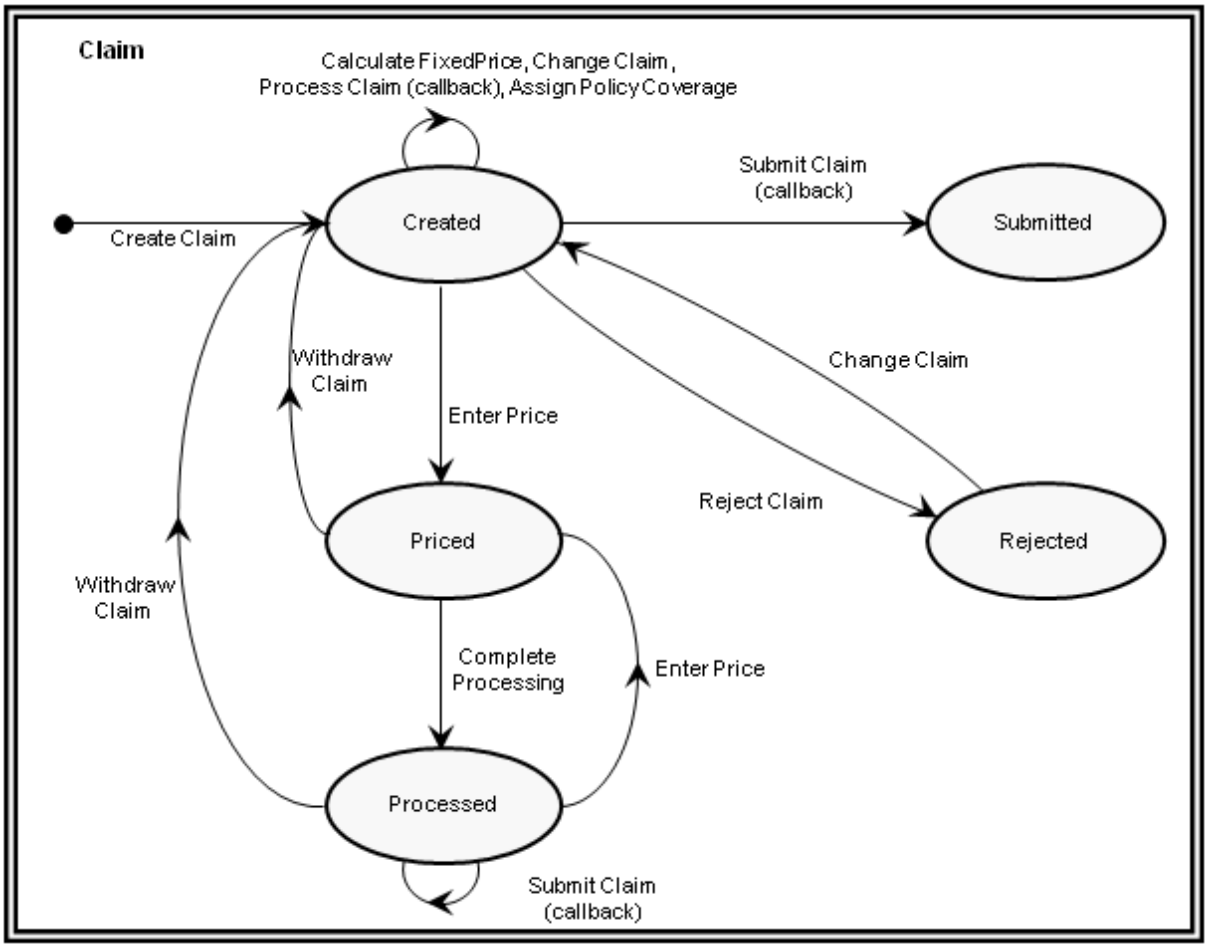


Figure 13 Claim Protocol Machine

State Submitted looks like a dead-end. This is however not the case. Event Submit Claim is handled by a callback and translated in either a Reject Claim or a Process Claim event. Process Claim in its turn generates an Enter Price event.

Claim includes behaviours Benefit and CoPayment. They store the results of the claims processing. Their state machines are shown in diagrams below:

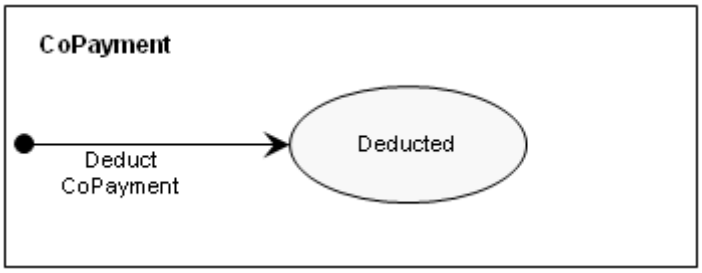


Figure 14 CoPayment Protocol Machine

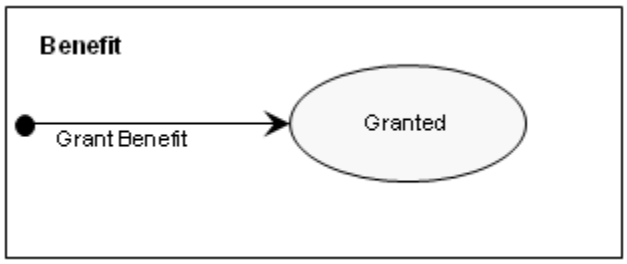


Figure 15 Benefit Protocol Machine

Both machines have an amount: CoPayment Amount and Benefit Amount respectively. After processing a claim, the Benefit and CoPayment machine give information about the coverage of the claim:

- When CoPayment is in the Deducted state, co-payment is applied for the claim. The amount is in attribute CoPayment Amount.
- When Benefit is in the Granted state, benefits were granted to the claim. The amount is in attribute Benefit Amount.

An alternative way of modelling was possible: just use the state of the claim. Because CoPayment and Benefit are independent factors, the state was normalized into the two included behaviours.

Claim events are generated by actor Member. Most events are however generated sub-events.

5.4.4. Modelling of the application of coverage rules

This section describes the part of the model that applies the coverage rules to submitted claims. Besides the structure of Policy Coverages, the internal sub-events are described.

5.4.4.1. PolicyCoverages

The coverage of a claim can depend on earlier submitted claims. This is for example the case if a coverage has a maximum defined: when the maximum has been reached, new submitted claims are not covered anymore and are rejected by the coverage. So it is important to capture the state of a coverage *in the context of the policy*. This context is stored in PolicyCoverage behaviours. Each policy contains a set of policy coverages. For each Coverage object described in section 5.4.2 "Modelling of coverage rules" a related PolicyCoverage object exists:

- PolicyCoverageFull is related to CoverageFull
- PolicyCoveragePercentage is related to CoveragePercentage
- PolicyCoverageAge is related to CoverageAge
- Etc.

When a new policy is created, PolicyCoverages are created for all Coverages of the product, using the Create Policy callback. See code-snippet below:

```
Callback for Create Policy

Instance[] coverages = product.selectByRef("Coverage", "Product");

for (Instance coverage: coverages) {
        //Generate a create policy coverage event
        String objectType = coverage.getObjectType();
        Event event = this.createEvent("Create PolicyCoverage");
        event.setNewInstance("PolicyCoverage","Policy" + objectType);
        event.setInstance("Coverage", coverage);
        event.setInstance("Policy", policy);
        event.submitToModel();
}
```

For all defined coverages, a Create event is submitted.
So, after creating a policy for a product with three coverages, three policy coverages are created of matching type. See the next picture[4]:
Product Base Insurance is associated with coverages of different type for General Practitioner Care, Dental Care and Nutritional Counselling. Policy 123-456-789 related with this product, has three policy coverages, one for each coverage.

[4] Not all available coverage types are shown in this diagram. See table 5 for the full list.

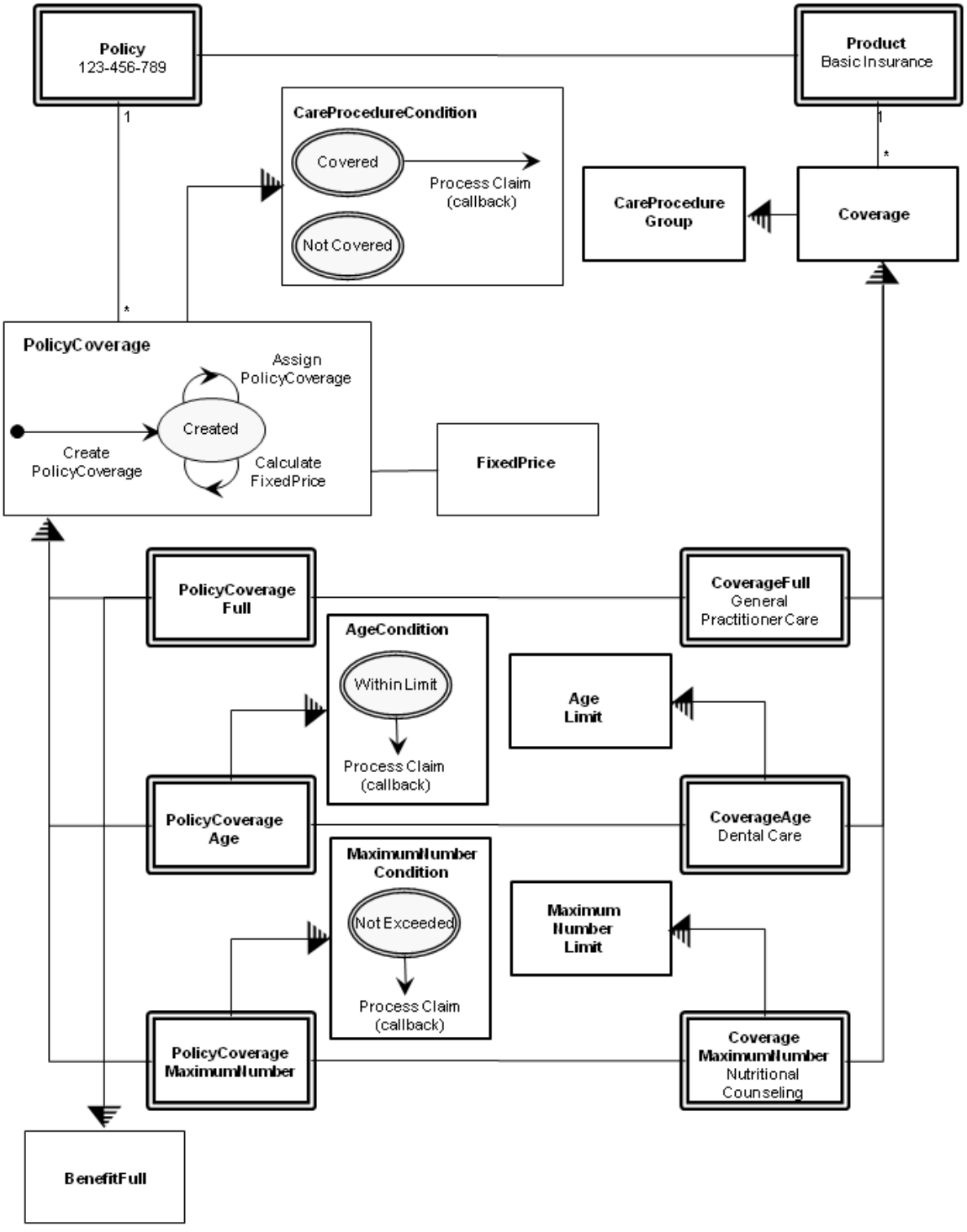


Figure 16 Policy and PolicyCoverages

During processing, the PolicyCoverage gets necessary information from its related Coverage object. For example PolicyCoverageAge retrieves the age limits from the AgeLimit behaviour included in the CoverageAge object.

The table below gives a short description of the objects and behaviours used in this part of the model. More details can be found in 15 “Appendix 6: Model Reference”.

Table 4 Behaviours used to model Coverage Rules

Behaviour	Description	Includes
AgeCondition	Only accepts event ProcessClaim if the age of the member at the claim date is within the defined age limits.	
BenefitCoPayment	Similar to BenefitFull, after copayment deduction.	
BenefitFull	Calculates benefits amount = number * price.	
BenefitPercentage	Calculates benefits amount = number * price * percentage.	
CareProcedureCondition	Only accepts event ProcessClaim if the care procedure is in the care procedure group of the coverage.	
FixedPrice	Price calculator that sets the price of the claim to the price of the procedure.	
MaximumNumberCondition	Only accepts event ProcessClaim if the total claimed number is below the defined maximum number of treatments.	
PolicyCoverage	Behaviour to be included by all PolicyCoverages.	FixedPrice CareProcedureCondition
PolicyCoverageAge	PolicyCoverage for fully covered care procedures for members of certain age only.	PolicyCoverage AgeCondition
PolicyCoverageCoPayment	PolicyCoverage for care procedures with copayment.	PolicyCoverage BenefitCoPayment
PolicyCoverageFull	PolicyCoverage for unconditional fully covered care procedures.	PolicyCoverage
PolicyCoverageMaximum-Number	PolicyCoverage for fully covered care procedures up to a defined maximum number of treatments.	PolicyCoverage MaximumNumber-Condition
PolicyCoverageMaximum-NumberCoPayment	Combination of PolicyCoverageMaximumNumber and PolicyCoverageCoPayment.	PolicyCoverage MaximumNumber-Condition BenefitCoPayment
PolicyCoveragePercentage	PolicyCoverage for partially covered care procedures.	PolicyCoverage BenefitPercentage
PolicyCoverageTreatment	PolicyCoverage for fully covered care procedures for certain treatments only.	PolicyCoverage TreatmentCondition
TreatmentCondition	Only accepts event ProcessClaim if the treatment sequence is within the defined limits.	

