Research number: 659

Radboud University Nijmegen


Belastingdienst

# Master's thesis

Using Formal Methods within the *Belastingdienst*

August 2012

*Author:*
Xander Damen
0213845
x.damen@student.science.ru.nl

*Supervisors:*

| | | |
|---|---|---|
| Prof. dr. B. Dankbaar | b.dankbaar@fm.ru.nl | Radboud University Nijmegen, ISIS |
| Dr. ir. G.J. Tretmans | tretmans@cs.ru.nl | Radboud University Nijmegen, ICIS |
| Dr. D. N. Jansen | d.jansen@science.ru.nl | Radboud University Nijmegen, ICIS |
| H. Somers | h.somers@belastingdienst.nl | Belastingdienst/CA |
| J. van Rooyen | jos.van.rooyen@bartosz.nl | Belastingdienst/CA, Bartosz |

# Managementsamenvatting

De stijgende complexiteit van de bedrijfsprocessen en systemen van de Belastingdienst maakt handmatig testen tot een moeilijke taak. Daarom kijkt de organisatie naar het uitbreiden van haar skillset, omdat zij van mening is dat de huidige technieken verbeterd moeten worden om het hoge kwaliteitsniveau dat de organisatie nastreeft te handhaven. De Belastingdienst neemt formele methoden, wiskundige technieken, in overweging. Twee formele methoden zijn overwogen: *model checking* and *model based testing*. De Belastingdienst heeft een casus rondom het nieuwe Toeslagen systeem aangedragen. Vanwege de afwijkingen die het systeem vertoond ten opzichte van verwacht gedrag, is gekozen voor *model checking*. Dit heeft geleid dit de volgende hoofdvraag:

**Welke stappen zijn er benodigd voor een succesvolle implementatie van *model checking* in het ontwikkelproces van de Belastingdienst?**

Het onderzoek is opgedeeld in drie fasen, omdat er meerdere aspecten van de organisatie en *model checking* bekeken moesten worden om juiste aanbevelingen over het gebruik van formele methoden bij de *Belastingdienst* te kunnen doen. In de eerste fase wordt een beeld van de organisatie van de Belastingdienst gegeven. Daarnaast wordt een overzicht gegeven van de verankering van Toeslagen in de organisatie, en wordt achtergrond informatie van Toeslagen als product gegeven. Ook het ontwikkelproces van de Belastingdienst wordt besproken. Met sleutelfiguren uit het ontwikkelproces zijn interviews afgenomen. De informatie uit deze interviews wordt gebruikt om aan te tonen hoe *model checking* in het ontwikkelproces van de Belastingdienst toegepast kan worden. De resultaten van deze interviews zijn gekoppeld aan Kritische Succes Factoren uit de literatuur over succes en falen van IT projecten.

In de tweede fase is een *model checking* casus over het systeem van de Kinderopvangtoeslag keten van Toeslagen uitgevoerd. In de derde en laatste fase, is kennis opgedaan in de eerste en tweede fase gebruikt om aanbevelingen te doen over het gebruik van formele methoden in het ontwikkelproces van de Belastingdienst.

De interviews hebben aangetoond dat de kritische succes factoren uit de literatuur ook een belangrijke rol spelen in het project van het nieuwe Toeslagen systeem. Verschillende verbeterpunten zijn geïdentificeerd:

- Documentatie actueel houden.
- Werken in multidisciplinaire teams voor korte communicatielijnen.
- Lange leercurve van het nieuwe Toeslagen systeem.

De uitgevoerde casus heeft de volgende resultaten opgeleverd:

- De specificatie van gedrag is op punten onduidelijk en bevat onduidelijke zinnen. Deze zinnen geven de programmeur ruimte tot eigen interpretatie.
- Functies zijn gespecificeerd door te refereren aan het gedrag van andere functies. Dit kan tot fouten in deze functie leiden als het orginele gedrag wordt aangepast en deze functie niet meeveranderd in het product.
- Gebruik van ongedefinieerde termineer *events* en hetzelfde event wordt met meerdere benamingen aangeduid.
- Ongedefinieerd gedrag voor de service die het event `Cevt_tijdstip_beschikken` afhandelt, in het geval dat er geen concept beschikkingen aanwezig zijn.

Dit soort bevindingen kunnen de volgende impact op de organisatie hebben:

- Door onduidelijkheden in de specificatie moet er opnieuw naar de specificatie gekeken worden, wat inhoudt dat het gehele proces weer doorlopen moet worden.
- Als deze problemen hun weg naar de software vinden, moeten er patches uitgebracht worden.
- Problemen in de software kunnen burgers raken. De analyse van een bekend probleem heeft aangetoond dat dit potentieel 500,000 burgers kan raken.

Voor het gebruik van formele technieken, en met name model checking, in de organisatie, zijn de volgende stappen vastgesteld:

1. Werken in multidisciplinaire teams van domeinexperts, architecten en analysten
2. Architecten en analysten scholen in het gebruik van de formele taal en bijbehorende gereedschappen
3. Andere betrokkenen een basisscholing geven in de formele taal zodat zij de specificatie kunnen reviewen
4. Domeinexperts scholen in het opstellen van voorwaarden waaraan de specificatie en de software moet voldoen. Deze voorwaarden worden gecontroleerd door de *model checker*
5. Verificatietijd plannen, zodat de specificatie doorgerekend kan worden. Ook moet rekening gehouden worden met tijd om problemen die tijdens de verificatie naar voren komen, te herstellen.

Verder onderzoek is nodig om de winst die door *model checking* behaald wordt te kwantificeren, maar dit onderzoek heeft duidelijk gemaakt dat formele methoden van **toegevoegde waarde** zijn voor het ontwikkelproces. Zodra de leercurve genomen is, kan het systeem in een taal zonder ambiguïteit gespecificeerd worden. Dit levert een duidelijkere en verbeterde specificatie op ten opzichte van de huidige methode. Daarnaast biedt *model checking* mogelijkheden om de analyse van software fouten te ondersteunen.

# Management summary

The increasing complexity of the business processes and systems of the *Belastingdienst* make manual testing a difficult task. Therefore, the organisation is looking to expand their skill set, as they feel that current testing techniques need to be improved to achieve the high quality level the organisation pursues. The *Belastingdienst* is considering the use of formal methods: mathematically based techniques. Two formal methods were considered, model checking and model based testing. The *Belastingdienst* proposed a case study of the system of *Toeslagen*. Because of the nature of the anomalies the system of *Toeslagen* is suffering from, the selected formal method was model checking. This has led to the following question:

**What steps are required for a successful implementation of model checking within the development process of the *Belastingdienst*'s *Toeslagen* program?**

This research was divided into 3 phases, as several aspects of the organisation and model checking needed to be examined in order to give proper recommendations on the usage of formal methods by the *Belastingdienst*. In the first phase, a clear view of the organisation, the embedding of *Toeslagen* in the organisation and the development process used by the *Belastingdienst* is given. Also background information on the organisation and *Toeslagen* as a product are given. An empirical research on the development process is conducted through structured interviews with key personnel from the development process. Information gained from these interviews has helped to establish how model checking can be embedded in the development process of the *Belastingdienst*. The results of these interviews are structured with Critical Success Factors (CSF) found in a study of the literature on success and failure of information system projects.

During the second phase, a case study on the supporting system of the *Kinderopvangtoeslag* chain of *Toeslagen* has been performed. In the third and final phase, knowledge gained from both the first and second phase is used provide recommendations on the usage of formal methods in the development process of the *Belastingdienst*.

The interviews have shown that the critical success factors from the literature play an important role in the project for *Toeslagen* and several improvements have been identified:

- Keeping documentation up to date

- Working in multidisciplinairy teams to ensure easy communication
- The long learning curve of the new *Toeslagen* system

The case study has led to the following results:

- The specification is unclear and contains unclear sentences. These sentences leaves room for the programmer to give his or hers own interpretation
- Functions are specified referring to specification of other functions. This can lead to errors when the original function is changed
- Use of terminate events, where the terminate event is not defined and multiple names for identical events
- Undefined behavior for the service handling the `Cevt_tijdstip_beschikken` event, for the case that no concept depositions have been created

These findings can have the following impact on the business organisation:

- Due to ambiguity in the specification, rework has to take place. This means the entire process is performed once again
- If anomalies find their way into the software, patches must be made
- Anomalies in the software can have impact on citizens. Analysis of a Known Error has shown that it can potentially have impact on 500,000 citizens

Using formal methods, and mainly model checking, within the organisation requires the following steps to take place:

1. Working in multidisciplinairy teams with domain experts, architects en analysts
2. Educate architects and analysts on the usage of the formal language and accompanying tools
3. Educate others involved in the basics of the formal language, so they can review the specification
4. Educate domain experts in the scholen in the drawing up of conditions that the specification should uphold. These conditions are verified by the model checker
5. Schedule verification time, so the specification can be verified by the model checker. If problems are found during verification, their should be enough time scheduled to analyse and repair these problems

Further research is needed on quantification on the gain provided by model checking, it is clear that formal methods provide **added value** to the development process: once the long learning curve has passed, the system is specified in an unambiguous language, that can be verified by model checking. This improves the system specification. The model checking tool can also aid in the analysis of software errors from the production phase, speeding up the analysis.

# Preface

With this thesis I conclude my period as a student of computer science at the Radboud University Nijmegen.

First of all, I would like to thank Hans Somers. He has provided me with the opportunity to conduct the research within the Belastingdienst in the form of an internship. He has supported me during the entire research, even during periods where results were scarce and the internship needed to be extended.

In particular, I thank Jos van Rooyen of Bartosz for his continuous enthusiasm and extensive reviewing of this thesis. His time investment in this research has been exceptional. At times when I was struggling to find my way in the research, he gave me the proper insights which I appreciate enormously.

Furthermore, I could not have completed this research without the supervisors from the University: Ben Dankbaar, Jan Tretmans and David Jansen. Their advice and feedback has given this thesis a important quality injection.

The Radboud University has provided me with the opportunity to pursue personal growth via several extracurricular activities, for which I am thankful. It has helped me to develop myself on an extra level.

I would to thank everyone at the implementation team *Toeslagen* for the great time I had during my internship. I want to specifically acknowledge the support of the following: Jurgen van Amerongen, Dennis Geerlink, René Getkate and Tinus Zorgdrager, my roommates at the *Belastingdienst*, who spent many hours listening to me talking about my research, this thesis and many many other things. You have made my time at the Belastingdienst a very pleasant one.

Finally, I would like to thank my girlfriend, Sophie Verdonschot, for her love and support during the final years of my study. You have helped me to finally achieve my goals, most people believed to be impossible.

Xander Damen
Nijmegen, August 2012

# Contents

# Chapter 1

# Introduction

The Dutch Tax and Customs Administration (*Belastingdienst*) is looking for new techniques for testing [1] the software solutions in the organisation. The increasing complexity of the business processes and systems of the *Belastingdienst* make manual testing a difficult task. The organisation feels that current testing techniques need to be improved to achieve the high quality level the organisation pursues. The *Belastingdienst* is considering the use of formal methods: mathematically based techniques. The proposed formal method is Model Based Testing [1] (MBT). This introduction will look into the current issues in software testing and into the proposed and closely related techniques. Later in the introduction, the problem statement will be presented, as well as an approach to the case study and research.