5.4.4.2. Delegate a claim to a PolicyCoverage

A member does not act upon policy coverages, but a member submits a claim to a policy. So a policy acts as a façade for the policy coverages.

A claim is covered if the policy contains a policy coverage that covers the procedure. In the model, this is implemented this way: "*A policy coverage covers a claim if the policy coverage accepts event Process Claim (i.e. if Process Claim is in context)*".

So the policy can forward the Submit Claim event to the (one and only) policy coverage for which Process Claim is in context. See the following code snippet:

```
Callback Submit Claim
//find all coverages that can process the event: should be exactly one.
Instance[] coverages = policy.selectInContext("PolicyCoverage", "Process
Claim");

List<Instance> policyCoverageList = new ArrayList<Instance>();
for (Instance coverage: coverages) {
    Instance coveragePolicy = coverage.getInstance("Policy");
    if (coveragePolicy.equals(policy)) {
        policyCoverageList.add(coverage);
    }
}
if (policyCoverageList.size() != 1) {
    log (policyCoverageList);
    Event event = this.createEvent("Reject Claim");
    event.setInstance("Claim", claim);
    event.setString("Processing Info", "Rejected: "+  coverages.length + "
coverages found");
    event.log();
    event.submitToModel();
}
else {
    //Create processClaim event
    Event event = this.createEvent("Process Claim");
    event.setInstance("PolicyCoverage", policyCoverageList.get(0));
    event.setInstance("Claim", claim);
    event.log();
    event.submitToCallback();
}
```

The acceptance or rejection of a claim is totally delegated from the policy to the policy coverages. Three situations are possible:

- No PolicyCoverage accepts the ProcessClaim event. This means the care procedure of the claim is not covered by the product.
- Multiple PolicyCoverages accept the ProcessClaim event. This is a setup error: The sets of care procedure group members for all coverages of the product are not disjunct. The care procedure of the claim is defined as a member of multiple coverages.
- Exactly one PolicyCoverage accepts the ProcessClaim event. The care procedure of the claim is covered by this coverage. The ProcessClaim event is sent to the PolicyCoverage.

5.4.4.3. Conditions

A policy coverage should only accept the ProcessClaim event if all conditions are met. Conditions are evaluated by Condition behaviours that are included in PolicyCoverages. Examples are:

- CareProcedureCondition only accepts the ProcessClaim event if the care procedure of the claim is in the list of covered procedures of the coverage.
- AgeCondition only accepts event ProcessClaim if the age of the member at the claim date is within the defined age limits.
- MaximumNumberCondition only accepts event ProcessClaim if the total claimed number is below the defined maximum number of treatments.

Adding conditions to a PolicyCoverage leverages composition of protocol modelling behaviours.

5.4.4.4. Price- and Benefit calculation

The moment a PolicyCoverage accepts a Process Claim event, all conditions are satisfied. The claim processing can start. Claim processing consists of the execution of a number of steps depicted in diagram below:

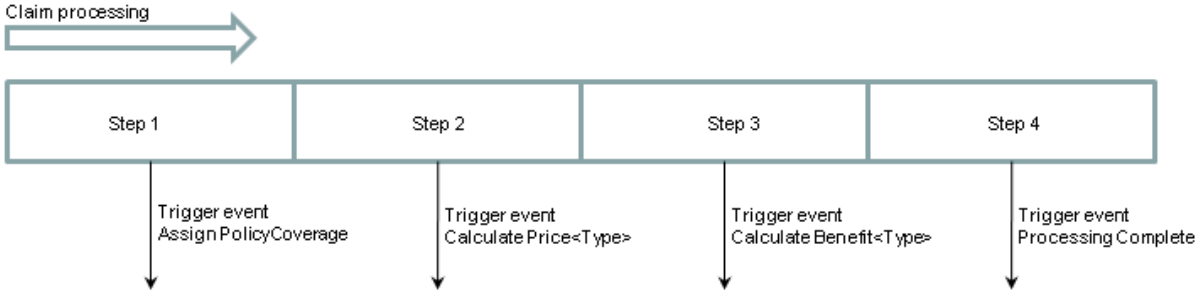


Figure 17 Claim Processing Steps

These steps are implemented in the Process Claim callback. This callback issues four events:

1. The Assign PolicyCoverage event associates the claim with the policy coverage that processes it. This facilitates calculation of total consumption on a policy coverage.
2. The Calculate Price<type> event populates the Price attribute in the claim. Price is the input of the benefit calculation.
3. The Calculate Benefit<type> event calculates the benefits.
4. The Processing Complete event sets the state of the claim to Processed.

Note that the implementation of step 3 and 4 is flexible, depending on the policy coverage type, a different event is triggered:

- The type of price calculation is determined by getting the value of the attribute PriceCalculatorName. This attribute is implemented in behaviour FixedPrice and has the value FixedPrice. So the flexibility for different price calculators is in place, but not used in the model. It's not required to support the Base Insurance.
- The same applies for the benefit calculation. The type is determined by getting the value of the BenefitCalculatorName. This attribute is implemented in the different Benefit objects and can have the values BenefitFull, BenefitPercentage or BenefitCoPayment.

With this convention, it is easy to add new Benefit and Price calculators as long as:

- A derived attribute BenefitName or PriceCalculatorName is defined.
- An event Calculate<BenefitName> or Calculate< PriceCalculatorName> exists.

5.5. Flexibility

This section describes how flexibility is designed into the model.

5.5.1. Configurable Coverages

All coverages can be setup using the Functional Management actor. All definitions and limits are configurable by setting values to parameters. This applies to:

- List of care procedures that are part covered by the coverage

- Configuration parameters of limits like Age From and Age To.

So setting up a coverage is just a matter of defining the coverage, as long as one of the defined coverage types is used. See table below for the defined coverage types and their purpose:

Table 5 Coverage Types

Type	Purpose
CoverageFull	Covers 100% of the price
CoveragePercentage	Covers a percentage of the price
CoverageAge	Covers 100% if age within limits
CoverageCoPayment	Covers after deducting a co-payment per unit
CoverageTreatment	Covers 100% for certain treatment number
CoverageMaximumNumber	Covers up to a maximum number of units
CoverageMaximumNumberCoPayment	Covers up to a maximum number of units and also deducts a co-payment per unit.

5.5.2. Adding Coverage types

In cases where the predefined Coverage Types are not sufficient, one can define new coverages by extending the model:

- Create a new Coverage<X> object that includes the Coverage behaviour.
- Include other behaviours that hold required parameters for Coverage<X>
- Create a PolicyCoverage<X> object.
- Add Condition behaviours as needed to PolicyCoverage<X>

Because each condition is modelled in its own behaviour, they can be reused when creating new coverages. In fact, this has been done for CoverageMaximumNumberCoPayment: it's just a combination of CoverageCoPayment and CoverageMaximumNumber.

5.5.3. Steps in the Flow

A claim is processed by executing several steps in sequential order. This process is implemented in the Process Claim callback. By changing this callback, new steps can be added.

5.5.4. Pluggable Price- and Benefit Calculator

Both for price- and benefit calculation, the Strategy pattern is used. So per coverage, the implementation of price- and benefit calculation can vary.
Steps required for adding new calculators:

- Create a new Benefit<X> or Price<X> behaviour
- Include the Benefit<X> or Price<X> in the PolicyCoverage that should use it.
- Implement the calculator in the callback of event Calculate Benefit<X> or Calculate Price <X>

5.6. Support of use cases

The developed model supports all use cases defined in 5.1.2 "Use Cases". "Appendix 7: Use cases" describes how the use cases can be executed.

6. Results and analysis

In the previous chapter, a flexible model is described for the Base Insurance coverages as of 1 January 2006. This chapter describes the impact on the model of changes in the Base Insurance since 1 January 2007 until 2011. As such, it captures the results of the third phase of the research project.
The first section gives an overview and classification of the type of changes for each year. The second section describes the impact of the changes on the model.

6.1. Overview and Classification of Changes

A detailed overview of all changes in the Base Insurance can be found in "Appendix 3: Changes in the coverage of the Base Insurance".
This section summarizes and classifies the changes. The classification scheme used conforms to the product and coverage definitions defined in 5.3.4 "Mathematical Model of Benefit Rules".

Changes are classified as one of:

1) Change in covered care procedures. Two subclasses are defined:
   a) Coverage Added. A care procedure that was uncovered previously, has become a covered procedure.
   b) Coverage Removed: A care procedure that was covered previously, has become uncovered.
2) Condition Change: A care procedure that was covered before is still covered but the conditions for coverage have become more restrictive or less restrictive.
3) Change in benefit calculation. The algorithm to calculate the benefit amount of a claim has changed.
4) Other Change: Changes that do not belong to one of the three categories above.

The table below classifies all changes since 1 January 2007 using the four change types:

Table 6 Overview of Changes

Year	Change in Covered Care Procedures		Condition Changed	Change in Benefit Calculation	Other change
	Coverage Added	**Coverage Removed**			
2007	Prenatal screening for congenital defects		First IVF treatment also covered		
	Abdominoplasty				Personal Budget for visual aids
2008	Mental Healthcare		Age limit for Birth Control removed	Mandatory Yearly Deductible	
			Dental Care age limit set to 21		
			Maximum number of hours maternity care increased with five.		
2009	Diagnosis and treatment of severe dyslexia	Lift Chair		Increase of Mandatory Yearly Deductible	
		Hypnotics and tranquillizers			
2010	Mandibular advancement devices	Mucolytic Agent Acetylcysteïne	Organ transplantation outside EU	Increase of Mandatory Yearly Deductible	
2011		Durable Medical Equipment	Birth Control age limit set to 21		
		Simple extractions by oral surgeons	Dental Care age limit set to 18.		

6.2. Impact of Changes on Model

The impact of changes is divided in two categories:

1. Changes that impact the *configuration* of the model. The change can be implemented in the model by changing the model configuration: adding/deleting or modifying model data only.
2. Changes that impact the *structure* of the model: the object and behaviour definitions of the model are not sufficient to implement the change. New objects and behaviours are needed.

Preferably, each change type only impacts the configuration of the model. The model represents a working software product so changes that impact the configuration only do not lead to software changes in the product.

The sections below each describe the impact of one of the change types defined in section 6.1 ”Overview and Classification of Changes”.

6.2.1. Impact of Change Type “Coverage Added”

In general, changes of this type only impact the configuration of the model. New covered procedures can be implemented in the model by defining additional Coverage and CareProcedureGroupMember instances.

6.2.2. Impact of Change Type “Coverage Removed”

The same can be said about the impact of change type “Coverage Removed’. Changes of this type can be implemented by removing/deactivating CareProcedureGroupMember instances.

6.2.3. Impact of Change Type “Condition Changed”

Changes of this type in general can be implemented by changing the configuration of the model. Take for example the change of 2011 “Dental Care age limit set to 18”. This is easily implemented by setting the “Age To” attribute of the AgeLimit behaviour to 18.

6.2.4. Impact of Change Type “Change in Benefit Calculation”

In 2006, a mandatory deductible did not exist for the Base Insurance. So the model described in chapter 5 “Design Protocol Model for Base Insurance 2006” does not contain behaviours to handle a mandatory deductible. So the introduction of the mandatory deductible in 2008 impacts the structure of the model.

6.2.5. Impact of Change Type “Other Change”

In 2007, a Personal Budget for visual aids for visually disabled people is created. When a Personal Budget applies, the financial flow becomes different: the budget amount is paid upfront, before the costs are made. When a Personal Budget is involved, no costs are claimed. There is no need to process a claim anymore. So this change does not impact the model.

6.3. Summary of Results

This section summarizes the results. All changes occurred in the Base Insurance since 2006, are classified in four different types. Each change can be implemented by either:

- Changing the model configuration.
- Changing the model structure.

The table below shows the relation between the four change types and the two possible model impacts:

Table 7 Change Types and Model Impact

Change Type	Impact on Model Configuration	Impact on Model Structure
Change in Covered Care Procedures	Yes	No
Condition Changed	Yes	No
Change in Benefit Calculation	Yes	Yes
Other Change	n.a.	n.a.

Only the changes of type “Change in Benefit Calculation” are impacting the model structure. The other change types can be implemented by changing the model configuration.
The introduction of a mandatory deductible in 2008 cannot be handled by the model developed in chapter 5, because this concept did not exist in 2006. The concept of a deductible must be added to the model developed in chapter 5 for two reasons:

- A deductible is a common used concept in insurances. The extensions to the model are useful not only for healthcare insurances but also for other type of insurances.
- It gives the opportunity to gain experience with model evolution: what's the impact of the structure change on other parts of the model. In other words, is it possible to enhance the model by adding a new concept without a major rewrite?

The next chapter analysis the concept of a mandatory deductible as it exists in the Base Insurance and describes how the model is extended.

7. Model Enhancements

In the previous chapter it is concluded that the model structure has to be changed to support a mandatory deductible. This chapter describes how that can be done.

7.1. Analysis of Mandatory Deductible

This section analysis the requirements of the mandatory deductible as introduced in the Base Insurance in 2008. See section "Mandatory Yearly Deductible" in "Appendix 3: Changes in the coverage of the Base Insurance" for more details.
The mandatory deductible:

- Has a configurable amount, starting with 150 euro in 2008.
- Does not apply to members below the age of 18.
- Does not apply to all coverages. General Practitioner Care is excluded for example.

The cost of care procedures that are subject to the yearly mandatory deductible have to be paid by the member up to the amount of the deductible. When the deductible is fully consumed, costs are reimbursed by the insurance company in the normal way.

Chronically ill and disabled people are financially compensated. This compensation does not impact claims processing and thus needs not to be implemented in the model. The compensation is afterwards at the end of the year.

7.1.1. Combination with Co-payment and partly Coverage

Some words need to be said about the combination of co-payment, partly coverage and a mandatory deductible because the sequence in which they are applied impacts the results. In the Base Insurance, the sequence is defined as follows:

1. Co-payment is deducted first.
2. After that, the coverage is calculated.
3. The calculated coverage amount is subject to the mandatory deductible.

Take this artificial example:

- Care procedure CP1 costs 400 euro and a co-payment of 80 euro is defined.
- The Base Insurance covers 75% of CP1.
- Member M1 has not consumed anything of the mandatory deductible of 150 euro.

M1 submits a claim for CP1. The processing of this claim is as follows:

- A co-payment of 80 euro is deducted.
- From the remaining 320 euro, 75% = 240 euro is covered.
- The yearly deductible is subtracted. The remaining 240-150 = 90 will be paid to M1.
- M1 now has completely consumed his yearly deductible. For a next claim of CP1, the insurance company will pay 240 euro.

7.2. Implementation of Mandatory Deductible

The introduction to the mandatory deductible is classified as a change in the benefit calculation.
Remember that a Coverage C is defined as: $C=(CPc, CN, bc)$ where $bc \in BC$ and $BC = \{full, percentage, co\text{-}payment\}$. The model described in chapter 5 supports adding new benefit calculation strategies, see "Pluggable Price- and Benefit Calculator" so it is possible to implement new benefit calculations that support a deductible.
It turns out however that a mandatory deductible is an independent concept compared to benefit calculation. All possibilities of benefit calculation (full, percentage and co-payment) can be combined with and without a mandatory deductible. As benefit calculation is independent of mandatory deductible, it is better to introduce a whole new concept to the

model. The introduction of the mandatory deductible in 2008 did not change the results of the benefit calculation, but did change the amount being paid to the member:

- In 2007: the covered amount was paid to the member.
- In 2008: the covered amount was paid to the member after subtraction of the mandatory deductible (if not consumed yet).

7.3. Extended Product Definition

To support mandatory deductibles, the mathematical model of 5.3.4 "Mathematical Model of Benefit Rules" is extended to (extensions in bold):
*Cp=({c_1, c_2, c_3...,c_n}**, d, al**)* where c_n is a coverage. Parameter deductible *d*, $d \geq 0$ defines the deductible amount. Parameter *al* defines the age limits of members for which the deductible applies.
Coverage C is defined as: *C=(CPc, CN, bc,**dc**)* where

1. *CPc ={cp_1, cp_2, cp_3...,cp_n}* is a list of covered care procedures.
2. *CN =* {cn_1, cn_2, ... cn_n}, n>=1 is a set of conditions to be satisfied.
3. *bc∈ BC and BC = {full, percentage, co-payment}*
4. *dc* ∈ {true, false} indicates whether the mandatory deductible applies to the coverage.

This is implemented in the model by including the new behaviour Deductible in the Product object. See diagram below:

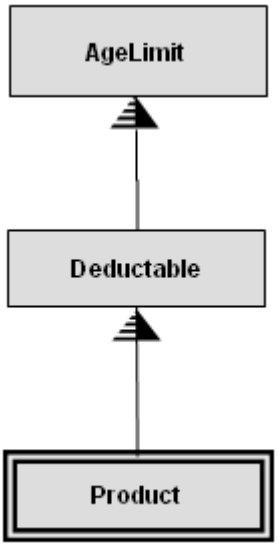


Figure 18 Product Definition Extension

The Deductible behaviour has the attribute Deductible Amount, holding the deductible amount for the product for members matching the age limits.

Attribute "Indicator Mandatory Deductible" is added to the behaviour Coverage to indicate for which coverages the mandatory deductible applies.

7.3.1. Extended Claim processing

During claims processing, the mandatory deductible should be calculated and stored in the claim object. A separate behaviour Deductible Consumption is created for that purpose. The claims object includes this behaviour:

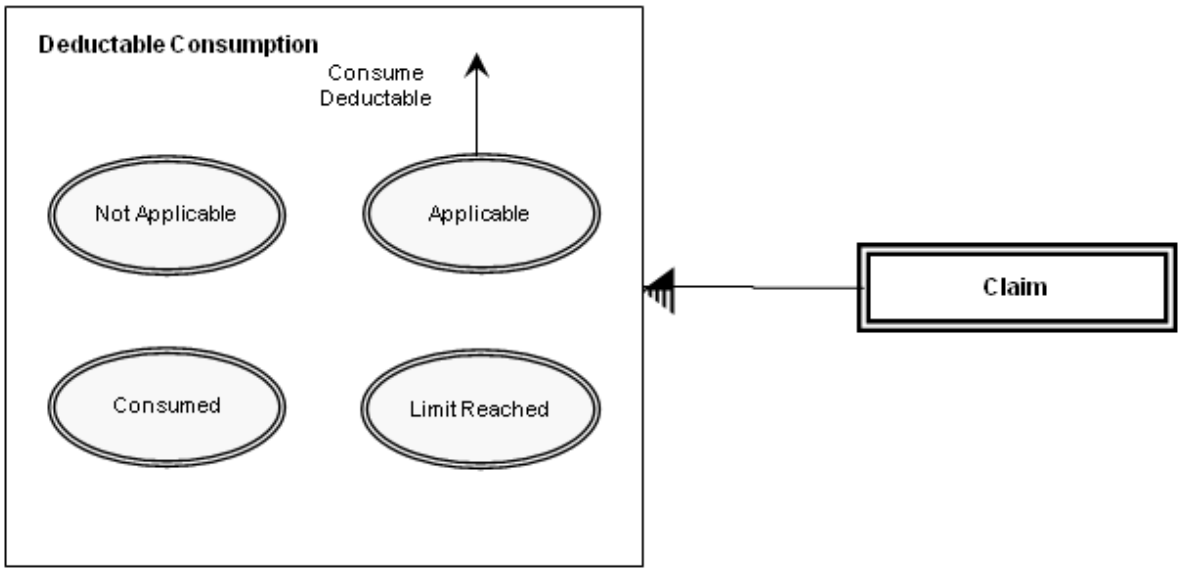


Figure 19 Deductible Consumption

The states of this object reflect how the claim is impacted by the mandatory deductible:

- Unknown: the claim is not related to a policy coverage yet, so it is not known whether a mandatory deductible applies
- Not Applicable: the mandatory deductible does not apply for this claim.
- Applicable: the mandatory deductible does apply but has not been calculated yet.
- Consumed: the mandatory deductible is calculated for the claim
- Limit Reached: the mandatory deductible does apply for the claim, but the member has completely consumed the deductible.

The calculation of the mandatory deductible is added as an additional last step in the claims processing:

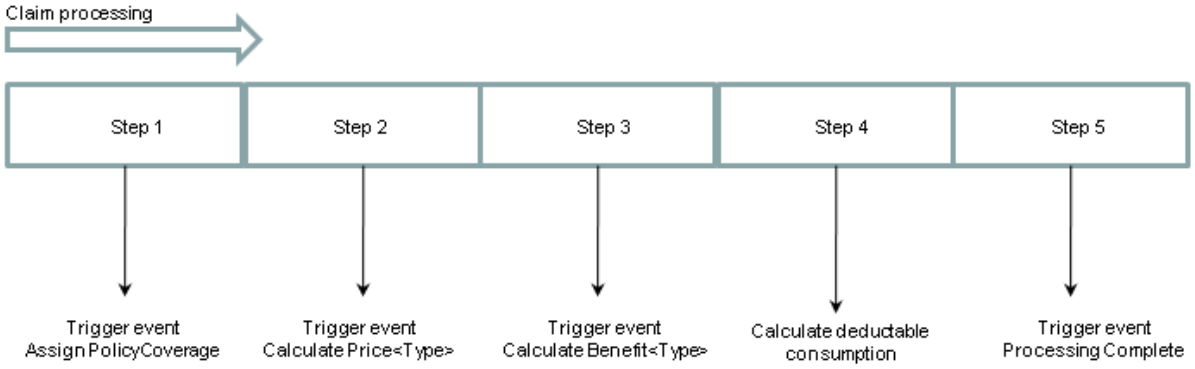


Figure 20 Claim Processing Steps with deductible calculation

(See 5.4.4.4 “Price- and Benefit calculation” for the steps in the original model).

The deductible is calculated using the callback code below:

```
String deductibleState=claim.getState("Deductible Consumption");
if (!("Not Applicable".equals(deductibleState))) {
    int deductibleAmount = 0;
    int unConsumedDeducatable = policy.getCurrency("PolicyDeductible",
"Unconsumed Amount");
    int benefitAmount = claim.getCurrency("Benefit Amount");

    if (benefitAmount <= unConsumedDeducatable) {
        deductibleAmount = benefitAmount;
    }
    else {
        deductibleAmount = unConsumedDeducatable;
    }
    if (deductibleAmount > 0) {
        Event consumeDeductible = this.createEvent("Consume Deductible");
        consumeDeductible.setInstance("Deductible Consumption", claim);
        consumeDeductible.setCurrency("Deductible Amount",
deductibleAmount);
        consumeDeductible.submitToModel();
    }
}
```

The current unconsumed deductible is retrieved from the PolicyDeductible behaviour. This behaviour is included in the Policy object and has three derived attributes:

1. Deductible Amount: the deductible amount applicable for the policy, depending on product and member age.
2. Consumed Amount: the total consumed mandatory deductible of the policy, calculated as ∑DeductibleConsumption.Deductible Amount of all claims of the policy.
3. Unconsumed Amount: Deductible Amount - Consumed Amount.

A new derived attribute Reimbursed Amount is added to the claim object with value Reimbursed Amount = Benefit Amount – Deductible Amount. The Reimbursed Amount has the value that finally will be paid to the member.

7.4. Summary

This section summarizes the extensions needed to the original model to support the new concept of mandatory deductibles. The following extensions to the model are made:

- A new behaviour Deductible is included by object Product. Deductible reuses the existing AgeLimit behaviour.
- A new attribute Indicator Mandatory Deductible is added to the behaviour Coverage
- A new behaviour Deductible Consumption is included by object Claim.
- A new behaviour PolicyDeductible is included in the object Policy
- A new derived attribute Reimbursed Amount is added to object Claim.
- Callback ProcessClaim is modified to incorporate the additional step of deductible calculation.

So apart from the modification of callback ProcessClaim, all extensions are additions of new behaviours and attributes. All other parts of the original model are not touched. This illustrates the power of the mixin composition style of Protocol Modelling.

8. Conclusions and Discussions

This chapter presents conclusions from the research, answers the research questions and discusses future research.

8.1. Conclusions

The goal for the research is to answer this research question:
Which semantic constructs reduce the impact of changes on a protocol model of a healthcare insurance?

This research question is detailed in the following sub questions:

1. Which flexibility is needed for a healthcare insurance model? In other words: which types of changes occur in the healthcare insurance domain?
2. Which semantic constructs of Protocol Modelling support the needed flexibility?
3. How can the semantic construct best be applied?

Next sections each deal with a sub question.

8.1.1. Flexibility needed in a Healthcare Insurance Model

Section 6.1 "Overview and Classification of Changes" classifies changes in the Base Insurance since 2006 as one of:

1) Change in covered care procedures. Two subclasses are defined:
   a) Coverage Added. A care procedure that was uncovered previously, has become a covered procedure.
   b) Coverage Removed: A care procedure that was covered previously, has become covered.
2) Condition Changed: A care procedure that was covered before, is still covered but the conditions for coverage have become more restrictive or less restrictive.
3) Change in benefit calculation. The algorithm to calculate the benefit amount of a claim has changed.
4) Other Change: Changes that do not belong to one of the three categories above.