Testing large, distributed software systems can be problematic: a decent test coverage is hard to achieve. Looking at Service Oriented Architecture (SOA, one of the software architectures used within the *Belastingdienst*), which is intrinsically distributed [14], using a test coverage of 80% per service in an environment using 3 services will result in a 50% ($80\% \times 80\% \times 80\%$) overall coverage [15]. Combined with the proposed method by the *Belastgindienst*, this raises the question how testing in such an environment can be extended or improved by the use of formal methods. It seems best to start at the beginning of the development process, as errors that emerge from the specification have the most impact [16]. When developing any software system, it is important to start with a clear, unambigious and error-free documented system specification and design. These design and specification documents play a crucial role in the software development proces and the maintenance of the system as they form the basis for the system. Problems within the initial (system) specification or the design of the system are often only noticed during the System or Acceptance test (which is performed after the realisation, see Figure 1.1) , or even later in the production phase. Errors in these specification documents are difficult and expensive to correct if propagated into the design or implementation phase [17, 18].

Errors in the information system can have a large impact on the organisation, as they are responsible for unsuccessful completion of the process. Have all choices been well thought

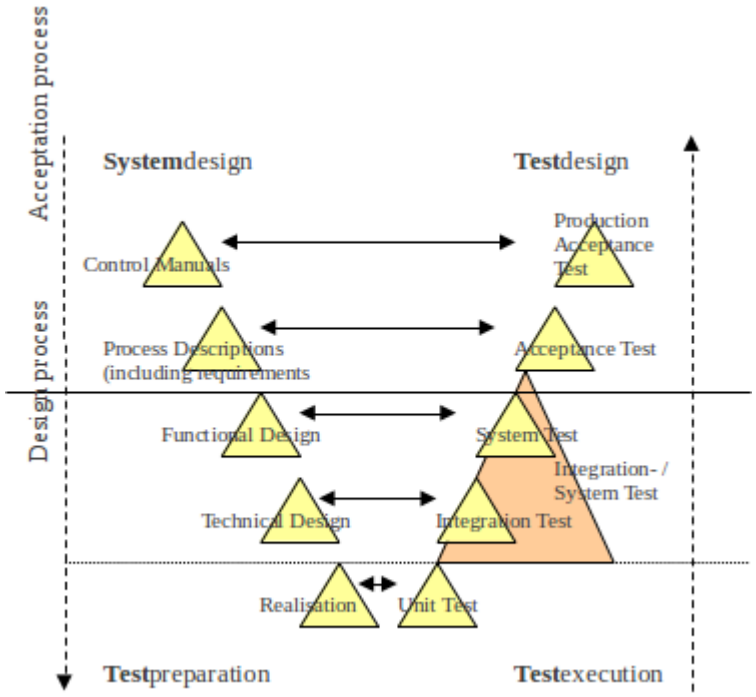


Figure 1.1: The life cycle development model [19]

through and are all possible situations covered by the specification? Current testing and review methods cannot keep pace with the construction of larger and more complex systems [20]. This asks for new technologies to meet these challenges, as is the case for the *Belastingdienst* as stated in [1]. Formal methods can aid the designers by formalizing the requirements before the system design begins. This is done by removing ambiguity from these specifications, as well as error detection, completeness of requirements [18] and verification of the implementation with regard to the specification, as the technique offers a complete view over the specification. However, these formal methods are mainly applied to reactive systems, typically embedded systems, because these systems provide a typical input/output response which is suitable for such formal methods: validating and verifying specification and implementation. Administrative systems typically do not have such input/output response defined. This makes them less suitable for formal methods. However, administrative systems can be built as reactive systems. The *Toeslagen* program of the *Belastingdienst*, which the *Belastingdienst* has proposed [1] as the system and specification to be researched, is built as a reactive system. Such behavior is created by using an Event-Driven Architecture, EDA. An EDA is a style of the earlier mentioned SOA [21,22]. However, as formal methods, and specifically model based approaches, have not yet been widely adopted in software or systems engineering [23,24], this field is yet to be explored, especially in administrative environments using EDA. Formal methods can be used to ensure that the provided specification of a business process or supporting

system is complete and error-free. Specification documents, especially early in the development life cycle (Figure 1.1) are written using natural language. This brings a certain level of ambiguity. Often designers try to take away this ambiguity by using visualization techniques. To be able to create a model of this specification, the specification should have as little ambiguity as possible.

As systems using an EDA are event-driven, and business processes are event-driven as well [25], there is a direct correlation between these business processes and systems. This implies that both the business processes and systems are specified in the development process. Faults within this specification will not only affect the software, but the entire process as the system is built to support the business process. When an error is not detected during a system or acceptance test, which is often only used to check the system and not the entire process, this can lead to process failure which affects the workforce and customer satisfaction [26].

Furthermore, the later an error is detected and changes need to be made to the system, the higher the cost of change is [16]. This is depicted in Figure 1.2. Formal methods can help to detect errors in design early, which leads to lower costs in the realisation of the system as well as less errors in production.

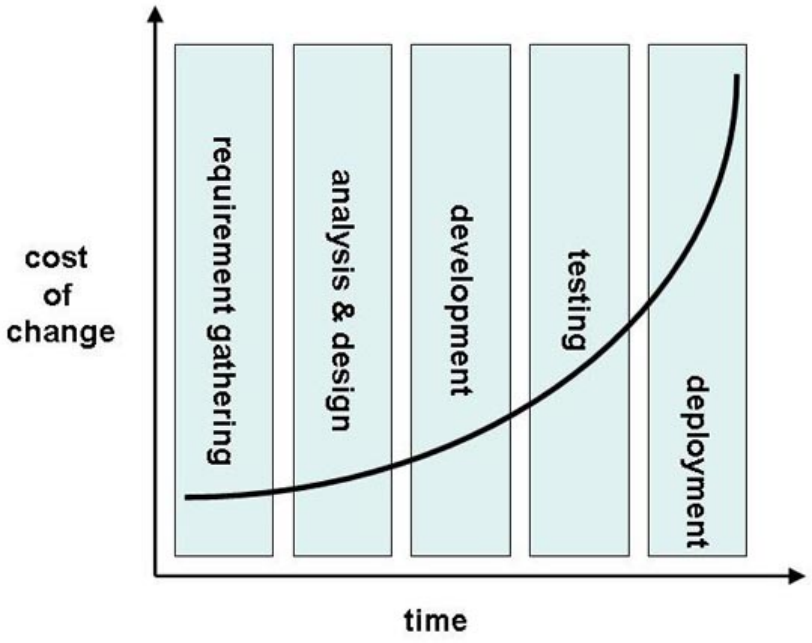


Figure 1.2: Cost of change curve [16, 27]

The *Belastingdienst* proposed MBT as the formal method to consider [1]. However, a technique called Model Checking (MC) is closely related to MBT, as both are formal methods that use modeling techniques for validation and verification. Both techniques

are considered and discussed briefly.

Model checking aims at showing that a model is valid and contains given properties. Model-based testing starts with an assumed valid model to show that the implementation under test behaves in compliance with this model [28]. As such, these two techniques are complementary. This is shown in figure 1.3.

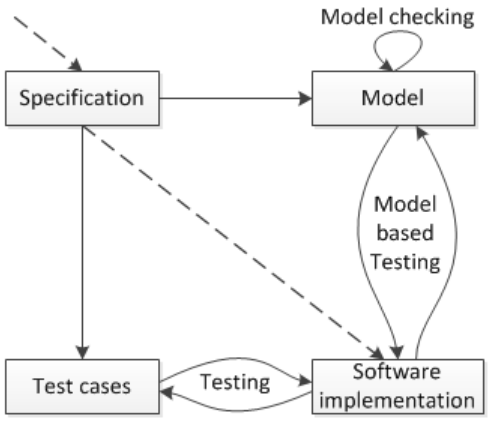


Figure 1.3: Testing, model checking and model based testing

Both techniques share several advantages: they are fast, exhaustive and can be performed automatically [29]. If errors are found, counter examples are provided [29]. In case of model checking, the counter example is an error path leading to a situation where the given property over the model did not hold. For model based testing, it consists of provided input, expected output and actual output.

One of the disadvantages of these formal methods is that they do not scale well to large systems [30,31]. The problem that occurs is called the " space explosion problem" [32,33]. This problem occurs when the complete set of instances of the system is too large and cannot fit into the memory of the computer system that is used to check the model. This means that no model is ideal to completely describe a complex or large system [34]. Therefore, among other things, abstractions to the system or selection of parts of the system have to be made [35].

Despite these disadvantages, model based techniques can be useful in the development of complex systems [36]. Overcoming these disadvantages is discussed in chapter 5.

## 1.1 Problem statement

The current system of the *Toeslagen* program of the *Belastingdienst* is still under development. The system is operational, but not all anomalies have been solved. Some

anomalies lead to the system being unable to make a new decision for the customer and unable to terminate successfully. It should be noted that this happens in rare cases, and most of the processing terminates succesfully. System architects and analysts have not yet been able to find the causes of these anomalies. System analysts believe that these anomalies occur in cases of high concurrency. They believe that concurrency is the cause, because the anomalies found in the production environment do not occur in the low concurrency test environment. In that test environment, the problematic cases lead to successful termination. The current application of testing techniques (based on TMAP) has been unable to find these errors.

In this high concurrency environment, identical services are connected to the enterprise service bus (a part of an EDA) to reach a higher throughput. It is important to note that in cases where no new decision is made, it is possible that the customer will receive a wrongful amount. These wrongful amounts are difficult to reclaim [37] and create an extra work load in the backoffice, as complaints will rise. More on this can be found in section 1.2.

As stated before, the *Belastingdienst* is interested in introducing new technology, a formal method for validation and verification, into the development process. However, it is unclear what the required characteristics and conditions for the application of these formal methods are [1]. It has therefore not yet been applied within the development process. In the literature is described that the introduction of a new technology into the development process is difficult if it requires fundamental change [38].

When applying modeling techniques, abstractions or a selection will have to be made to handle the state space explosion problem. Making the right abstraction and selection is difficult, as the model must be capable of describing system behavior and possible errors or problems of the system should not be abstracted from.

## 1.2 Case study

The *Belastingdienst* has provided the specifications of the *Toeslagen* program, which is the system under investigation in the case study. This case study should provide a basis for a general solution for the usage of formal methods within the *Belastingdienst*.

The program currently consists of 4 chains: *Zorgtoeslag*, *Huurtoeslag*, *Kindgebonden budget* (KGB) and *Kinderopvang Toeslag* (KOT) which operate in the EDA/SOA [21]) environment of the *Belastingdienst*. As it is not feasable to apply formal methods on all chains simultaneously, validate, verify and document findings within the time frame of the internship at the *Belastingdienst*, the scope has been narrowed. Therefore, only one of these chains has been selected.