From Table 6 Overview of Changes, it can be concluded that:

- Changes in covered care procedures occur each and every year.
- Changes in conditions occur in most years. Most changes concern changes in age limits.
- Change in benefit calculation in fact happens once: when the mandatory deductible is introduced in 2008. In later years only the deductible mount is increased.
- Other change: the only change of this type is the introduction of a personal budget for visual aids. This is outside the scope of the model as this does not impact claims processing.

So, to answer the first research sub question, a healthcare insurance model needs to supply the following flexibility:

- Flexibility in addition and removal of covered care procedures.
- Flexibility in conditional coverage. Conditions of different types (age, treatment) need to be supported. Conditions need to be parameterized.
- Flexible deductible. For the Base Insurance an age and care procedure dependent mandatory deductible should be supported.

8.1.2. Flexibility Support in Protocol Modelling

This section describes different ways to construct flexible protocol models. For more details, see section 4.5 “Modelling Adaptation using Protocol Modelling”.

Parameterization can be implemented in a protocol model by defining separate parameter objects. The parameter values can be added as attributes to the parameter objects. If needed, a separate Functional Management role can be added to set/change parameter values. Parameter objects can be related to operational objects or retrieved in callback code. See section 4.5.1 for details.

User Exits can be implemented in a protocol model by firing an event from a predefined place in the model. Customers can implement the event handling in a callback. See section 4.5.2 for details.

Composition and *derived* attributes and states are semantics constructs that also can increase model flexibility:

- With composition, one can define more complex machines from simpler ones. This enables the extension of a model by including new machines.
- Derived attributes and states can abstract over implementation details and thus become a dependency wall.

To answer the second research sub question: flexibility can be incorporated in the protocol model by using:

- Parameterization.
- User Exits.
- Composition.
- Derived Attributes and States.

8.1.3. Flexibel Protocol Model of Healthcare Insurance

This section describes how the flexibility support described in the previous section, is used to create a flexible healthcare insurance model. This model is described in detail in chapter 5 “Design Protocol Model for Base Insurance 2006”.
The information in this section as a whole is the answer to the third research sub question.

Flexibility in addition and removal of covered care procedures is implemented in the model by the objects Product, Coverage and Care Procedure. A Product consists of multiple Coverages and a Coverage consists of multiple Care Procedure. So addition and removal of coverages requires only the change of model data by the Functional Management Role.

Flexibility in conditional coverage is implemented in the model by defining behaviours for each different condition:

- AgeLimit is used for conditional coverage depending on the age of the member.
- TreatmentLimit is used for conditional coverage depending on the sequence number of the treatment.
- MaximumNumberLimit is used for coverage up to a maximum number of procedures.

These behaviours can be included by (subtypes of) Coverage objects. Inclusion of multiple behaviours in a single Coverage object is possible, leveraging the composition semantics of Protocol Modelling. Necessary attributes are added to support parameterization. For example AgeLimit has attributes “Age From” and “Age To”.
The conditions are checked in the context of a claim by Condition behaviours. Each condition behaviour is related to a Limit behaviour so it can get the required parameters. For example AgeCondition is related to AgeLimit.

Algorithmic flexibility is implemented for both Price- and Benefit calculation. These calculations can differ per coverage, depending on the included behaviour. For example PolicyCoverageFull includes FixedPrice and BenefitFull, whereas PolicyCoveragePercentage includes FixedPrice and BenefitPercentage. New price- and benefit calculations can be added by defining new behaviour and include them in the appropriate PolicyCoverage. So the price- and benefit calculations are implemented as *User Exits* customers can change themselves.

The mandatory deductible required an extension of the model. The model supports a mandatory deductible using a variable amount and age limit. Per coverage it can be indicated whether the procedures in the coverage are subject to a mandatory deductible.

8.1.4. Impact of changes

The changes of the base insurance from 2007 until 2011 were exposed to the model. Only the introduction of the mandatory deductible required a *structural* change of the model. The required structural change involved adding new attributes and behaviours. Existing behaviours and attributes were (almost) not impacted at all.

All other changes could be implemented by changing the model *configuration* only. See Table 7 Change Types and Model Impact.

So it is concluded that the developed model is flexible and supports common changes in healthcare insurances:

- Changes in covered care procedures.
- Changes in conditions of coverage.

The initially developed model could easily be extended to support the new concept of a mandatory deductible.

To answer the research question: the flexibility of the model was achieved by:

1. Parameterization.
2. User Exits for Price- and Benefit calculation.
3. Composing conditions and price- and benefit calculations into PolicyCoverage objects.

The first two options can also be achieved by other modelling techniques. The third option of composition leverages the composition semantics of Protocol Modelling. This enables the reuse of model elements.

8.2. Discussions

8.2.1. Validity

The developed and extended model only implements the Base Insurance. Also, only the changes in the Base Insurance are classified and their impact on the model is determined. It is however expected that many of the findings in this research also apply to other healthcare insurances in- and outside of the Netherlands:

- The developed model does not contain any Base Insurance specific elements, but uses generalized abstractions.
- It is expected that changes in other healthcare insurance products have similar classifications.

More research is however needed to validate the second point.
Besides appliance in healthcare insurances in other countries, concepts of this model could also be used in other insurance types like travel insurances. More research is needed to

discover the different types of insurances and how the concepts defined in this research also apply to those other insurances. It is expected that some insurance types are more similar to healthcare insurance than others. A factor might be the policy-claim ratio. For healthcare insurance, many claims are created per policy, whereas for life insurances at most one claim per policy can exist.

8.2.2. General Modelling Techniques

This research also resulted in the implementation of parameterization and user exists in a protocol model. These techniques are generic and can be applied to all type of models. The implementation of the claims processing steps as described 5.4.4.4 "Price- and Benefit calculation" uses the Template Pattern (Gamma et al, 1995): the order and number of steps is fixed, but implementation can vary (per coverage in this case). This is also a technique that can be applied everywhere.

8.2.3. Guidelines

As said in the previous section, model flexibility is implemented using (among others) composition and derived attributes and states. At least derived states and composition are specific to Protocol Modelling. No guidelines or standards exist that define how these concepts should be applied to real life problems. Guidelines and standards are needed to increase the acceptance of Protocol Modelling. More research is needed to come up with such standards and guidelines.

8.2.4. Completeness

The developed model is by no means functional complete. Important parts are missing:

- The ability to handle multiple insurance products. Most members not only have the Base Insurance, but also multiple supplementary insurances.
- The ability to handle multiple members on a policy.
- Authorizations. For many care procedures, an approval is needed beforehand. The existence of the authorization is checked when the costs are claimed.

More research is needed to analyze the requirements of these parts and extend the developed model.

9. References

Ahituv, N., Neumann, S., & Zviran, M. (2002). A System Development Methodology for ERP Systems. *Journal of Computer Information Systems, Vol. 42, (3), Spring 2002*, 56-67.

Boehm (2002): Get Ready for Agile Methods, with Care. *Computer Volume 35 Issue 1, January 2002*, 64–69.

Brehm L., Heinzl A., & Markus M.L. (2000). Tailoring ERP Systems: A Spectrum of Choices and their Implications. *Proceedings of the 34th Hawaii International Conference on System Sciences. Maui, Hawaii*, 3-6.

Xu, L., & Brinkkemper, S. (2005). Concepts of Product Software: Paving the Road for Urgently Needed Research. *First International Workshop on Philosophical Foundations of Information Systems Engineering, LNCS, Springer-Verlag*.

Carney, D. (1997). Assembling Large Systems from COTS Components: Opportunities, Cautions, and Complexities. *SEI Monographs on Use of Commercial Software in Government Systems,*
Software Engineering Institute.

France, R., & Rumpe, B. (2007). Model-driven Development of Complex Software: A Research
Roadmap. *Proceedings of the International Conference on Software Engineering*, 37-54.

Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal 45 (3)*, 451–461.

Machado, J.P., & Menezes P.B. (2006). Defining Atomic Composition in UML Behavioral Diagrams. *Journal of Universal Computer Science, 12(7)*, 958–979.

Mellor S.J., Clark A.N., & T. Futagami (2003), Model-Driven Development. *IEEE Software, no. 5*, 14-18.

Morisio M., & Torchiano M. (2002). Definition and classification of COTS: a proposal. *1st International Conference on COTS-Based Software Systems (ICCBSS).*.

McNeile, A., & Roubtsova, E. (2009). Composition Semantics for Executable and Evolvable Behavioural Modeling in MDA. BM-MDA'09. *Proceedings of the 1st Workshop on Behaviour Modelling in Model-Driven Architecture*, 1-8.

McNeile, A., & Simons N (2006). Protocol Modelling. A modelling approach that supports reusable behavioural abstractions. *Software and System Modeling*, 5(1), 91-107.

McNeile, A. & Roubtsova, E (2008). Executable Protocol Models as a Requirements Engineering
Tool. *Proceedings of the 41st Annual Simulation Symposium*, 95–102.

McNeile, A.,& Roubtsova, E. (2008-2). CSP parallel composition of aspect models. *Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling*, 13–18.

McNeile, A., & Simons, N. (2004). Mixin Based Behaviour Modelling. *Proceedings of the 6th International Conference on Enterprise Information Systems (3),* 79-183.

Memmel, T., Bock, C., & Reiterer, H.(2007). Model-driven prototyping for corporate software spe-cification. *Proceedings. of the Engineering Interactive Systems Conference EIS'2007*.

Menzis. (2006). *Vergoedingenoverzicht*. Retrieved November 1, 2010, from http://www.tcfzorg.nl/pdf/vergoedingenoverzicht_menzis.pdf

Menzis, (2007). *Wijzigingen in de polisvoorwaarden Basisverzekering, Aanvullende Verzekeringen en Tandartsverzekeringen.* Retrieved November 1, 2010, from http://www.polisvoorwaardenonline.nl/docs/polisvoorwaarden/menzis/zorg/128235_622_1163071181503-br-1115-1106_wijzigingen_bv_av_tv.pdf

Metamaxim. *Modelscope 2.0. Modellers' Guide.* Retrieved June 1, 2010, from http://www.metamaxim.com/

OMG (2009-2). *Uni¯ed Modeling Language, Superstructure, v2.2.* OMG Document formal/09-02-02 Minor revision to UML, v2.1.2. Supersedes formal 2007-11-02, 2009.

Roubtsova, E., Wedemeijer, L., Lemmen, K., & McNeile, A. (2009). Modular Behaviour Modelling of Service Providing Business Processes. *Proceedings of the International Conference on Enterprise Information Systems,* 338-341.

Rumpe, B. (2004). Agile modeling with the UML. *Radical innovations of software and systems engineering in the future. 9th International Workshop.*

Sawyer, S. (2000). Packaged software: Implications of the differences from custom approaches to software development. *European Journal of Information System, 9*, 47–58.

Selic, B (2003): The pragmatics of model-driven development. IEEE Softw., 19–25. (Special issue on model-driven development.)

Vektis. Prestatiecodelijsten. Beschikbaar op http://tog.vektis.nl/

Wortmann, J.C., Kusters, R.J.: Cursus Bedrijfsprocessen (B44322)

Y. Yang, J. Bhuta, D. Port, and B. Boehm, Value-based processes for COTS-based applications, IEEE Software (July/August 2005), 54–62.

Zorgverzekering (2011) http://www.zorgverzekering-informatie.nl/index.php/wijzigingen-zorgverzekering

10. Appendix 1: Usage of Models

Wortmann and Kusters define a *model* as:
A formal representation of a limited number of aspects of reality developed for a specific purpose.
Some aspects in this definition draw attention:

1. It is a *formal* representation. So the meaning of elements of the model is defined.
2. It is a presentation of a *limited number of aspects*. Which aspects of reality are included in the model and which are not included, depends on the goal of the model.

A model can be used to assess characteristics of a future system, even before it is built. A model also enables knowledge transfer of a design (Milicev, 2009).

By modelling, four goals can be reached (Booch, 1999):

1. Visualisation. The model visualizes current and future operation of a system.
2. Specification. The model defines structure and behaviour of a system.
3. Construction. The model behaves as a template for the construction of the system.
4. Documentation. The model stores design decisions.

For a software development organization, models can be used internally and externally:

- Internally for the transfer of requirements and specifications between various project roles like analysis, design, build and test.
- Externally in the communication with new and existing customers.

It is important that the model can be understood by all stakeholders.

For reaching the four goals, models are used as *engineering model* in lots of technical disciplines like civil and electrical engineering.
Models can also be used for the construction of software. An important difference with other technical disciplines: the model and the resulting system are constructed of the same "material", software.
If the model is a *formal* specification, the model can be executed after a transformation or interpretation step. The model is called *executable* in that case. Model Driven Development deals with the development of software using modelling languages and modelling tools (Milicev, 2009).

Bernhard Rumpe (2004) differentiates between two current trends that influence software engineering:

1. Model Driven Development, where the *model* is central.
2. Agile Development, where *source code* is central.

One could consider those trends as opposites, but it is also possible and desirable to combine elements of both trends.
This is also the opinion of Barry Boehm, talking about plan-driven and agile software development methods (Boehm, 2002):
Although many of their advocates consider the agile and plan-driven software development methods polar opposites, synthesizing the two can provide developers with a comprehensive spectrum of tools and options.
According to Boehm, plan-driven development, and also the usage of models, is most appropriate for projects with these characteristics:

- The requirements are stable and known at an early stage.
- The architecture is developed for current and future requirements.
- Bigger teams and products.
- Required reliability.