There are several things to consider in selecting a chain: current state of implementation, impact of errors and known errors. The KOT chain is best suited, for several reasons. This chain has the lowest number of eligible "users", but the amount of money concerned

per "user" is highest. The system is known to have problems with rollback events (a rollback event is used to unset data set by previous event), as undefined situations can occur in those cases. In undefined situations the event-chain stops unexpectedly and no new decision regarding KOT is made when it in fact should have been made. No new decision means that the old decision continues and wrongful amounts are paid to an individual citizen.

Considering the given problems noted above and the concurrency issues mentioned in section 1.1, the formal method of choice is model checking. This is a proven technique in the analysis of concurrent programs [29, 39, 40]. Therefore, the main hypothesis for this case study is:

**Concurrency is the cause of the anomalies in KOT and model checking can detect these anomalies in the design**.

By creating a model of the initial system design and verifying properties over this model, errors in the design can be ruled out. Although architects and analysts believe that the current specification of the system is complete, it is crucial to verify the initial design. This verification is needed as the introduction of concurrency will raise the complexity due to liveness, fairness and deadlock properties. Therefore, introduction of concurrency will most likely raise the number of anomalies. By increasing the concurreny of the model, the errors the system currently experienced, and possibly other anomalies caused by concurrency, can be detected.

To be able to determine how to embed this model checking technique in the development process, the current development process must be mapped. This is done by examening internal documentation at the *Belastingdienst*, as well as by retrieving a list of people involved in the development process of the *Toeslagen* program and conducting an interview with a selection of people from this list. These interviews obtain experiences with the development process. This will give a good overview over the process, and recommendations on the process can be made. From these findings, recommendations on the embedding of model based approaches in the development process will be made. This will be based on findings from the case study, supported by literature.

## 1.3 Purpose

By means of the case study, the hypothesis stated in section 1.2 will be checked: "Concurrency is the cause of the anomalies in KOT and model checking can detect these anomalies". Although *Toeslagen* provides a basis for the study, findings will be generalised for a *Belastingdienst* wide usage.

Secondly, this study aims to get a clear view on the development process and its organisation within the *Belastingdienst*. This is done by examining documents from this process and conducting interviews with people involved.

Furthermore, this study will show the application of model checking to administrative

systems. In order to do so, a case study is performed. This will reveal possible anomalies (namely ambiguity, incompleteness and inconsistency [18]) in the specification of (part of) the supporting system and business processes. The case study also aims at detecting concurrency problems, following the hypothesis from section 1.2.

Fourthly, this study provides a set of guidelines on specification of business processes and systems to be suitable for model based approaches to testing. These guidelines come from the performed case study and literature.

Finally, this study will point out the prerequisites and changes needed for the adoption of model based techniques, mainly model checking, within the development process used by the *Belastingdienst*. This is done by linking findings from the conducted case study to the development process used in the organisation.

## 1.4 Questions

Because the *Belastingdienst* is looking for a general solution for the usage of formal methods, the case study provides the basis for this study. The hypothesis guides the research performed in the case study, while the main question guides the research in a broader view, presenting a broader view on the usage of formal methods. By means of several subquestions in support of this main question, the general solution the *Belastingdienst* is looking for will be investigated and presented.

The main question for this study is:

> What steps are required for a successful implementation of model checking within the development process of the *Belastingdienst*'s *Toeslagen* program?

This main question is supported by subquestions, based on [1]. Questions have been categorized in *Belastingdienst*, *Toeslagen*, KOT and model checking. Section 1.5 will provide the places in this thesis where the seperate questions are answered.

*Belastingdienst*

1. What is the organisational structure of the *Belastingdienst*?
2. What prerequisites and changes are needed in the development process of the *Belastingdienst* for a successful usage of model checking?

*Toeslagen*

3. What departments and units are involved in the *Toeslagen* program at the *Belastingdienst*?
4. Who was involved in the development process of the *Toeslagen* program, what role did they have and how have they experienced this development process?

KOT

5. What business processes are involved in KOT?
6. Where is the system for KOT described?

Model checking

7. What specification language and tool is best suited for the modeling and verification of KOT?
8. What level of abstraction is to be used for the modeling of the system supporting KOT?
9. What characteristics should the specification of business processes or systems have to be suitable for model checking?
10. What kind of errors does model checking detect?
11. To what extent does model checking improve the specification of the supporting systems?
12. What is the education level and knowledge needed for model checking?
13. What are the general usability, costs and time intensity for model checking within the *Toeslagen* program at the *Belastingdienst*?
14. Does model checking provide added value to an organisation, taking into account costs and benefits?
15. What view does model checking deliver of the supporting system of KOT?
16. Do stakeholders involved in KOT share the view delivered by model checking?

## 1.5 Method

This research is divided into 3 phases, as several aspects of the organisation and model checking need to be examined. The first phase, which will be described in chapters 2, 3 and 4, is meant to give a clear view of the organisation, the embedding of *Toeslagen* in the organisation and the development process which is used by the organisation. This

will answer questions 1, 3 and 5 of 1.4. Also background information on the organisation and *Toeslagen* as a product are given. An empirical research on the development process is conducted through structured interviews with key personnel from the development process. This answers question 4. The results of these interviews are structured with the help of Critical Success Factors (CSF) found in a study of the literature on success and failure of information system projects. Information gained from these interviews helps to establish how model checking can be embedded in the development process of the *Belastingdienst*.

The second phase, the information system of *Toeslagen* is modeled in a modeling language and verified with a model checker. Through this research with model checking, the hypothesis stated in section 1.1 will be tested. This is described in chapter 5. This research is expanded by analysing (Candidate) Known Errors (CKE and KE) using the built model. This analysis is explained in chapter 6. Both chapters 5 and 6 will answer the questions 6, 7, 8, 9, 10 and 11 stated in section 1.4. By performing an empirical research, experience of this model checking process can be used to give recommendations to the *Belastingdienst* on the embedding of this method in the development process of the organisation.

In the third and final phase, knowledge gained from both the first and second phase is used to answer questions 2, 15, 16, 13, 12 and 14 of 1.4. The combination of these questions will lead to a proposal of steps to take to use model checking within the *Belastingdienst*. This is presented in chapter 7. Before answering the main question of section 1.4 and delivering a final conclusion in chapter 9, chapter 8 will deliver a view of the related research and research areas to explore.

Please note that some complex figures are in Dutch. As the source images were not available, translations of these images could not be made in a timely manner.

## 1.6 Internship

This research is performed as part of an internship within the Central Office of the *Belastingdienst*. This internship took place from February 1, 2012 until August 31, 2012. The performed activities include modeling the system, conducting interviews, writing research proposal and this report. No *Belastingdienst* regular duties were performed during this internship.

# Chapter 2

# Organisation

As stated in the introduction, the research took place through an internship within the Central Office of the *Belastingdienst* (see 1.6). Before looking into the details of the development process of the *Belastingdienst*, its business processes and the IT systems involved in *Toeslagen*, an overview of the organisation is given. This overview provides a picture which will help to establish the position of the development proces, *Toeslagen* and its business processes within the organisation. Parts of the organisation with little to no involvement in *Toeslagen* are discussed briefly. Other, more involved parts of the organisation will be described in more detail. Within the figures shown in this chapter, units and departments marked red have involvement in one or more processes concerning *Toeslagen*.

## 2.1 Ministry of Finance

The *Belastingdienst* is part of the Dutch Ministry of Finance. The Ministry of Finance consists of four Directorate-Generals, of which the Directorate-General for the *Belastingdienst* (*DGBel*) is responsible for the *Belastingdienst*. This Directorate-General ensures, together with the *Belastingdienst*, that the national tax policy is implemented. An overview of the organisational structure of the Ministry of Finance is given in figure 2.1.

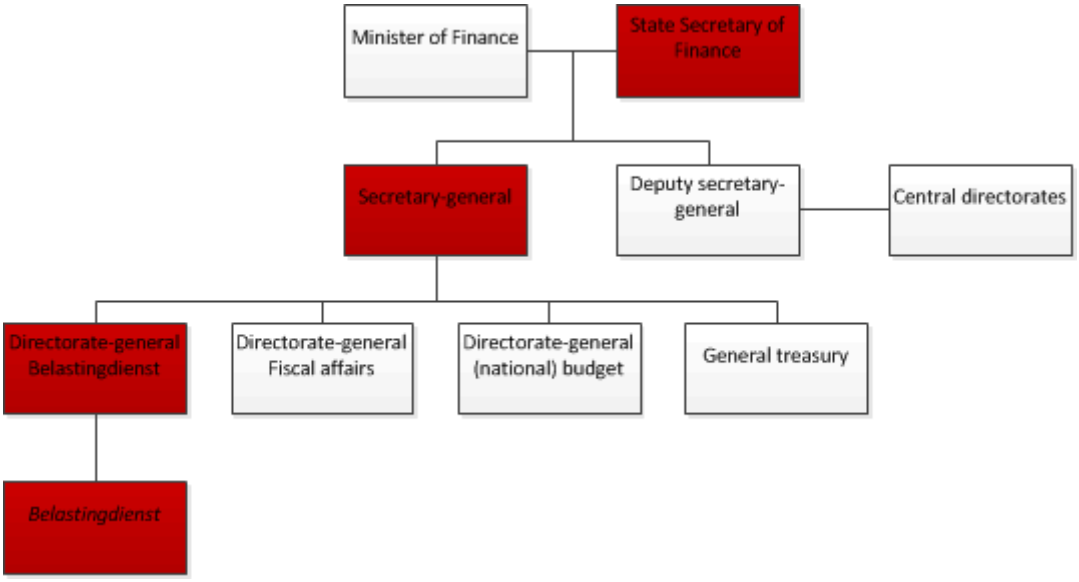


Figure 2.1: Organisational structure of the Ministry of Finance[1]

## 2.2 *Belastingdienst*

As mentioned before, the *Belastingdienst* is part of the Ministry of Finance. The more than 30,000 staff members of the *Belastingdienst* are responsible for a wide range of activities, but the *Belastingdienst* is best known for levying and collecting taxes and national insurance contributions. Each year, the *Belastingdienst* processes the tax returns of 10 million private individuals and 1.1 million entrepreneurs. The core duties of the *Belastingdienst* are listed below.

- levying and collecting taxes
- detecting fiscal, economic and financial fraud
- paying out income-related benefits for childcare, rent and health care
- supervising the import, export and transit of goods
- supervising compliance with tax laws and regulation

As can be seen in the core duties, the *Belastingdienst* not only collects, but also pays out. The *Belastingdienst* pays out provisional refunds and benefits (the earlier mentioned

[1]Information as displayed on the website of the Ministry of Finance. Current structure does not contain the Deputy secretary general.

*Toeslagen*) that are available to households towards the costs of childcare, rent or health care. More on *Toeslagen* can be found in chapter 3.