10.1.1.1. Model Driven Prototyping

As said before, all stakeholders must be able to understand a model. Memmel, Bock and Reiterer (2007) point out that this is not the case with the widely used UML: *"Apart from software engineers, other stakeholders usually cannot understand UML".*
According to them, text based methods to gather and document requirements lead to frustrating communication problems between business- and development teams.
They suggest preventing these problems by using *prototypes* during requirements gathering. These prototypes are model-based.
These prototypes also assist when the system is ultimately built: "*to build ... a system with the help of a running simulation (prototype) is much easier than doing it from scratch based on textual descriptions.*"

10.1.1.2. Model Driven Development

Model Driven Development (MDD) goes a step further compared to Model Driven Prototyping: the model is either automatically translated into a working system, or the model is the working system itself.

A number of developments have lead to Model Driven Development:

- Increased complexity of platforms like J2EE and .NET. This increase can't be handled anymore by existing general-purpose languages. This leads to a complexity ceiling.
- Increased complexity of the software to be developed (France & Rumpe, 2007).

France and Rumpe mention a *problem-implementation gap*: there is a big gap between problem domain and software implementation domain. MDD can play a role in bridging the gap by hiding developers from implementation details.

Mellor, Clark and Futagami (2003) see these benefits for MDD:

- Enables reuse on domain level
- Increases quality of software by continuously improved models.
- Lowers costs by automating software development processes.
- Lengthens lifetime of applications by simplifying migrations to other platforms.

France and Rumpe also mention testing and simulation using models.
MDD can be used in several ways. Hailpern and Tarr (2006) divide the MDD community into three categories (with increased order of model penetration):

1. *Sketchers* model only a part of a system for communication and documentation purposes.
2. *Blueprinters* make detailed models of a design and transfer them to implementers.
3. *Model programmers* make models with executable semantics.

According to Selic (2003), MDD must meet the following conditions:

- MDD must result in complete programs, not code skeletons only.
- Automatic verification of models must be possible.

11.Appendix 2: The Dutch Base Insurance (2006)

Dutch law defines what is covered by the Base Insurance, but the government does not publish an easy to understand overview of the coverages. So the policy conditions of insurance provider Menzis (Menzis, 2006) are used as basis for the design of the model. The highlighted lines are used for defining a use case in section 5.1.2 ”Use Cases”. As this is a ‘source’ document, it is presented in the original Dutch language. For terminology used in use cases, a translation is provided in the second section of this appendix.

Table 8 Vergoedingenoverzicht Basisverzekering 2006

Behandeling	Vergoeding
Alternatieve geneesmiddelen	
Ambulancevervoer	100%
Audiologisch centrum	100%
Bevalling en Kraamzorg	
- Delivery poliklinisch	100%
(medisch noodzakelijk)	
- Bevalling poliklinisch	- verloskundige zorg: 100%
(niet-medisch noodzakelijk)	- polikliniek: gedeeltelijke vergoeding
- Communicatiemiddel	
- Kraampakket	
- Kraamzorg	100% (er geldt een eigen bijdrage)
- Kraamzorg na adoptie	
- Kraamzorg na couveuseopname	
- Meerlingenuitkering	
Bezoeks- en verblijfskosten	
- Logeerhuizen	
- Ziekenhuis/revalidatiecentrum	
Buitenland	
- Spoedeisende zorg	100%
- Niet-spoedeisende zorg	100%
- Hulpverlening door Alarmcentrale	
- Vervoer naar Nederland	
- Vervoer bij overlijden	
Chronisch intermitterende beademing	100%
Dieet advisering	Max. 4 uur behandeling p.kljr
Dieetpreparaten	100%
Erfelijkheidsonderzoek	100%
Ergotherapie	Max. 10 behandeluren p.p.p.kljr
Farmaceutische zorg	100% (conform Regeling Zorg-
	verzekering, vergoedingssysteem GVS)
Fertiliteitsbehandelingen	
- IVF and ICSI	2e en 3e behandeling
- IUI-OI (onderzoek en	
specialistenkosten)	
- Medicatie fertiliteitsbehandelingen	100%

Behandeling	Vergoeding
Fysio- en oefentherapie Cesar/	
Mensendieck	
- Gecontracteerde therapeut	100% vanaf behandeling 10 bij
	chronische indicaties
- Niet-gecontracteerde therapeut	100% vanaf behandeling 10 bij
	chronische indicaties conform
	Verzekeringsreglement Zorg
Gezondheidscursussen	
Handicap, vakantie en begeleiding	
Herstellingsoorden	
Huidtherapieën	
- Acnétherapie	
- Camouflagetherapie	
- Camouflagemiddelen	
- Epilatie	
- Psoriasisdagbehandeling	
- UVB-lichttherapie	
Huisarts	100%
Hulpmiddelen	
- Hulpmiddelen	100% (conform regeling Zorgverzekering, voor bepaalde hulpmiddelen geldt een maximale vergoeding of eigen bijdrage)
- Alarmeringsapparatuur	
(sociale indicatie)	
- Bewakingsmonitor voor baby's	
- Brillenglazen/contactlenzen	
1) Alle sterktes	
2) Arrangementen:	
a) Specsavers (brillen en	
contactlenzen)	
b) Hans Anders (brillen)	
c) Het Huis (brillen)	
U kunt max. 1x per 2 jaar gebruik	
maken van een vergoeding of van	
een arrangement.	
- Hoortoestellen	
- Plaswekker	
- Pruiken	
- Orthopedisch schoeisel	
- Orthopedische steunzolen	
- Softbraces	
- Steunpessarium	
Kinderopvang	

Behandeling	Vergoeding
Leukemie bij kinderen	100%
Logopedie	100%
Manuele lymfedrainage door	100%
Huidtherapeut	100%
Orgaantransplantatie	
Overgangsconsulente	
Patiëntenverenigingen	
- Lidmaatschappen	
- Therapieën	
Podotherapie	
Poliklinische zorg	100%
Preventie	
- Algemene Check-up	
- Griepvaccinatie	
- Reizen naar het buitenland	
- Vaccinatie Hepatitis-B	
Psychologische zorg	
Psychotherapie	AWBZ
Revalidatie	100%
Second opinion	
Sport Medisch Advies	
Sterilisatie	
Stottertherapie	
Thuiszorg	AWBZ
Trombosedienst	100%
Vakantiereizen Rode Kruis of	
Zonnebloem	
Vervangende mantelzorg tijdens	
Vakantie	
Verbandmiddelen	100%
Verpleegartikelen	AWBZ
Voorbehoedsmiddelen	
(anticonceptiva)	
Ziekenhuisopname	100%
Zittend ziekenvervoer	Indien voldaan aan criteria, max. € 0,22 per km. of laagste tarief openbaar vervoer, met een eigen bijdrage van € 83,-
Zorgprogramma's (speciale patiëntengroepen)	

Behandeling	Vergoeding
Tandheelkundige hulp tot 18 jaar	
- In bijzondere gevallen	100%
- Kaakchirurgische behandeling	100% (na machtiging)
- Tandheelkundige implantaten	100% (na machtiging)
- Tandheelkundige zorg aan verzekerden met een lichamelijk of verstandelijk handicap	100%
- Prothesen	100%
- Kronen, bruggen en gegoten vullingen	
- 1e consult (jaarlijkse controle)	100%
- 2e en volgende consult	100%
- Incidenteel consult	100%
- Röntgenfoto's	100%
- Chirurgische ingrepen	100%
- Verdoving	100%
- Wortelkanaalbehandeling	100%
- Fluoride behandeling vanaf 6 jaar	100%
- Tandsteen verwijderen	In bijzondere gevallen: 100%
- Vullingen	100%
- Parodontologie	100% (na machtiging)
- Gnathologie	100% (na machtiging)
- Orthodontie	100%
Tandheelkundige hulp vanaf 18 jaar	
- In bijzondere gevallen	100%
- Kaakchirurgische behandeling	100% (na machtiging)
- Tandheelkundige implantaten	100% (na machtiging)
- Mesostructuctuur en prothese op implantaten	100%
- Tandheelkundige zorg aan verzekerden met een lichamelijk of verstandelijk handicap	100%
- Prothesen	a) 75% (uitgezonderd reparatie en rebasen waarvoor 100% geldt)
a) Volledig	
b) Partieel	
c) Frame	
- Kronen, bruggen en gegoten vullingen	
- 1e consult (jaarlijkse controle)	
- 2e en volgende consult	
- Incidenteel consult	
- Röntgenfoto's	
- Chirurgische ingrepen	
- Verdoving	
- Wortelkanaalbehandeling	
- Tandsteen verwijderen	
- Vullingen	
- Parodontologie	
- Orthodontie	In bijzondere gevallen: 100%

11.1. Translation of Dutch terminology

The table below translates Dutch terminology that is being used for the definition of use cases.

Table 9 Translation of Dutch terminology

Dutch	English	Used in
Alternatieve geneesmiddelen	Alternative Medicine	Use Case 1
Bevalling poliklinisch (niet-medisch noodzakelijk)	Inpatient Delivery (without diagnosis)	Use Case 5

Kraamzorg	Maternity Care	Use Case 7
Dieetadvisering	Nutritional Counselling	Use Case 4
IVF en ICSI	IVF	Use Case 8
Huisarts	General Practitioner	Use Case 2
Tandheelkundige hulp	Dental Care	Use Case 3
Fysiotherapie	Physiotherapy	Use Case 6
Prothesen	Prosteses	Use Case 9

12. Appendix 3: Changes in the coverage of the Base Insurance

This appendix describes the changes of the Base Insurance in from its start in 2007 onwards. Each paragraph describes the changes of one year.

12.1. Base Insurance Changes in 2007

The information in this section is based on (Menzis, 2007).

12.1.1. Added Coverage

The coverage is extended with:

1) Prenatal screening for congenital defects by echoscopy in the second trimester of the pregnancy, if the member is younger than 36 and there is a diagnosis.
2) The first IVF attempt per planned pregnancy.
3) Abdominoplasty (abdominal reduction by plastic surgery);
4) The possibility of a personal budget for visual aids in the case of a serious visual handicap.

Remarks:

- 1) is implemented by using different care procedure codes for members of younger than 36 without diagnosis.
- 4) is out of scope for the model: when a personal budget applies, the costs are not claimed anymore.

12.2. Base Insurance Changes in 2008

The information in this section is based on (Zorgverzekering, 2011).

12.2.1. Added Coverage

The coverage is extended with:

1. Birth Control regardless of age.
2. Dental Care for members younger than 22.
3. Five hours of additional Maternity Care.
4. Mental Healthcare.

12.2.2. Mandatory Yearly Deductible

A mandatory Yearly Deductible of 150 Euro is defined. Medical costs up to this amount are not reimbursed by the insurance company. The Yearly Deductible applies for members of 18 years and older.

The yearly deductible does not apply to the following care procedures:

- Visits to General Practitioners.
- Obstetrical care.
- Maternity Care.
- Dental care for people younger than 22.

Chronically ill and disabled people are financially compensated. This group is selected by looking at specific medication use.

12.3. Base Insurance Changes in 2009

The information in this section is based on (Zorgverzekering, 2011).

12.3.1. Added Coverage

The coverage is extended with:

1. The diagnosis and treatment of severe dyslexia for children born after 1 January 2001. The school functions as a gatekeeper[5].

12.3.2. Reduced Coverage

No longer covered are:

1. Lift Chairs for elderly and disabled people.
2. Hypnotics and tranquillizers.

12.3.3. Increase of Mandatory Yearly Deductible

The mandatory yearly deductible is increased to 155 Euro.

12.4. Base Insurance Changes in 2010

The information in this section is based on (Zorgverzekering, 2011).

12.4.1. Added Coverage

The coverage is extended with:

1. Organ transplantation outside the European Union/EEA (under conditions[3]).
2. Mandibular advancement devices for treatment of obstructive sleep apnea syndrome[3].

12.4.2. Removed Coverage

1. Compensation for the mucolytic agent acetylcysteine.

12.4.3. Increase of Mandatory Yearly Deductible

The mandatory yearly deductible is increased to 165 Euro.

12.5. Base Insurance Changes in 2011

The information in this section is based on (Zorgverzekering, 2011).

12.5.1. Reduced Coverage

No longer covered are:

1. Birth Control (except for women younger than 21 years).
2. Dental Care for 18 to 21-year-olds.
3. Compensation for Durable Medical Equipment.
4. Simple extractions by oral surgeons.

12.5.2. Increase of Mandatory Yearly Deductible

The mandatory yearly deductible is increased to 170 Euro.

[5] Authorization is required before costs can be claimed.

13.Appendix 4: Explanation of graphical symbols

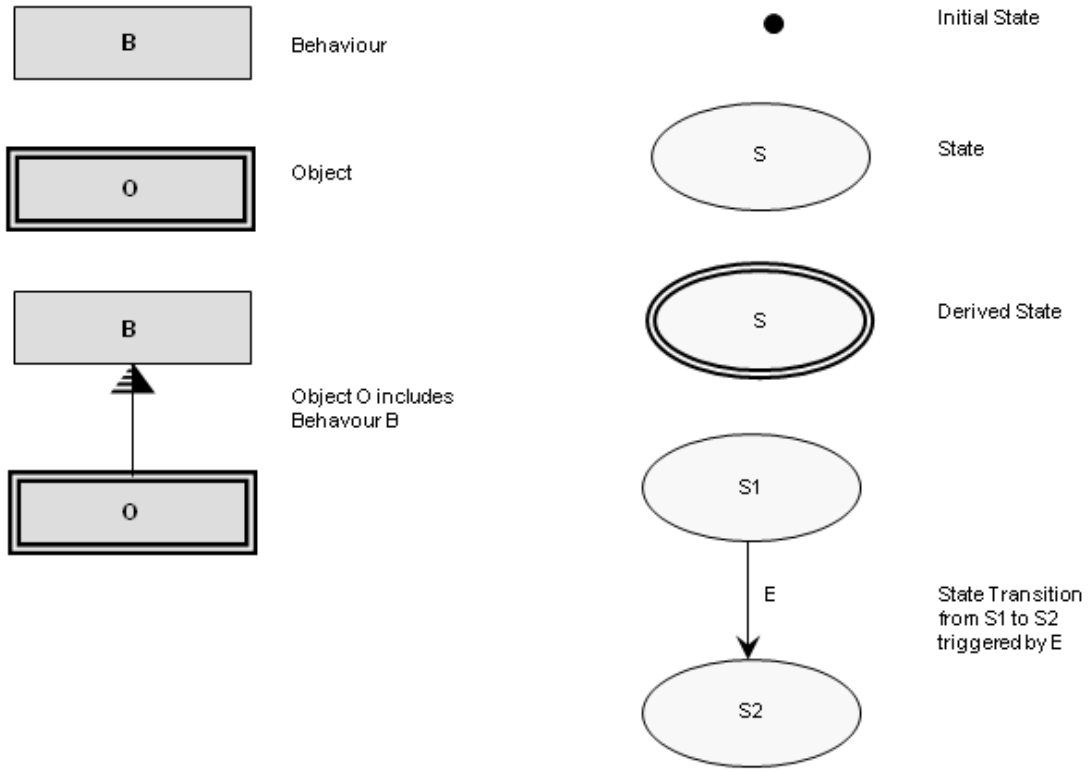


Figure 21 Symbols used in Protocol Models

14. Appendix 5: UML Behaviour Models

This appendix provides a limited overview of the possibilities UML offers to create executable behaviour models. A complete description of UML is outside the scope of this research. Only a few important points are mentioned that are relevant to the modelling of the Base Insurance. Statements about UML in this chapter are based on version 2.2 of the UML specification (OMG, 2009-2).

1. Section "UML types of Behaviour Models" summarizes the different UML behaviour models.
2. Section "UML State Machine Semantics" describes the semantics of UML state machines.
3. Section "UML State Machines and Transactions" describes why modelling transactional behaviour is essential for business information systems and why Protocol Modelling is better suited to do that compared to UML State Machines.

14.1. UML types of Behaviour Models

UML offers a number of modelling techniques that are meant to model behaviour:

- *Interaction models* describe the communication between instances of objects.
- *Activity Models* describe the sequence and (conditional) execution of steps in an activity.
- *State machine models* describe the status transitions of an object as a response to events.

McNeile and Roubtsova assess these modelling techniques on the basis of suitability for creating an executable model (McNeile & Roubtsova, 2009):

- Interaction models describe scenarios. They give examples of possible interaction patterns. For an executable model, however, it is important that all scenarios can be defined. This is not possible with interaction models. This is also confirmed by the UML specification: "*The traces that are not included are not described by this Interaction at all, and we cannot know whether they are valid or invalid.*"
- Activity models are basically executable. However, they describe "*lower-level behaviors, rather than which classifiers [i.e., classes] own those behaviors*" (UML). They are not meant to model behaviour on object level.
- State machine models "*can be used for modelling discrete behaviour through finite statetransition systems*" (UML).

So state machine models are best suited for modelling the complete behaviour of objects. The next paragraph explains their semantics.

14.2. UML State Machine Semantics

UML has two types of state machines:

1. Behavioural state machines. "*State machines can be used to specify behaviour of various model elements*" (UML specification).
2. Protocol state machines. "*Protocol state machines are used to express usage protocols*".

It's obvious that behavioural state machines are meant to model behaviour. Their semantics can be summarized as follows:

- A state machine has a set of *states*.
- An *event* can result in the transition of the state machine to a new state.
- A *guard* is a condition attached to a transition. The transition only occurs when the guard has the true value. A guard is not allowed to have a side effect.
- States can be hierarchical: a *composite* state consisting of *substates*.

- A state machine can have multiple *regions:* Each region has its own states and transitions. The state of the state machine is the combined set of states of the regions.

Behavioural state machines can respond in different ways to events:

- The current active state has an enabled transition for the event. The event will result in a state transition to the new state.
- The state machine can *defer* the processing of the event to some later moment.
- The state machine can *ignore* the event.

These semantics make UML behaviour state machines less suitable for use in the domain of Business Information Systems, where transactional integrity is important. The next section will explain why.

14.3. UML State Machines and Transactions

In Business Information Systems, an event impacts multiple objects in many cases. It is important that processing of the events either results in:

- All objects going to their new states.
- All objects remaining in their original state when one or more of the involved objects fails to process the events.

This is called transactional integrity. An obvious example is the transfer of money between bank accounts where it should never happen that money 'is created' or 'disappears'.

Transactional integrity is also important when processing claims. Consider the following example:

- The event 'Submit Claim' causes the claim object to go to the Submitted state.
- A claim can only go to the Submitted State when it's not a duplicate. (Insurance companies have fraud detection in place to detect the illegal double declaration of the same medical treatment).
- The event 'Submit Claim' causes the increment of the state variable Deductible Amount of the PolicyDeductible object. When the Deductible threshold is reached, the Policy Object transitions to the Deductible limit reached.

See the UML state machines below:

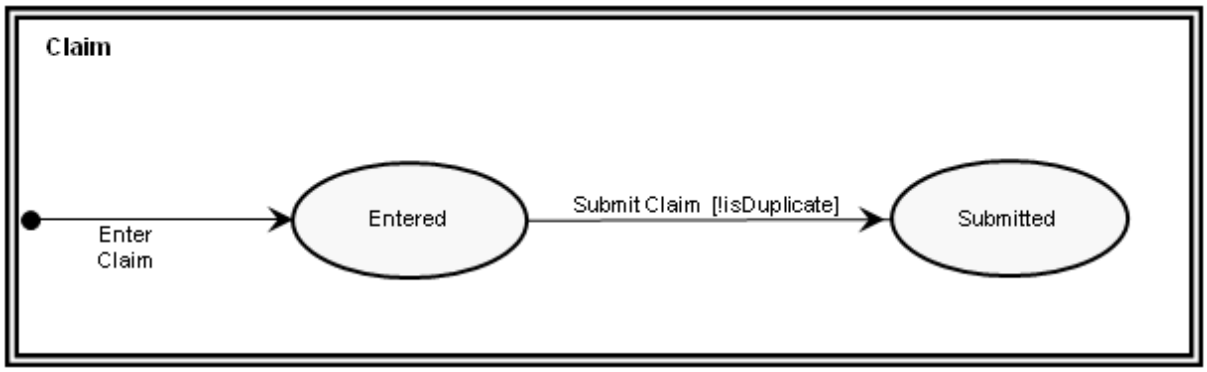


Figure 22 UML state machine for Claim

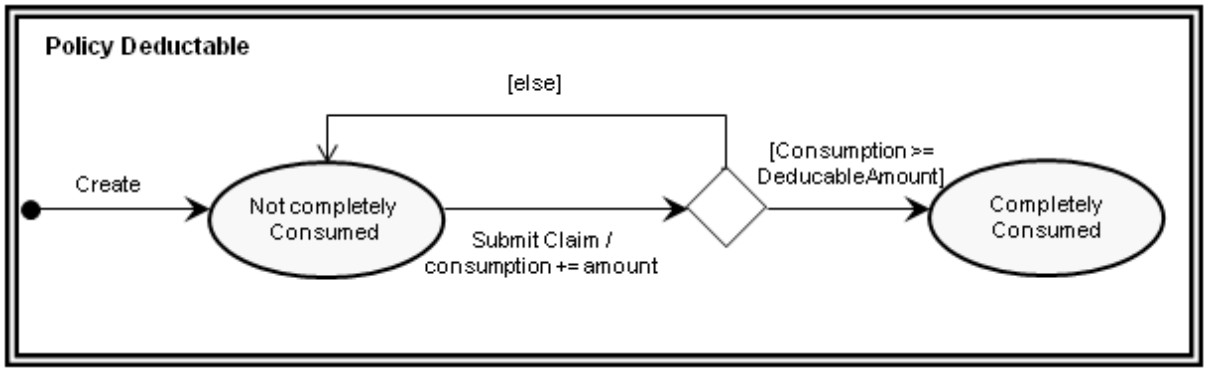


Figure 23 UML state machine for PolicyDeductible

So the event 'Submit Claim' event impacts the objects Claim and PolicyDeductible. Because of the required transactional integrity, when a duplicate claim is submitted, the claim object should not change its state, nor should the PolicyDeductible object change.

This transactional behaviour cannot be modelled in an UML state machine. A UML state machine has the choice to accept, ignore or defer an event but they do that independent of each other. In this example, Policy Deductible should only accept the Submit Claim when the Submit Claim event in Claim leads to a transition to the Submitted state. Using UML state machines, this can be implemented by:

- Also doing the isDuplicate check in the PolicyDeductible. This leads to unacceptable redundancy.
- Only present Submit Claim events to the Claim and PolicyDeductible state machines for non-duplicate claims. This moves the whole burden of business rules checking to calling clients.

As can be seen, both options have significant and unacceptable disadvantages.
Machado and Menezes also report that "UML seems to lack compositional constructs for defining atomic actions/activities/operations" (Machado and Menezes, 2006)

These disadvantages disappear when using Protocol Modelling where a Protocol Machine also can *refuse* an event. When any of the involved protocol machines refuses the event, all of the protocol machines remain in their original state. So the refusal of an event results in 'rolling back the transaction'. The ability to refuse an event is essential for transactional business systems:
"Without the ability to refuse events, state machines cannot describe event protocols in situations where an event must be accepted by multiple objects, which is usual in transactional business systems" (McNeile and Simons, 2006).

So it is included that the semantics of Protocol Machines as described in section 4.4 "Protocol Modelling Semantics" make Protocol Machines better suited than UML state machines to describe the behaviour of Business Information Systems.

15.Appendix 6: Model Reference

This appendix gives detailed descriptions of the behaviours and objects of the developed model.

15.1. Behaviour AgeCondition

This behaviour only accepts event ProcessClaim if the age of the member at the claim date is within the defined age limits.

15.2. Behaviour AgeLimit

This behaviour must be added to all coverages where the age of the member determines the coverage.

Table 10 Behaviour AgeLimit

Attribute	Description	Example
Age From	Lower bound of allowed age	0
Age To	Upper bound of allowed age	18

15.3. Behaviour Benefit

This behaviour stores the benefit amount calculated by the claims processing.

Table 11 Behaviour Benefit

Attribute	Description	Example
Benefit Amount	Benefit Amount	10.00

15.4. Behaviour BenefitCoPayment

This behaviour must be included by all policy coverages where a co-payment should be deducted.

Table 12 Attributes of BenefitCoPayment

Attribute	Description	Example
Benefit Name	A derived “technical” attribute. See section “Modelling of the application of coverage rules”.	
Co-payment Amount	The co-payment per unit.	261.50

15.5. Behaviour BenefitFull

This behaviour must be included by all policy coverages that cover the full price of the care procedure.

Table 13 Attributes of BenefitCoPayment

Attribute	Description	Example
Benefit Name	A derived “technical” attribute. See section “Modelling of the application of coverage rules”.	

15.6. Behaviour BenefitPercentage

This behaviour must be included by all coverages that only cover a percentage of the cost.

Table 14 Attributes of BenefitPercentage

Attribute	Description	Example
Benefit Name	A derived “technical” attribute. See	

	section “Modelling of the application of coverage rules”.	
Percentage	The percentage that is covered	75

15.7. Object CareProcedure

A *care procedure* is a definition of a medical treatment.

Table 15 Attributes of CareProcedure

Attribute	Description	Example
Code	Code	01/12002
Description	Description	
Price	Fixed price	13.50

15.8. Behaviour CareProcedureCondition

This behaviour only accepts event ProcessClaim if the care procedure of the claim is a member of the care procedure group of the policy coverage.

15.9. Behaviour CareProcedureGroup

A CareProcedureGroup models a set of care procedures. CareProcedureGroup is modelled separately to enable reuse in other cases where a set of procedures must be handled. This does not occur in this model however.
CareProcedureGroup has no attributes.