The *Belastingdienst* is structured in departments along these core duties (see figure 2.2):

- Customs
- *Toeslagen*
- Fiscal Information and Investigation Service
- Central Office
- Facility centers
- TaxLine
- National Office Tax Regions

The *Belastingdienst* considers it self-evident that its staff members are helpful and service oriented, and that they assume that taxpayers are acting in good faith.

The organisation applies three basic values [41]:

- credibility: the *Belastingdienst* takes its tasks seriously and stands by agreements
- responsibility: the *Belastingdienst* exercises its powers in a responsible manner and is prepared to account for its actions
- care: the *Belastingdienst* treats everyone with respect and takes everyone's expectations, rights and interests into account

These three basic values make it clear what the *Belastingdienst* stands for and what taxpayers may expect.

### 2.2.1 Fiscal Information and Investigation Service

When the *Belastingdienst* suspects fraud, the matter is referred to the Fiscal Information and Investigation Service (*Fiscale Inlichtingen- en OpsporingsDienst*, FIOD). The FIOD then assesses whether fraud is indeed being committed. If this is the case, the FIOD, in consultation with the Public Prosecution Service, may decide to start a criminal investigation.

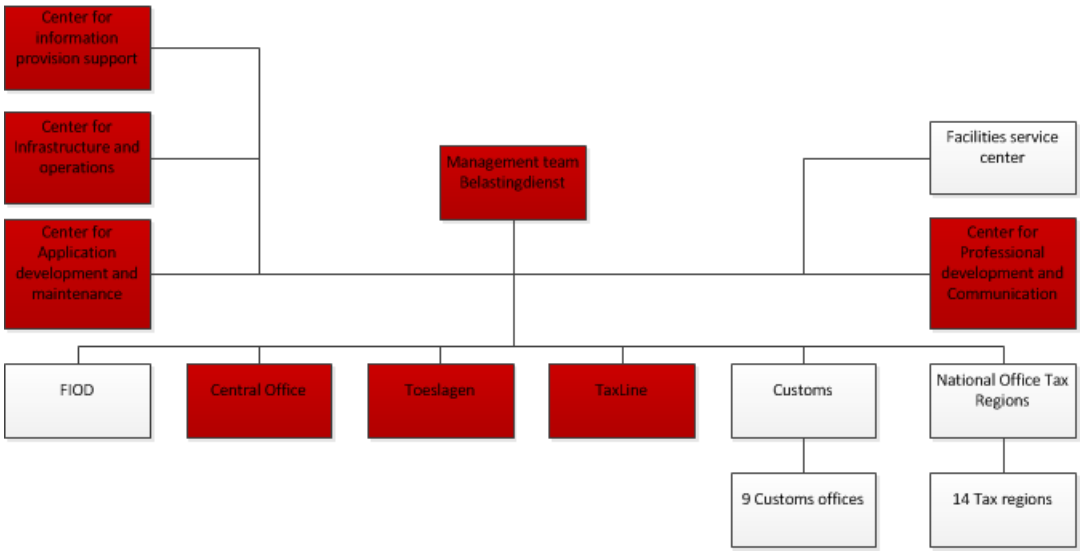


Figure 2.2: Organisational structure of the *Belastingdienst*

### 2.2.2 Central Office

The Central Office (*Centrale Administratie*, B/CA in short) is responsible for the content of the products and services to citizens and businesses. B/CA supports the other business units. Most orders and assessments originate from the B/CA. It is also data provider for the *Belastingdienst* and other public authorities. Other responsibilites include monetary transactions, levying and collecting taxes. More on the Central Office can be found in section 2.3.

### 2.2.3 Facility Centers

The *Belastingdienst* has five facility centers to ensure that all its duties are performed properly:

- Facilities Service Center (*Centrum voor Facilitaire Dienstverlening,* B/CFD)
- Center for Professional Development and Communication (*Centrum voor Kennis en Communicatie,* B/CKC)
- Center for Application Development and Maintenance (*Centrum voor Applicatie-ontwikkeling en onderhoud,* B/CAO)
- Center for Infrastructure and Operations (*Centrum voor Infrastructuur en Exploitatie,* B/CIE)

- Center for Information Service Support (*Centrum voor Ondersteuning IV-keten*, B/COI)

#### Facilities Service Center

Facilities Service Center is the internal service system of the *Belastingdienst*. Management of emergency and first-aid service, building management, distribution of postal items and archive management are examples of the duties performed by this center.

#### Center for Professional Development and Communication

B/CKC supports and advises the *Belastingdienst* in education and informing, communication and personnel and organisational development.

#### Center for Application Development and Maintenance

This center is the system integrator for the *Belastingdienst*, developer and administrator of ICT applications, sometimes in collaboration with external parties.

#### Center for Infrastructure and Operations

B/CIE provides data center services for the *Belastingdienst*, citizens and entrepreneurs.

#### Center for Information Service Support

B/COI supports the information services, offering methods, techniques, tools and regulations with regard to information services.

### 2.2.4 *Toeslagen*

*Belastingdienst/Toeslagen* is responsible for execution of the law on benefits. More on *Belastingdienst/Toeslagen* is found in section 2.4.

### 2.2.5 TaxLine

Taxline, or *BelastingTelefoon* is the unit of the *Belastingdienst* that private individuals and entrepreneurs can contact with questions about, for instance, tax returns, national insurance contributions and benefits.

### 2.2.6 Customs

Customs has 3 core tasks: stop goods at the border, control the proper application of laws and regulations and to levy and collect taxes.

### 2.2.7 National Office Tax Regions

The National Office Tax Regions (*Landelijk Kantoor Belastingregio's*) supports the Tax Regions to work in an unambiguous manner. The regions are specialised in individual supervision of taxpayers.

## 2.3 Central Office

The Central Office (Centrale administratie or B/CA) is responsible for the execution of the bulk and central part of the processes of the *Belastingdienst*. This involves administrative duties such as dispatching and processing various tax returns, dispatching notifications and bulk (supervisory) duties.

The B/CA supervises the automated handling of tax returns, remittances and payments. Three quarters of the total returns, remittances and payments are handled via automated systems. The supervision ensures that the various processes remain clear and manageable. Tax returns and notifications that cannot be handled automatically are also part of the duties of the B/CA. Furthermore the B/CA regulates the moments when the computer centre processes the data received from third parties.

For the bulk processes of the *Belastingdienst*, B/CA provides tailor-made services where possible. Speed, completion time, continuity and efficiency are key concepts in this respect.

B/CA combines all processes as regards the collection, registration and storage of data, the automatic processing of that data and its preparation for the notification process. This means that the client base consists of 12 million people, which covers a great deal of the population of the Netherlands.

### 2.3.1 Corporate identity

The B/CA is responsible for the content and delivery of products and services to citizens and entrepreneurs and supports the other business units of the *Belastingdienst*.

### 2.3.2 Mission

B/CA has responsibility over the handling of clients for *Toeslagen*, Tax Regions, Customs and TaxLine. This is carried out in support of the other business units and translated

to:

- Responsibility for the client to client (request to disposition) process for many citizens and entrepreneurs, for data-intensive processes that do not require extensive client handling and have a short turnaround time. This is a primairy process for the *Belastingdienst* with a shared responsibility with *Toeslagen*, Tax regions, Customs and TaxLine. This requires a view from citizen and entrepreneur perspective.
- Responsibility for collecting, processing, and timely provision of reliable information used within or outside the *Belastingdienst*. This requires thinking as the recipient of the data. These recipient are fellow business units, internal units or citizens.
- Responsibility for payment and collection. This is an important part of the government cash flow.
- Responsibility for corporate administration. This is an facilitating task for the *Belastingdienst*.

### 2.3.3 Vision

The core competencies of the B/CA are the fast, high quality delivery of products and services. Common characteristics of these products and services include massiveness, standardization and data-intensity. An increasing part of these products and services will be dealt with completely automated.

### 2.3.4 Structure

The Central Office has identified three core businesses: production, information and business administration. The B/CA is structured along these core businesses, as units are linked to a core business of the organisation. The business units are made up of one or more teams, each with their own specific area of expertise. For readability, these teams are not included in the organisational overview of the B/CA (see figure 2.3).


Figure 2.3: Organisational structure of the *Belastingdienst*/Central Office [2]

## 2.4 *Belastingdienst/Toeslagen*

*Belastingdienst/Toeslagen* (B/T) (see figure 2.4 for an overview of the structure of *Belastingdienst/Toeslagen*) is responsible for execution of the benefits acts. The mission of Belastingdienst/Toeslagen is to ensure that benefits are granted accurate, timely and lawfully and payed out in an efficient manner with minimal effort by the citizen.

This mission is translated into 4 core competencies [3]:

- Lawful granting of benefits
- Customer-oriented service

- Working efficiently
- Versatility

To perform its tasks, B/T collaborates with over 15 chain partners from inside and outside the *Belastingdienst* [3].

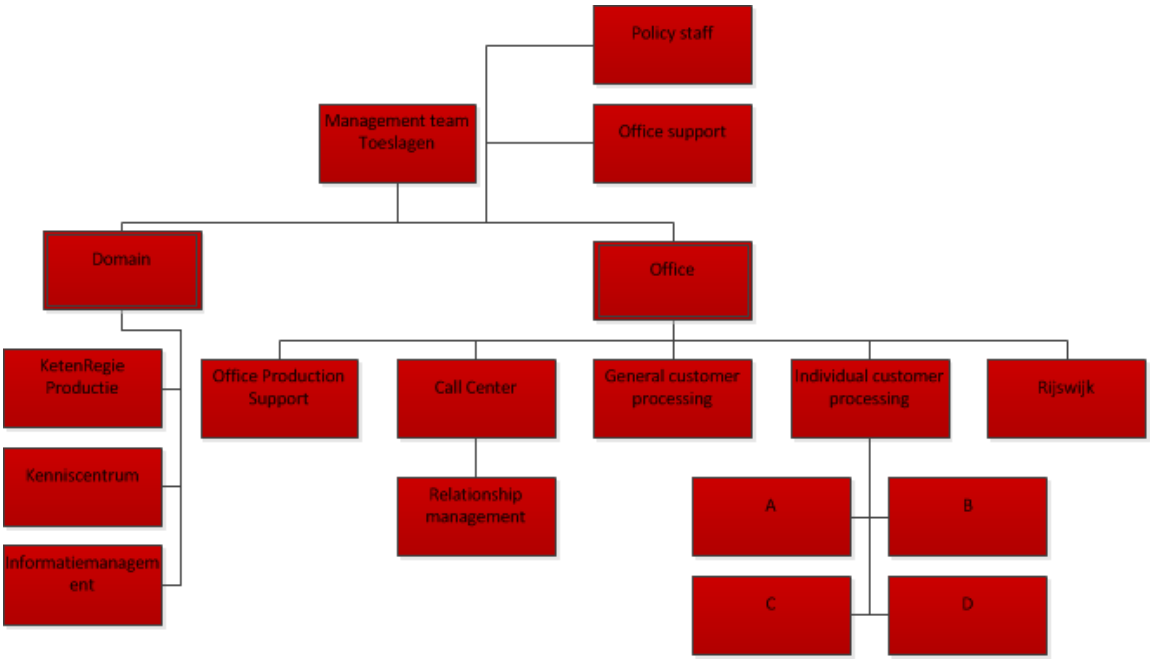


Figure 2.4: Organisational structure of the *Belastingdienst/Toeslagen*

As mentioned, *Belastingdienst/Toeslagen* is part of the executive body of government. While the *Belastingdienst* executes the policy of the Ministry of Finance, *Belastingdienst/Toeslagen* executes the legislation of benefits, which is created by other ministries. See chapter 3.