15.10. Object CareProcedureGroupMember

This object relates a care procedure to a care procedure group.

Table 16 Attributes of CareProcedureGroupMember

Attribute	Description	Example
CareProcedureGroup	The care procedure group	General Practitioner Care
Procedure	The procedure	01/12002
Description	A derived attribute to display context information.	General Practitioner Care-01/12002

15.11. Object Claim

The table below shows the claim attributes that are entered by a member when creating a claim.

Table 17 Attributes of Claim, entered by a member

Attribute	Description	Example
Policy	The policy used to claim.	123-456-789
Customer Reference	Label customer can assign to claim	Dentist visit
Care Procedure	The received medical treatment	01/12002
Number	Number of units	2
Service Date	Date of medical treatment	1 February 2006
Treatment	Sequence number of treatment	2

Other Claim attributes are populated during the processing of the claim. See table below:

Table 18 Derived and calculated attributes of Claim

Attribute	Description	Example
Price	The price of the care procedure	10.00

Total Amount	Derived attribute: Number * Price	20.00
Member Age	Age of the member at service date	42
Processing Info	Textual information about the processing status of the claim	Completed

15.12. Behaviour CoPayment

This behaviour stores the co-payment amount calculated by the claims processing.

Table 19 Behaviour CoPayment

Attribute	Description	Example
CoPaymentAmount	Co-payment Amount	10.00

15.13. Behaviour Coverage

The first two aspects of the model defined in 5.3.4 “Mathematical Model of Benefit Rules” are modelled in behaviour Coverage. See table below. Behaviour Coverage models a set of care procedures that are covered.

Table 20 Attributes of Coverage

Attribute	Description	Example
Product	Product the coverage belongs to	Base Insurance

Coverage includes the behaviour CareProcedureGroup. All coverages of the Base Insurance are selected based on the care procedure of the claim, so all coverages include behaviour Coverage. Other aspects, like additional conditions and benefit calculation can be added by including other behaviours (mixin approach). See next sections for examples.

15.14. Object CoverageAge

This object models a coverage with has an age condition: the claim is only covered when the age of the member at the claim date is within the specified limits.

Table 21 Attributes of CoverageAge

Attribute	Description	Example
Name	The name of the coverage	Dental Care

Includes behaviours:

- Coverage
- AgeLimit

15.15. Object CoverageCoPayment

This object models a coverage for which a co-payment is deducted.

Table 22 Attributes of CoverageCoPayment

Attribute	Description	Example
Name	The name of the coverage	Inpatient delivery

Includes behaviours:

- Coverage
- BenefitCoPayment

15.16. Object CoverageFull

This object models a coverage that covers the full price of the care procedure.

Table 23 Attributes of CoverageFull

Attribute	Description	Example
Name	The name of the coverage	General

		Practitioner Care

Includes behaviours:

- Coverage

15.17. Object CoverageMaximumNumber

This object models a coverage that covers up to a maximum number of units.

Table 24 Attributes of CoverageMaximumNumber

Attribute	Description	Example
Name	The name of the coverage.	Nutritional Counselling

Includes behaviours:

- Coverage
- MaximumNumberLimit

15.18. Object CoverageMaximumNumberCoPayment

This object models a coverage which both a co-payment amount and a maximum.

Table 25 Attributes of CoverageMaximumNumberCoPayment

Attribute	Description	Example
Name	The name of the coverage	Nutritional Counselling

This object illustrates the mixin/multiple-inheritance capabilities of protocol modelling.
The combination of maximum covered number of units and co-payment is implemented by including both the behaviours MaximumNumberLimit and BenefitCoPayment.
So this object includes the behaviours:

- Coverage
- MaximumNumberLimit
- BenefitCoPayment

15.19. Object CoveragePercentage

This object models a coverage that only covers a percentage of the cost.

Table 26 Attributes of CoveragePercentage

Attribute	Description	Example
Name	The name of the coverage	Prostheses

Includes behaviours:

- Coverage
- BenefitPercentage

15.20. Object CoverageTreatment

This object models a coverage for certain treatments only.

Table 27 Attributes of CoverageTreatment

Attribute	Description	Example
Name	The name of the coverage	IVF

Includes behaviours:

- Coverage
- TreatmentLimit

15.21. FixedPrice

This behaviour must be included by all policy coverages where the claim price is taken from the care procedure.

Table 28 Attributes of FixedPrice

Attribute	Description	Example
PriceCalculatorName	A derived "technical" attribute. See section "Modelling of the application of coverage rules".	

15.22. Behaviour MaximumNumberCondition

This behaviour only accepts event ProcessClaim if the total number of covered procedures does not exceed the defined maximum.

15.23. Behaviour MaximumNumberLimit

This behaviour must be included by all coverages that cover up to a maximum number of units.

Table 29 Attributes of MaximumNumberLimit

Attribute	Description	Example
Maximum Number	The maximum number of covered units.	4

15.24. Object Person

A *person* is a human being known by the insurance company. A person may have (had) a policy.

Table 30 Attributes of Person

Attribute	Description	Example
Person Name	Name of the person	Mrs. Johnson
Date of Birth	Date of Birth of the person	25-01-1968
Age	Age of the person now, derived from Date of Birth	43

15.25. Object Policy

A *policy* grants a person the right to claim healthcare cost covered by the policy product.

Table 31 Attributes of Policy

Attribute	Description	Example
Policy Number	Identifying number	123-456-789
Start Date	Start Date of the policy	1 January 2006
End Date	Derived attribute. Start Date + 1 year	31 December 2006
Product	The product of the policy	Base Insurance
Person	The person enrolled to the policy	John Johnson

Object Policy has only state Valid. This is a simplification of the real world.

15.26. Behaviour PolicyCoverage

A PolicyCoverage captures the state of a coverage in the context of a particular policy. A PolicyCoverage has the following attributes:

Table 32 Attributes of PolicyCoverage

Attribute	Description	Example

Policy	The policy the policy coverage belongs to.	123-456-789
Coverage	The related coverage.	Dental Care

15.27. Object PolicyCoverageAge

Again this object is very similar to PolicyCoverageFull. However, the Process Claim event is only accepted if the age of the member is within the defined age limits.
So besides CareProcedureCondition, PolicyCoverageAge also includes AgeCondition.

15.28. Object PolicyCoverageCoPayment

Again a variation of PolicyCoverageFull. It includes BenefitCoPayment. This Benefit object subtracts a co-payment per claimed unit: benefit = number * (price – co-payment).

15.29. Object PolicyCoverageFull

This object is the simplest policy coverage object. Besides PolicyCoverage, it includes behaviour BenefitFull. PolicyCoverageFull does not impose any additional condition to the claim: if the care procedure matches one of the covered procedures, the full amount (number * price) of the claim is paid.

15.30. Object PolicyCoverageMaximumNumber

This object covers the full amount (number * price), but only up to a maximum number of units per policy. So it includes behaviour MaximumNumberCondition.

15.31. Object PolicyCoverageMaximumNumberCoPayment

This object covers up to a maximum number of units per policy, after deducting a co-payment.
It is a "mixin" of PolicyCoverage, MaximumNumberCondition and BenefitCoPayment.

15.32. Object PolicyCoveragePercentage

This object is very similar to PolicyCoverageFull. Instead of BenefitFull, it includes BenefitPercentage. BenefitPercentage gets the covered percentage from the associated Coverage object and calculates the benefits: benefit = percentage * price * number/100.

15.33. Object PolicyCoverageTreatment

This behaviour includes a TreatmentCondition. A claim is only covered when the treatment sequence number is within the defined limits.

15.34. Object Product

A *product* is a set of coverages of healthcare costs. See table below. Column Example shows a typical value for attributes.

Table 33 Attributes of Product

Attribute	Description	Example
Product name	Name of the product	Base Insurance

15.35. Behaviour TreatmentCondition

This behaviour only accepts event ProcessClaim if the treatment sequence of the claim is within the defined limits.
This behaviour must be included by all behaviours that only cover certain treatments.

Table 34Attributes of TreatmentLimit

Attribute	Description	Example
Treatment From	Lower bound of covered treatment.	2
Treatment To	Upper bound of covered treatment.	3

16. Appendix 7: Use cases

This appendix shows how the model handles the use cases of the Base Insurance.

16.1. Basic setup

Some steps to be performed in preparation are described in this paragraph.

16.1.1. Product

The Base Insurance product needs to be defined by actor Functional Management. See table below:

Table 35 Definition product Base Insurance

Actor	Functional Management
Object	Product
Instance	new Product
Event	Create Product
Product Name	Base Insurance

16.1.2. Persons

Some sample persons of different type are needed: a grown up and a child.

Table 36 Register grown up

Actor	Relation Management
Object	Person
Instance	new Person
Event	Register Person
Person Name	Mrs. Johnson
Date of Birth	25 January 1968

Table 37 Register child

Actor	Relation Management
Object	Person
Instance	new Person
Event	Register Person
Person Name	John Johnson
Date of Birth	5 April 1994

16.2. Use case 1: Not covered (Alternative medicine)

An example of an uncovered treatment is care procedure "90/931001", "Acupuncture treatment". Define this care procedure as follows:

Table 38 Definition procedure "Acupuncture treatment"

Actor	Functional Management
Object	CareProcedure
Instance	new CareProcedure
Event	Create CareProcedure
Code	90/931001
Description	Acupuncture treatment
Price	0

Price is irrelevant because the cost of this care procedure is not covered by the basic insurance.

16.2.1. Create Policy

Mrs. Johnson creates a policy for the Base Insurance.

Table 39 Create Policy

Actor	Relation Management
Object	Policy
Instance	new Policy
Event	Create Policy
Person	Mrs. Johnson
Product	Base Insurance
Policy Number	Policy-1
Start Date	1 January 2006

16.2.2. Submit claim

A claim needs to be created first:

Table 40 Create claim "Acupuncture treatment"

Actor	Member
Object	Claim
Instance	new Claim
Event	Create Claim
Policy	Policy-1
Customer Reference	Acu1
CareProcedure	Acupuncture treatment
Number	1
Treatment	0 (default)
Date	1 January 2006

After creation, the claim can be submitted:

Table 41 Submit claim "Acupuncture treatment"

Actor	Member
Object	Claim
Instance	Acu1
Event	Submit Claim

Because alternative medicine is not covered, the claim will not get paid. The processing result will be:

- Benefit Amount = “0.00”
- Processing Info = “Rejected: 0 coverages found”

16.3. Use case 2: Covered 100% (General Practitioner Care)

An example of general practitioner care is care procedure 01/12000, Short Visit. Define this care procedure as follows:

Table 42 Definition procedure Short Visit

Actor	Functional Management
Object	CareProcedure
Instance	new CareProcedure
Event	Create CareProcedure
Code	01/12000
Description	Short Visit
Price	9,00

16.3.1. Coverage

General Practitioner Care is 100% covered. So setup a CoverageFull object:

Table 43 Definition coverage General Practitioner Care

Actor	Functional Management
Object	CoverageFull
Instance	new CoverageFull
Event	Create CoverageFull
Product	Base Insurance
Name	General Practitioner Care

Add procedure 01/12000 as a procedure group member:

Table 44 Definition procedure group member Short Visit

Actor	Functional Management
Object	CoverageFull
Instance	General Practitioner Care
Event	Create CareProcedureGroupMember
CareProcedureGroupMember	new CareProcedureGroupMember
CareProcedure	Short Visit

16.3.2. Create Policy

Mrs. Johnson creates a policy for the Base Insurance.

Table 45 Create Policy

Actor	Member
Object	Policy
Instance	new Policy
Event	Create Policy
Person	Mrs. Johnson
Product	Base Insurance
Policy Number	Policy-2
Start Date	1 January 2006

16.3.3. Submit claim

First a claim is created:

Table 46 Create Claim GP Short Visit

Actor	Member
Object	Claim
Instance	new Claim
Event	Create Claim
Policy	Policy-2
Customer Reference	Visit doctor
CareProcedure	Short Visit
Number	1
Treatment	0 (default)
Date	1 January 2006

And submitted:

Table 47 Submit claim GP Short Visit

Actor	Member
Object	Claim
Instance	Visit doctor
Event	Submit Claim

Because General Practitioner Care is completely covered, the benefit amount is calculated as price * number.

The processing result is:

- Benefit Amount = “9.00”

16.4. Use case 3: Covered 100% with age limit (Dental Care)

An example of Dental Care is care procedure 12/D61, “First Visit”. First this procedure is defined:

Table 48 Definition of procedure First Visit

Actor	Functional Management
Object	CareProcedure
Instance	new CareProcedure
Event	Create CareProcedure
Code	12/D61
Description	First Visit
Price	18,40

16.4.1. Coverage

Dental Care is covered up to and including the age of 18.
So define a CoverageAge for Dental Care:

Table 49 Definition coverage Dental Care

Actor	Functional Management
Object	CoverageAge
Instance	new CoverageAge
Event	Create CoverageAge
Product	Base Insurance
Name	Dental Care

Assign this care procedure to the coverage:

Table 50 Definition procedure group member First Visit

Actor	Functional Management
Object	CoverageAge
Instance	Dental Care
Event	Create CareProcedureGroupMember
CareProcedureGroupMember	new CareProcedureGroupMember
CareProcedure	First Visit

Define the age limits:

Table 51 Definition of age limits for Dental Care

Actor	Functional Management
Object	CoverageAge
Instance	Dental Care
Event	Change AgeLimit
Age From	0
Age To	18

16.4.2. Create Policy

Mrs. Johnson and John Johnson both create a policy for the Base Insurance:

Table 52 Create Policy

Actor	Relation Management
Object	Policy
Instance	new Policy
Event	Create Policy
Person	Mrs. Johnson
Product	Base Insurance
Policy Number	Policy-3a
Start Date	1 January 2006

Repeat these steps for John Johnson and Policy Number "Policy-3b".