More on *Toeslagen* as a product, its supporting systems and business processes can be found in chapter 3.

### 2.4.1 Conclusion

This chapter has shown an overview of the organisational structure of the *Belastingdienst*, answering question 1 stated in section 1.4. As can be seen from this overview, many parts of the *Belastingdienst* have involvement in *Toeslagen*. The figures show which parts of the organisation exactly have envolvement (see question 3 in 1.4: what departments and units are involved in the *Toeslagen* program at the *Belastingdienst*).

# Chapter 3

# Toeslagen

In the Netherlands, citizens can be elegible for income-related benefits. These benefits find their basis in the General Benefits Act (*Algemene wet inkomensafhankelijke regelingen*, AWIR). *Belastingdienst/Toeslagen* is responsible for the execution of this law. The organisation and its structure have been discussed in 2.4. This chapter focusses on *Toeslagen* as a product and service and the processes and information services behind the delivery of *Toeslagen*.

As stated in chapter 1, there are currently four income-related benefits:

- Health care benefits (*Zorgtoeslag*), a compensation to the premium of health care insurance.
- Rent benefits (*Huurtoeslag*), intended for people on low incomes. With this benefit, people can afford to live in a rented accommodation.
- Child budget (*Kindgebonden budget*), a contribution to the living expenses of children.
- Childcare benefits (*Kinderopvangtoeslag*, KOT). Under the Childcare Act (*Wet kinderopvang*), the State, parents and employers together pay the cost of childcare.

These four benefits are established and organised by three different ministeries. This is done in five different acts: one for each benefit and in a General benefits Act (AWIR). The *Belastingdienst* is responsible for execution of these acts:

- The health care benefits fall under the authority of the Ministry of Health, Welfare and Sport (*Ministerie van Volksgezondheid, Welzijn en Sport*). These benedits are arranged in the Health care benefits Act (*Wet op de zorgtoeslag*).
- Rent benefits are defined in the Rent benefits Act (*Wet op de huurtoeslag*), of which the Ministry of the Interior and Kingdom Relations (*Ministerie van Binnenlandse Zaken en Koninkrijksrelaties*) is the legislating authority.

- The Ministry of Social Affairs and Employment (*Ministerie van Sociale Zaken en Werkgelegenheid*) is responsible for the Childcare Act (*Wet kinderopvang*). This ministery is also responsible for the Child budget, arranged in the Child budget Act (*Wet op het kindgebonden budget*) and the General benefits Act (*Algemene Wet Inkomensafhankelijke regelingen*, AWIR).

Benefits are related to the current situation of a citizen. Changes in the every day life of the citizen, such as changes in income, moving, marriage or death of a partner, can have impact on the benefits the citizen is entitled to. Most citizens, especially those receiving rent benefits and health care benefits, are highly dependent on these benefits, and will have problems covering their expenses if benefits do not arrive correctly and in time. It is therefore important to get the advance payment right and to work with current data. This requires an approach different from ex post calculation of due taxes.

The final income for a calendar year determines the actual benefits a citizen was entitled to. Therefore, an ex post calculation is part of the benefits proces, as final incomes are known a distinctive period after the calendar year. The difference between the advance payment by the *Belastingdienst* and the ex post determined final benefit will have to be settled. Before looking into the details of these benefits, first some facts on *Toeslagen*.

## 3.1 Facts and figures

In the Netherlands, 6.5 million households are eligible for one or more income related benefits. In total, over €12 billion is paid out each year. The division over the different benefits is shown in table 3.1. Note that the number of households does not add up. Households can be eligible for several benefits, these are only included once in the total number of households.

| **Benefit** | **Households*** | **Amount in €** |
|---|---|---|
| *Zorgtoeslag* | 5.600.000 | 5.333.200.000 |
| *Huurtoeslag* | 1.300.000 | 2.744.900.000 |
| *Kindergebonden budget* | 1.172.000 | 1.193.300.000 |
| *Kinderopvangtoeslag* | 539.000 | 3.178.800.000 |
| **Total** | **7.720.000**** | **12.450.200.000** |

* Monthly average

** Monthly average of households receiving one or more benefits

Table 3.1: Advance payments of benefits in 2011 [42]

### 3.1.1 Development

In 2007, the active information systems for *Toeslagen* were found to be inadequate for the job. It was too difficult to work with data depicting the current situation of the

citizen and the number of manual actions was too high. A new, event driven information system using a service oriented architecture was designed to reduce complexity and to put the citizen at the center of the organisation. The plan was named *Programma Toeslagen 2009* and its information system was named *Nieuwe Toeslagen Systeem*, NTS. This information system is made up of three parts: Facts Registration System (*Feiten Registratie Systeem*, FRS), Toeslagen (TSL), which forms the heart of the system, and the Office Portal (*Kantoorportaal*). A schematic overview of NTS is given in figure 3.1. This figure shows four different events: Bevent, Fevent, Gevent, and Hevent.

- Bevent is an event that contains a decision by the backoffice (*Beslis event*).
- Fevent is an event that contains facts about a citizen (*Feit event*).
- Gevent is an event that contains (*Grondslagen event*).
- Hevent is an event that requires a manual action by the backoffice (*Handmatig event*).

NTS is integrated in the existing IT architecture of the *Belastingdienst*. Section 3.3 will show which business processes have common ground with TSL. FRS and TSL are briefly discussing in sections 3.4.1 and 3.4.2 respectively.

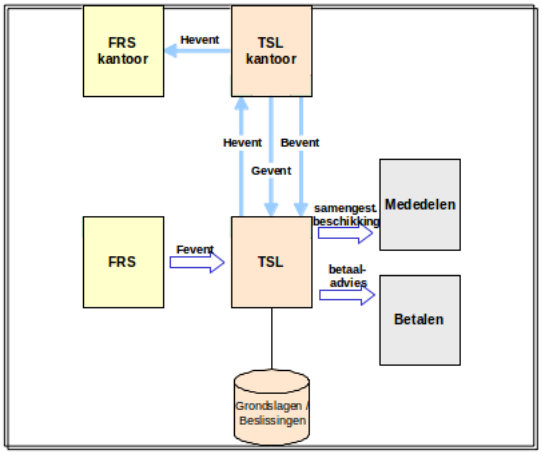


Figure 3.1: Global overview of NTS [4]

The development of this new information system NTS was estimated to take 2 years with total costs at €56 million. The planned date to start production in the organisation was set to November 2008, to calculate the benefits for 2009 (hence the name *Programma Toeslagen 2009*). This date was not met, and eventually the project was postponed three years for several reasons (see i.a. [43–50]). The project ran until July 1, 2012. Total costs have come to €238 million [51].

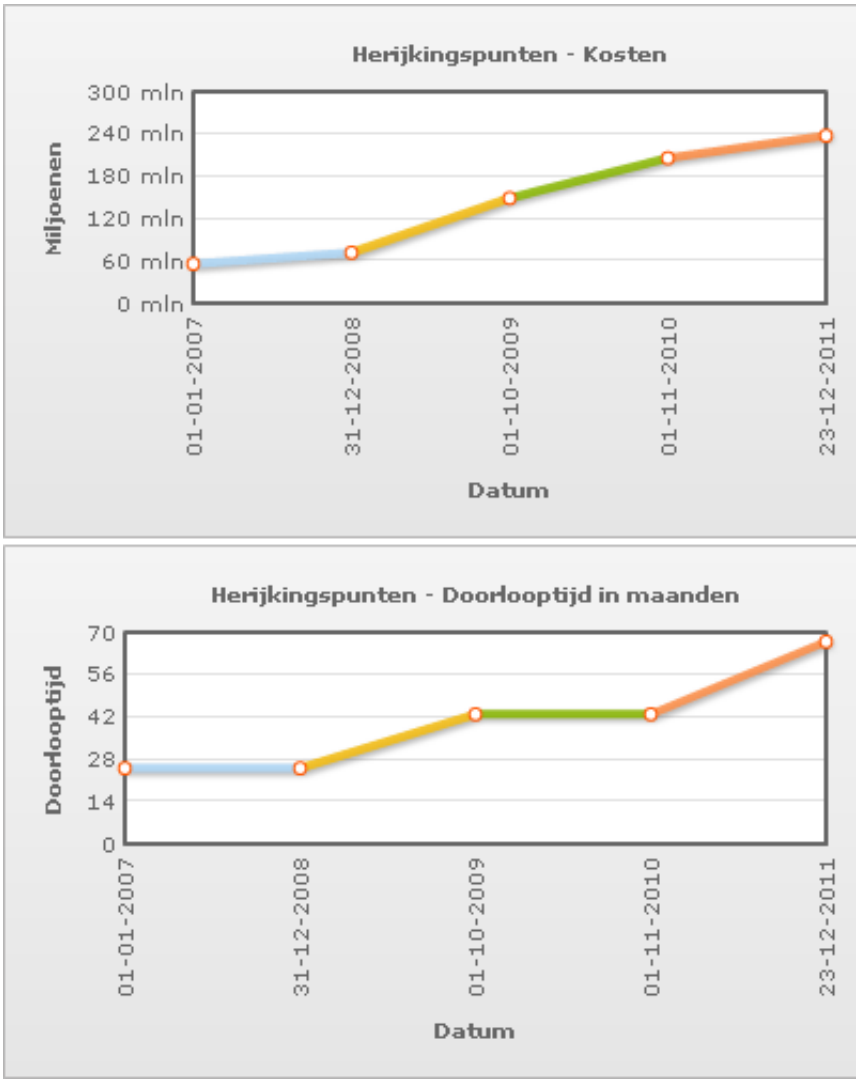


Figure 3.2: NTS time and costs [52]

Looking further into these costs, external personnel plays a large factor. In the beginning of 2011, 190 FTE (fulltime-equivalent) of external personnel per month was involved in the development and implementation of this NTS. Of this 190 FTE, 110 FTE was spent on software development and 80 FTE on system integration. Monthly costs for the development of NTS were €3.8 million each month [53].

As the case study (see 1.2) is focussed on a single benefit, KOT, some background information on this benefit is provided.

## 3.2 Kinderopvangtoeslag

Under the Childcare Act (*Wet kinderopvang*), the State, parents and employers together pay the cost of childcare. Childcare benefits (*Kinderopvangtoeslag*, KOT) are the State component, which is handled by the *Belastingdienst*.