16.4.3. Submit Claim

Create a claim first:

Table 53 Create Claim Consult Dentist

Actor	Member
Object	Claim
Instance	new Claim
Event	Create Claim
Policy	Policy-3a
Customer Reference	Dentist
CareProcedure	First Visit
Number	1
Treatment	0 (default)
Date	1 January 2006

And submit the claim:

Table 54 Submit Claim First Visit

Actor	Member
Object	Claim
Instance	Dentist
Event	Submit Claim

Because Mrs. Johnson is older than 18, the claim is rejected.
Repeat the steps in this paragraph for Policy-3b of John Johnson. Because he is under 18, his claim is accepted and completely covered.

16.5. Use case 4: Covered 100% up to maximum number (Nutritional Counselling)

An example of Nutritional Counselling is procedure 016/290161, “Nutritional Counselling”. Define this procedure as follows:

Table 55 Definition of Procedure Nutritional Counselling

Actor	Functional Management
Object	CareProcedure
Instance	new CareProcedure
Event	Create CareProcedure
Code	016/290161
Description	Nutritional Counselling
Price	46.40

16.5.1. Coverage

Nutritional Counselling is covered for maximum four hours per year. So define a CoverageMaximumNumber for Nutritional Counselling:

Table 56 Definition coverage Nutritional Counselling

Actor	Functional Management
Object	CoverageMaximumNumber
Instance	new CoverageMaximumNumber
Event	Create CoverageMaximumNumber
Product	Base Insurance
Name	Nutritional Counselling

Assign this care procedure to the coverage:

Table 57 Definition procedure group member Nutritional Counselling

Actor	Functional Management
Object	CoverageMaximumNumber
Instance	Nutritional Counselling
Event	Create CareProcedureGroupMember
CareProcedureGroupMember	new CareProcedureGroupMember
CareProcedure	Nutritional Counselling

And define the maximum covered number of units:

Table 58 Definition of Maximum Number of Nutritional Counselling

Actor	Functional Management
Object	CoverageMaximumNumber
Instance	Nutritional Counselling
Event	Change Maximum Number
Maximum number	4

16.5.2. Create Policy

Mrs. Johnson creates a policy for the Base Insurance

Table 59 Create Policy

Actor	Relation Management
Object	Policy
Instance	new Policy
Event	Create Policy
Person	Mrs. Johnson
Product	Base Insurance
Policy Number	Policy-4
Start Date	1 January 2006

16.5.3. Submit claim

Create the claim first:

Table 60 Create Claim Nutritional Counselling

Actor	Member
Object	Claim
Instance	new Claim
Event	Create Claim
Policy	Policy-4
Customer Reference	Nutritional Counselling
CareProcedure	Nutritional Counselling
Number	4
Treatment	0 (default)
Date	1 January 2006

And submit the claim:

Table 61 Submit claim Nutritional Counselling

Actor	Member
Object	Claim
Instance	Nutritional Counselling
Event	Submit Claim

This claim will be covered, because the limit is not reached yet. However a next claim will be rejected.

16.6. Use case 5: Cover with copayment (Inpatient Delivery)

An example of a care procedure with copayment is “041/190036”, “Inpatient Delivery without diagnosis”.
Define this procedure first:

Table 62 Definition procedure “Inpatient Delivery without diagnosis”

Actor	Functional Management
Object	CareProcedure
Instance	new CareProcedure
Event	Create CareProcedure
Code	041/190036
Description	Inpatient Delivery without diagnosis
Price	442.50

16.6.1. Coverage

A copayment of 261.50 is required by law. So define a CoverageCopayment object:

Table 63 Definition of Coverage Inpatient Delivery without diagnosis

Actor	Functional Management
Object	CoverageCoPayment
Instance	new CoverageCoPayment
Event	Create CoverageCoPayment
Product	Base Insurance
Name	Inpatient Delivery (no diagnosis)

Assign this care procedure to the coverage:

Table 64 Definition of Procedure Group Member Inpatient Delivery without diagnosis

Actor	Functional Management
Object	CoverageCoPayment
Instance	Inpatient Delivery (no indication)
Event	Create CareProcedureGroupMember
CareProcedureGroupMember	new CareProcedureGroupMember
CareProcedure	Inpatient Delivery without diagnosis

And define the copayment:

Table 65 Definition Copayment for Inpatient Delivery without diagnosis

Actor	Functional Management
Object	CoverageCoPayment
Instance	Inpatient Delivery (no indication)
Event	Create BenefitCoPayment
CoPayment Amount	261.50

16.6.2. Create Policy

Mrs. Johnson creates a policy for the Base Insurance.

Table 66 Create Policy

Actor	Relation Management
Object	Policy
Instance	new Policy
Event	Create Policy
Person	Mrs. Johnson
Product	Basic Insurance
Policy Number	Policy-6
Start Date	1 January 2006

16.6.3. Submit claim

Create the claim first:

Table 67 Create Claim Bevalling

Actor	Member
Object	Claim
Instance	new Claim
Event	Create Claim
Policy	Policy-6
Customer Reference	Delivery
CareProcedure	Inpatient Delivery without diagnosis
Number	1
Treatment	0 (default)
Date	1 January 2006

And submit the claim:

Table 68 Submit claim Bevalling

Actor	Member
Object	Claim
Instance	Delivery
Event	Submit Claim

Benefit amount for this claim is 442.50 – 261.50 = 181.00 euro.

16.7. Use case 6: Coverage of treatment (Physiotherapy)

An example is care procedure 02/2000, "Treatment Method Cesar". Define this procedure as follows:

Table 69 Definition of Procedure "Treatment method Cesar""

Actor	Functional Management
Object	Procedure
Instance	new Procedure
Event	Create Procedure
Code	02/2000
Description	Treatment method Cesar
Price	26.40

16.7.1. Coverage

Physiotherapy for chronic patients is covered from treatment ten onwards. So define a CoverageTreatment for Physiotherapy:

Table 70 Definition Coverage Physiotherapy

Actor	Functional Management
Object	CoverageTreatment
Instance	new CoverageTreatment
Event	Create CoverageTreatment
Product	Base Insurance
Name	Physiotherapy for chronic patients

Assign the procedure to the coverage:

Table 71 Definition Procedure Group Member Physiotherapy for chronic patients

Actor	Functional Management
Object	CoverageTreatment
Instance	Physiotherapy for chronic patients
Event	Create CareProcedureGroupMember
CareProcedureGroupMember	new CareProcedureGroupMember
Procedure	Treatment method Cesar

With treatment limits as follows:

Table 72 Definition treatment for Physiotherapy for chronic patients

Actor	Functional Management
Object	CoverageTreatment
Instance	Physiotherapy for chronic patients
Event	Change TreatmentLimit
Treatment From	10
Treatment To	0

16.7.2. Create Policy

Mrs. Johnson creates a policy for the Base Insurance:

Table 73 Create Policy

Actor	Relation Management
Object	Policy
Instance	new Policy
Event	Create Policy
Person	Mrs. Johnson
Product	
Policy Number	Policy-6
Start Date	1 January 2006

16.7.3. Submit Claim

Create the claim first.

Table 74 Create Claim Physiotherapy

Actor	Member
Object	Claim
Instance	new Claim
Event	Create Claim
Policy	Policy-6
Customer Reference	Cesar
Procedure	Treatment method Cesar
Number	1
Treatment	9
Date	1 January 2006

And submit the claim

Table 75 Submit claim Physiotherapy

Actor	Member
Object	Claim
Instance	Physio 9
Event	Submit Claim

This claim is rejected, because physiotherapy is only covered from treatment 10 onwards.

16.8. Use Case 7: Cover to Maximum Number of Units with Copayment (Maternity Care)

Maternity Care is covered with a maximum of 80 hours, and a copayment of 3.50 per hour. So use case 7 combines the use cases 4 and 5.
Define care procedure 06/196201, "Hour Maternity Care".

Table 76 Definition of Procedure Hour Maternity Care

Actor	Functional Management
Object	CareProcedure
Instance	new CareProcedure
Event	Create CareProcedure
Code	016/290161
Description	Hour Maternity Care
Price	37.90

16.8.1. Coverage

Define CoverageMaximumNumberCoPayment as follows:

Table 77 Definition Coverage Maternity Care

Actor	Functional Management
Object	CoverageMaximumNumberCoPayment
Instance	new CoverageMaximumNumberCoPayment
Event	Create CoverageMaximumNumberCoPayment
Product	Base Insurance
Name	Maternity Care

And assign the care procedure to the coverage:

Table 78 Definition Procedure Group Member Hour Maternity Care

Actor	Functional Management
Object	CoverageMaximumNumberCoPayment
Instance	Maternity Care
Event	Create CareProcedureGroupMember
CareProcedureGroupMember	new CareProcedureGroupMember
CareProcedure	Hour Maternity Care

Define a maximum number of units:

Table 79 Definition of maximum number for Maternity Care

Actor	Functional Management
Object	CoverageMaximumNumberCoPayment
Instance	Maternity Care
Event	Change MaximumNumber
Maximum Number	80

Define the copayment:

Table 80 Definition Copayment for Maternity Care

Actor	Functional Management
Object	CoverageMaximumNumberCoPayment
Instance	Maternity Care
Event	Change BenefitCoPayment
Maximum number	3.50

16.8.2. Create Policy

Mrs. Johnson creates a policy for the Base Insurance.

Table 81 Create Policy

Actor	Relation Management
Object	Policy
Instance	new Policy
Event	Create Policy
Person	Mrs. Johnson
Product	Base Insurance
Policy Number	Policy-7
Start Date	1 January 2006

16.8.3. Submit claim

Create the claim first:

Table 82 Create Claim Maternity Care

Actor	Member
Object	Claim
Instance	new Claim
Event	Create Claim
Policy	Policy-7
Customer Reference	Maternity Care
CareProcedure	Hour Maternity Care
Number	10
Treatment	0 (default)
Date	1 January 2006

And submit it:

Table 83 Submit claim "Maternity Care"

Actor	Member
Object	Claim
Instance	Maternity Care
Event	Submit Claim

This claim is covered, because the maximum number limit is not reached. The benefits is 10 * (37.90 – 3.50) = 344.00 euro.

16.9. Use case 8: Cover specific treatments (IVF)

The handling of use case 8 is comparable to use case 6.

16.10. Use case 9: Cover partly (Prostheses)

An example is care procedure 12/P25, “Lower Prosthesis”. Define this procedure as follows:

Table 84 Definition of procedure Lower Prosthesis

Actor	Functional Management
Object	CareProcedure
Instance	new CareProcedure
Event	Create CareProcedure
Code	12/P25
Description	Lower Prosthesis
Price	194.00

16.10.1. Coverage

Prostheses are 75% covered, so define CoveragePercentage as follows:

Table 85 Definition Coverage Prostheses

Actor	Functional Management
Object	CoveragePercentage
Instance	new CoveragePercentage
Event	Create CoveragePercentage
Product	Base Insurance
Name	Prostheses

Assign the care procedure to the coverage:

Table 86 Definition Procedure Group Member Lower Prosthesis

Actor	Functional Management
Object	CoveragePercentage
Instance	Prostheses
Event	Create CareProcedureGroupMember
CareProcedureGroupMember	new CareProcedureGroupMember
CareProcedure	Lower Prosthesis

And define the coverage percentage:

Table 87 Definition percentage for Prostheses

Actor	Functional Management
Object	CoveragePercentage
Instance	Prostheses
Event	Create BenefitPercentage
Percentage	75

16.10.2. Create Policy

Mrs. Johnson creates a policy for the Base Insurance:

Table 88 Create Policy

Actor	Relation Management
Object	Policy
Instance	new Policy
Event	Create Policy
Person	Mrs. Johnson
Product	Base Insurance
Policy Number	Policy-9
Start Date	1 January 2006

16.10.3. Submit claim

Table 89 Create Claim Lower Prosthesis

Actor	Member
Object	Claim
Instance	new Claim
Event	Create Claim
Policy	Policy-9
Customer Reference	Lower Prosthesis
CareProcedure	Lower Prosthesis
Number	1
Treatment	0 (default)
Date	1 January 2006

Table 90 Submit claim Lower Prosthesis

Actor	Member
Object	Claim
Instance	Lower Prosthesis
Event	Submit Claim

Benefit amount = 75% * 194.00 = 145.40 euro.