There are several conditions to be fulfilled for a citizen to be entitled to childcare benefits:

- The citizen receives child benefit (*kinderbijslag*), foster parent contribution for the child or the citizen supports the child to a large extent
- The child is registered at the same address as the citizen
- The childcare center (*kindercentrum*) or host parent agency (*gastouderbureau*) is registered. For host parent agencies, the host parent must be registered as well
- A written agreement exists between citizen and childcare center or host parent agency
- The child is not enlisted in secondary education
- Citizen or partner have childcare expenses.
- Citizen and parent:
  - are of Dutch nationality or have a valid residence permit
  - have a job or study, or are following a reintegration program or citizenship course

The maximum income for child care benefits is higher than that of the other benefits and is based on the number of children for whom the benefits are received. For the first child, benefits are received for incomes until €117.000 per year. The benefit is however related to the level of income. The higher the income, the lower the benefit. Furthermore, the State has determined a maximum number of hours of child care each month and a maximum hourly rate.

## 3.3 Business processes

*Toeslagen* involves several business processes. By analysing these business processes, it will become clear how Toeslagen are handled in the organisation. Furthermore, these processes can help to establish which software components are used to calculate the benefits and are therefore within the scope of this research. Within the B/CA, these business processes have been registered and visualised (see [5]). These processes are general processes for each benefit:

- Processing notifications (*Verwerken meldingen*)
- Defaulters (*Wanbetalers*)
- Residence factor (*Woonlandfactor*)
- Automatic continuation (*Automatisch continueren*)
- Decision on Objection (*Beslissing op Bezwaar*)
- Appeal (*Beroep*)
- Mass supervision (*Massaal toezicht*)
- Final awarding (*Definitief toekennen*)

As the system was meant to put the citizen at the center and through notifications keep track of the current situation of the citizens, "Processing notifications" seems the most interesting business process to look at. This is confirmed by figure A.1 (see appendix). Two distinct parts can be distinguished from this process: Register facts (*Feiten registreren*) and settle benefits (*Afhandelen toeslagen*). Register facts occurs in FRS (See 3.4.1 for information on FRS). Figure A.1 clearly shows that a large part of the settle benefits process is embedded in TSL (see 3.4.2). This means that this TSL component is an important part of NTS: this is the component where the concurrency problems mentioned in chapter 1 occur. TSL is therefore the target of this research, this is were specific regulations for benefits are used. When looking more closely at figure A.1, several subprocesses can be identified:

- Workprocess receive notifications (*Werkproces Ontvangen meldingen*)
- Workprocess destack (*Werkproces Ontstapelen*)
- Workprocess process notifications (*Werkproces Verwerken meldingen*)
- Workprocess determine deviant handling (*Werkproces Bepalen Afwijkend behandelen*)
- Workprocess AWIR (*Werkproces* AWIR)
- Workprocess settle benefits regulations (*Werkproces Afhandelen Toeslagregeling*)
- Workprocess formal decision (*Werkproces Formeel beschikken*)
- Workprocess determine SVB (*Sociale Verzekeringsbank*, Social Insurance Bank) (*Werkproces Bepalen* SVB)
- Workprocess payment advice (*Werkproces Opstellen betaaladvies*)
- Workprocess composing content (*Werkproces Samenstellen Content*)

- Workprocess notify (*Werkproces Mededelen*)
- Workprocess stack (*Werkproces Stapelen*)
- Workprocess collect (*Werkproces Invorderen*)

Several workprocesses are embedded in the information system TSL, as can be seen in figure A.1. *Werkproces AWIR* and *Werkproces Afhandelen Toeslagregelingen* make up the automated part of the handling of notifications as they form the services AWIR and GBB (*Grondslagen, Beslissen en Beschikken* - Foundations, Decide and Disposition) of TSL (see figures 3.3 below and D.1 in appendix D). Figure 3.3 shows that *Bepalen* AWIR is part of the TSL component of NTS.

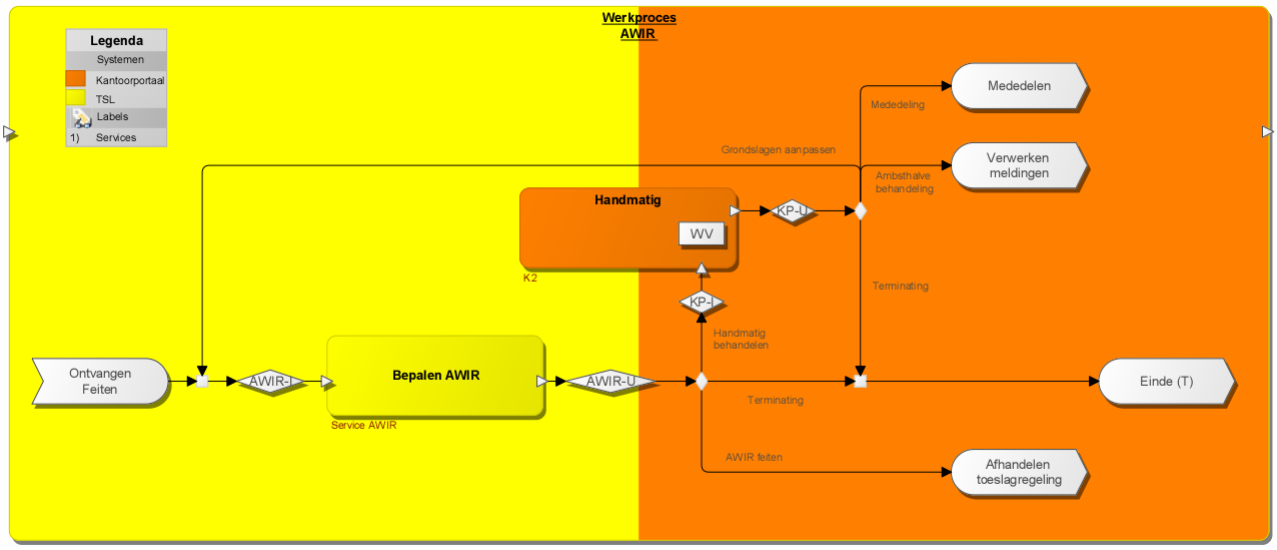


Figure 3.3: Workprocess AWIR [5]

### 3.3.1 Workprocess handling benefits regulations

Four business functions of the workprocess handling benefits regulations are embedded in TSL (see figure D.1). Together these business functions make up the GBB service of TSL as depicted in figures D.1 and 3.4.

These business functions are:

1. Recalculate expenses (*Herberekenen lasten*)
2. Determine household (*Vaststellen huishouden*)

3. Determine financial capacity (*Bepalen draagkracht*)

4. Decide on benefit regulation (*Beslissen toeslagregeling*)

Business functions 1-3 together form the foundations for *Toeslagen*.

## 3.4 ICT infrastructure

The Belastingdienst has a large ICT infrastructure. For *Toeslagen*, over 30 ICT components are involved in the delivery and processing of data. See figure B.1.

As stated before, NTS was to be embedded in the existing ICT infrastructure of the *Belastingdienst*, because the current systems needed to function as a data provider to the new system. While the new system is event driven, the existing systems are batch oriented. To link these two types of systems stacking and destacking mechanisms (see figure B.1, *Stapelaar / Ontstapelaar*) need to be placed. Work processes are in place for these actions, see 3.3, workprocess stack and destack.

Looking at the communication between the different services of NTS, this requires an Enterprise Service Bus [54] (ESB). The *Belastingdienst* has chosen to use Microsoft BizTalk.

### 3.4.1 FRS

As seen in figures 3.1 and A.1, the *Feiten Registratie Systeem* (FRS) registers all facts (*feiten*) concerning *Toeslagen* of which the *Belastingdienst* is notified by a citizen or third party. It is the first handler of messages after the receiving process. Because it is the first handler of messages, FRS also checks the incoming data for inconsistencies or missing values. If those checks are passed, the new situation is stored. Based on the new situation, zero to several events are placed on the ESB to be handled by TSL.

### 3.4.2 TSL

As mentioned earlier, TSL is made up out of several business functions. These business functions are components of the "Workprocess settle benefits regulations", part of het business process processing notifications. The position of these business functions in TSL is displayed in figure 3.4. This picture clearly shows that the different benefits are separated within TSL and that AWIR functions are shared throughout all benefits.

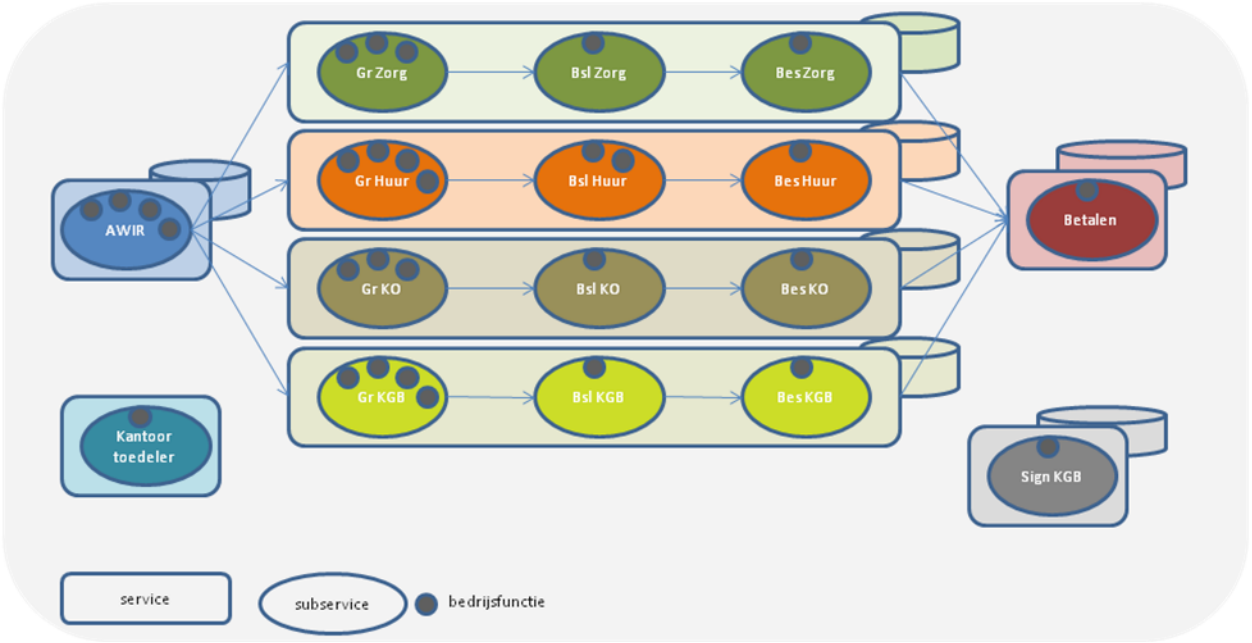


Figure 3.4: TSL

## 3.5 Conclusions

This chapter focussed on the product *Toeslagen*. *Toeslagen* involves several business processes, which are identical for each benefit. The information system components for the processes are used to distinguish between the different regulations for benefits. The leading question for this chapter was question 5 of 1.4: what business processes are involved in KOT? This question was answered in 3.3, showing the relevant processes: processing notifications, defaulters, residence factor, automatic continuation, decision on objection, appeal, mass supervision and final awarding. The most relevant business process was also identified: Processing notifications (*Verwerken meldingen*).

# Chapter 4

# Development process

As mentioned in the introduction, the research focuses on the usage of formal methods, mainly model checking, in the development process of the *Belastingdienst*. This embedding of model checking in the software development life cycle (SDLC) of the *Belastingdienst*, requires an overview of the SDLC. To ensure that formal methods provide added value to this development process through optimisation of this process, experiences of key persons in this development process have been collected. This was done by conducting interviews with these key persons. These key persons are positioned throughout the development process, giving a full coverage over the development process at the *Belastingdienst*. These interviews, combined with experiences from the case study (see chapter 5) will form the basis for the recommendations in chapter 7.

Before discussing these interviews, an overview of the process is given. The results from these interviews are structured along Critical Success Factors (CSF), identified through a study of the literature. This is presented in section 4.3.

## 4.1 Development process

The development process of the *Belastingdienst* is extensively described in [6]. The organisation uses a layered V-model. It is depicted in figure 4.1. A clearer view of the process, showing the layered V-model is presented in figure E.1. Figure E.2 also depicts the process. Each step from this development process is described briefly.

### Impulse

Impulse (*Impuls*) is a change request or change proposal. An impulse can originate from for example new legislation, the performing organisation, market, partners in the development process or the unit information management. As the origin of an impulse is very wide, so are the goals. From improvement of system or process to totally new

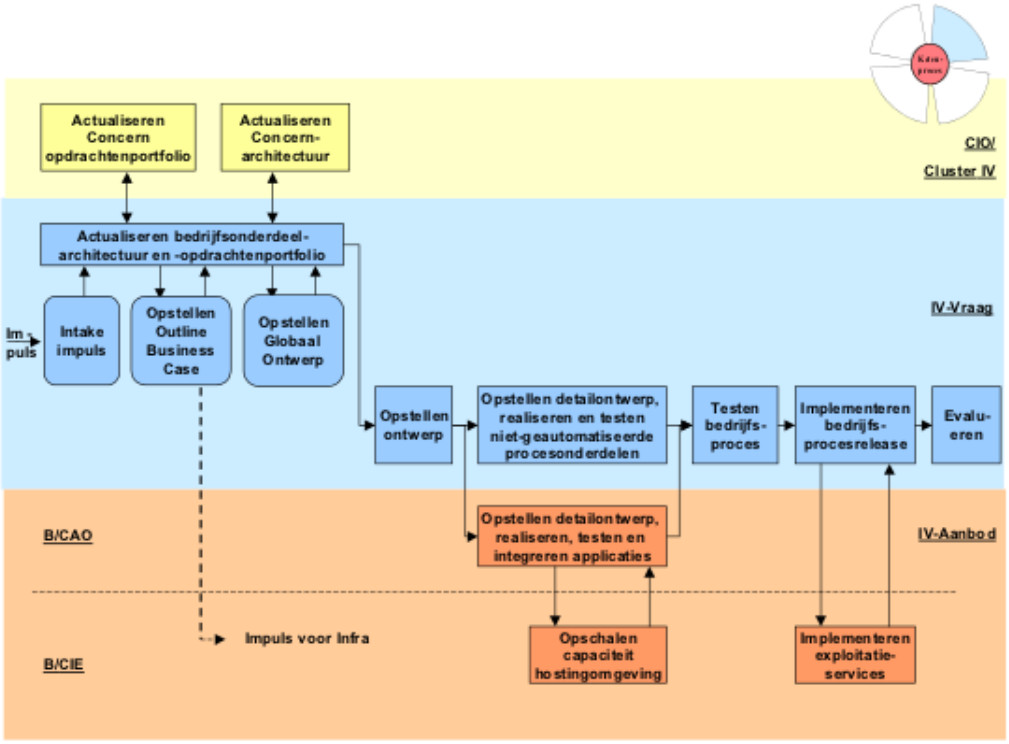


Figure 4.1: Development process [6]

systems all have an impulse as a starting point in the development process. Involved business units are consulted on the availability of capacity.

### Intake impulse

Intake impulse (*Intake impuls*) registers the change request or proposal and an impact analysis is performed. Goal of the registration is to determine the deadline for impact analysis. The impact analysis is performed to give all stakeholders a global assessment of the full extent of the request. Both tasks are performed by Information Management (IM).

### Drafting outline business case

After the impact analysis of the intake, an outline business case is drafted (*Opstellen outline business case*). This intends to determine if it is desirable to invest in the proposed change. Performed by Information Management, B/CIE and B/CAO.

### Drafting global design

If the outline business case leads to an order global design (*Opdracht globaal ontwerp*), this is performed by IM and B/CAO. It results in a global design, detailed business case and Product Risk Analysis (PRA). In case of a project, a Project Initiation Document (PID) is drafted. Phase plan, control plan, end stage report and highlight report are delivered as well in case of a project.

### Update business architecture components and task portfolio

In "Update business architecture components and task portfolio" (*Actualiseren Bedrijfsonderdelen Architectuur en -opdrachtenportfolio*) changes from intake impulse, outline business case and global design are put into the architecture. After changes have been made, a business process release is composed. These tasks are performed by IM and B/CAO.

### Update corporate architecture and corporate task portfolio

The subprocess "Update corporate architecture and corporate task portfolio" (*Actualiseren concern architectuur en concern opdrachtenportfolio*) tests if the proposed changes to the business architecture fit within the corporate architecture. The architecture board and the portfolio board of the *Belastingdienst* is responsible.

### Draft design

Changes to the process, business process release, actualising the design of the business process and determining the functional and non-functional requirements of this business process make up the Draft design (*Opstellen design*) phase in the V-model. Performed by: IM, B/CIE and B/CAO.

### Draft detailed design, realise and test automated information services

Draft detailed design, realise and test automated information services (*Opstellen detailontwerp, realiseren en testen van* ICT-*services*) takes place within or under the authority of B/CAO. The ICT start architecture is the basis to which details are added. These details in the design are necessary to start building and testing the software. B/CIE also plays a role in this phase.

### Upscaling hosting capacity

B/CIE and B/CFD are responsible for "Upscaling hosting capacity" (*Opschalen capaciteit hostingomgeving*). Despite the name of the process, downscaling the hosting capacity is

also considered during this phase.

### Draft detailed design, realise and test non automated information process components

All non automated information components for the business process are designed in the "Draft detailed design, realise and test non-automated information process components" (*Opstellen detailontwerp, realiseren en testen niet-geautomatiseerde procesonderdelen*) phase. Forms, letters, education are examples of these components. B/CKC, B/CIE, B/CFD and all units needed for extra expertise or domain knowledge take part in this phase.

### Test business process

Test business process (*Testen bedrijfsproces*) tests the ICT, non-ICT and hosting aspects of the business process integrally. Performed by B/CAO, together with B/CIE and involved business units.

### Implement operational services

After the testing of the business process and a lifting of an embargo by IM, the product can be placed (implemented) on the production environment (*Implementeren exploitatieservices*) by B/CIE.

### Implement business process release

In the Implement business process release (*Implementeren bedrijfsprocesrelease*) phase, the business process release is made available to the production crew. Handled by B/CAO, B/CIE, IM and other delivery parties of non-ICT process components.

### Evaluate

Through the results of the process and the business case the development is evaluated (*Evalueren*). This is done by IM. Lessons learned are reported to management teams of involved business units, to be recorded in the management cycle.

Next to this development process, several boards have been initiated to share knowledge, improve communication between the information management of the different units and assist the management teams in their decision making process:

- Information service board (*IV-overleg*), ISB
  - Provides a frame for Information Service partners to uphold
  - Advice on *Belastingdienst* portfolio
  - Alignment Information Services and business goals
  - Mitigate risks
- *Belastingdienst* portfolio board (*Concernportfolioboard*), BD PB
  - Optimises allocation of resources for information services on business level
  - Optimises information service portfolio
  - Advices about minimising risks for business continuity
  - Creates option scenarios for *Belastingdienst* management team
- Architecture board *Belastingdienst* (*Architectuurboard Belastingdienst*), AB BD
  - Advices on architectural products
  - Performs architectural control on global designs
  - Focusses on professional decision making for the integral business architecture
  - Commits participants to the given advices
  - Mitigate risks
- Unit portfolioboard (*Bedrijfsonderdeelportfolioboard*), Unit PB
  - Decides on unit task portfolio and balances this portfolio on unit level
  - Optimises allocation of resources for information services on unit level
  - Mitigate risks
  - Commits participants to decisions
- Unit architecture board (*Bedrijfsonderdeelarchitectuurboard*), Unit AB
  - Guards consistency and quality of designs and processes
  - Focusses on rational decision making for unit architecture
  - Commits participants to decisions
  - Mitigate risks
  - Is a decision making body considering the coherence of information services on unit level

To sum up, the processes and tasks are mapped in a responsibility assignment matrix (or RASCI-table: Responsible, Accountable, Support, Consulted, Informed) for an overview of the development process within the *Belastingdienst*. This matrix is shown in table F.1 in appendix F.

Some processes indicate a shared accountibility or responsibility shared with a consultancy role in table F.1. This is caused by the different roles these units play in the subprocesses of these processes.

The development process can be mapped to phases from the PRINCE2 project management methodology, which is frequently used for projects within public sector entities [55], and is the method used for *Programma Toeslagen 2009*. For this mapping, see figure 4.2.

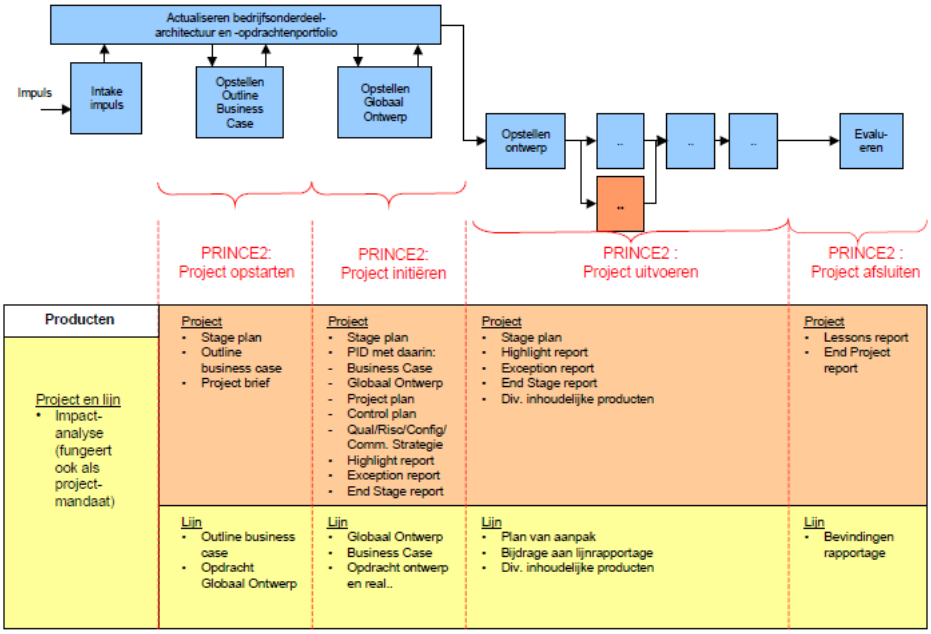


Figure 4.2: Development process and PRINCE2 phases [6]

## 4.2 Critical Success Factors

To get a better understanding of the development process besides the formal description of this process, key persons from this process were interviewed. Besides improving the understanding of the development process, the goal of these interviews was to identify experiences with this development process. The selected method of interviewing was a structured interview. This is presented in appendix H. In order to be able to structure the results of these interviews, a study of the literature was conducted on the Critical Success Factors (CSF) for ICT projects. This study of the literature mainly focussed on CSF for public sector entities, but some more general studies were also found to be useful.

From the 1960s project management researchers have been trying to discover which factors lead to project success (see for example [56]). This has lead to the term Critical Success Factors [57]. A critical success factor is that factor which must receive ongoing attention from management. Over the last few decades, IT projects have received

attention for those critical success factors (for example [58–60]). However, most managers focus on the control aspects of their project management method. This method is often, as stated earlier, the PRINCE2 methodology. This method uses management aspects time, cost, quality and scope for steering. These factors also determine project success: it has the previously agreed functionality (quality and scope), it is delivered on time (time) and within the agreed budget (cost). However, from literature, several other factors come forward that are critical to the success of an information system project, as steering on time, cost, quality and scope itself is difficult. These CSF are listed below:

- top management support and involvement [55, 61–64]
- planning [61, 62, 64, 65]
- communication [55, 62, 64, 66, 67]
- staff (number, skills, involvement) [61–64, 67]
- project mission (business case, goals) [55, 62, 64, 66]

While customer involvement is a critical success factor that is found in literature, this is not part of the list. As the *Belastingdienst* is a tax and customs organisation, it does not involve its "customers", the taxpayer in the development process.

As stated in chapter 3, the *Belastingdienst* aims to put the citizen at the center of its organisation. This is a process of becoming customer oriented. Figure 4.3 shows a fishbone model that specifies the steps that are necessary [68] to drive the process of becoming customer oriented:

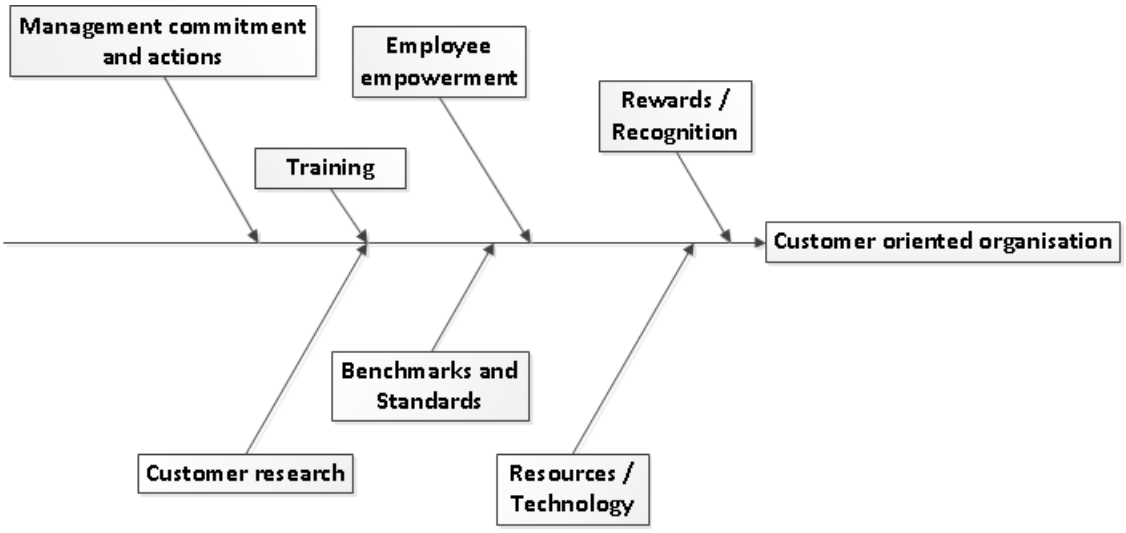


Figure 4.3: The process of becoming customer oriented [68]

The aspects from the fishbone closely resemble the CSFs. Management commitment and actions are similar to top management support and involvement. Employee empowerment, training and rewards / recognition are aspects related to staff. Benchmark and Standard, Resources / Technology and Customer research are all aspects that are included in the project mission of the NTS project. So there is a close relationship between the earlier mentioned CSF and the steps from the transformation process of becoming customer oriented.

The questions shown in appendix H have been created with formal methods, the development process and these CSF in mind. Questions have been grouped in several categories: personal information, development process, quality, testing, impact and changes to the process.

## 4.3 Results

In the following subsections (4.3.1-4.3.6), results from the twenty conducted interviews are discussed. References to respondents are formatted as [R0X], where X ranges from 01 to 20. Each CSF is discussed seperately, discussing the overall experiences of this CSF by the respondents. Other interesting experiences, that are not related to one of the CSF are listing in section 4.3.6. The distribution of the respondents over the development process is shown in table G.1 in appendix G. Translation of the function names is given in table G.2.

### 4.3.1 Top management support and involvement

Experience has taught that empowering an Executive Committee (*Dagelijks bestuur*) and a Program Board has significantly improved the governance of the NTS project [R001, R003, R005, R011, R015]. The direct involvement of the top management is experienced as very positive and was one of the key factors for the successful implementation of NTS.

With the chosen governance structure, it was ensured that the resulting products of NTS met the expectations of the workforce.

### 4.3.2 Planning

The scope of the initial PROGRAMMA TOESLAGEN 2009 program was initially small. The scope of the current program NTS is much larger. This gradual expanding of the scope in combination with the program target for 2009 has led to extra time pressure ([R011, R015]) and costs in the beginning of the project. Later, the program was renamed to NTS, removing the year from the name. This enabled the organisation to better manage the expectations of the project.

The experience shows that due to time pressure, new, chosen, methods are left and the old, known methods are reused ([R014]).

While a respondent [R015] stated that it has been difficult to explain to the contractor (CapGemini) that the error level should be as low as possible to be acceptable in production, timeboxing has lead to the concept to accept a certain level of errors ([R010, R019]). These errors are handled in production phase via workarounds [R005] and resolved in future releases.

### 4.3.3 Communication

Most respondents agree that communication is a point of attention. Issues mentioned include structure of organisation and project, development issues, customer-contractor relation, the workforce and documentation.

Several respondents have experienced that due to the large nature of the project, it had to be split into several subprojects. This has led to an increased intensity in communication ([R001, R002, R005, R009]).

The Belastingdienst has several basic concepts (Methods, Techniques, Tools and Instructions - *Methoden, Technieken, Hulpmiddelen en Voorschriften*, MTHV) for usage in their development process (see section 4.1). These concepts are constantly optimised and brought into the organisation. Units have different speed in adoptation of these improved concepts, which can lead to different approaches being used by different units during the same time frame ([R001, R002, R005]). This requires extra communication between the units.

Experience of several respondents shows that multidisciplinairy teams help to improve the communication (as the team contains the needed knowledge) and approaches. This approach helps to gear the activities to one another and improves the end result, which better fits the expectations.

### 4.3.4 Staff

The learning curve of a SOA/EDA system has been longer than originally anticipated. External knowledge was brought into the organisation to help the organisation use such a system. However, as external personnel will leave at a defined time, regular personnel must be able to continue the usage of the system. Intensive knowledge transfer has been organized and is still going on at this time ([R001, R007]).

Great concern is the dependency on a few people, a concern that is shared by most respondents. This dependency is mainly on architects, designers and builders. Currently, this knowledge is being transferred to the organisation.

### 4.3.5 Project mission

Project NTS was started to create a system that contains data that reflects the current situation of the citizen. In first instance the main focus of the project was on the ICT

components needed for this change in data focus ([R004, R011, R013]). Later in the project, the translation to the organisation was made ([R013]).

The initial request for the system contained little requirements with regard to the office portal. A respondent [R014] states that this portal was therefore built by the contractor without an initial design provided by the customer. This supply-driven approach has led to several change requests regarding this portal.

The chain of KOT is very wide, a lot of units have involvement in the chain. This creates a lot of different requirements for building and testing the system ([R001, R005, R006]), making it more difficult to see what are the most important aspects.

### 4.3.6 Miscellaneous

Testing is an aspect for which many respondents see possible improvements. Tests regarding functional details and non-functionals can be expanded [R001, R003, R007, R008]. This expansion might require other techniques to be added to the toolkit of the *Belastingdienst* to cover these issues.

Finally, the documentation of the system is not written down in a central place. This can be improved by updating the document that should serve as the test basis, the TSL Service Document, more regularly ([R009, R010, R016, R019]). This will move the needed information from the seperate places to a main document. This information is currently found in change requests, emails, memo's or is tranferred verbally. It requires an active attitude and depends on the presence of key persons to obtain the relevant information.

## 4.4 Conclusion

This chapter focussed on the development process of the Belastingdienst. This was guided by question 4 of section 1.4: Who was involved in the development process of the *Toeslagen* program, what role did they have and how have they experienced this development process? Via a study of the literature, Critical Success Factors (CSF) were identified to be able to structure the results from the interviews. The results from the interviews indicate that the critical success factors from the literature are also relevant for the NTS project. Several improvements for the project have been identified. Roles of respondents have been summed up in table G.1 in appendix G. Keeping documentation up to date, multidisciplinairy teams to ensure easy communication and the learning curve of the new system for employees of the *Belastingdienst* are important factors mentioned.

# Chapter 5

# Modeling *Toeslagen*

This chapter focusses on the activities performed during the case study, as described in section 1.2. The main goal of the case study was to test the hypothesis: "Concurrency is the cause of the anomalies in KOT and model checking can detect these anomalies". Before these anomalies can be detected with model checking, a model of the system has to be created. This requires a description of the system. The following section will describe the process of selecting the proper documents following the development process described in chapter 4. Then, the choice for SPIN is explained. The next section, 5.3, describes the model created in PROMELA, along with the abstractions made. The validation process of the model is described in section 5.6. Verification approaches and results are listed in section 5.7 and 5.8. These results are translated to reflect the business impact.

## 5.1 System description

Before being able to model the KOT part of TSL, a description of the system is needed. This description of the system needs to at least describe the system at the level of the events, as this is where the anomalies due to concurrency occur. With this description of the behavior of the events within the system, a model of the system can be created. In this model, the events behave and move between the different services of the system as in the actual system.

Several products and artifacts are created in the development process (see chapter 4). These documents describe the system at different levels. The description needed for the modeling of the system is preferably as formal as possible, has a certain level of abstraction with regard to the architecture, and contains conditions as to when each event is sent.

The Functional Model created by CapGemini in the context of Functional Model Driven Development (FMDD) describes the system in the most detail. It could therefore be the best place to start. As mentioned by several respondents (see appendix I), this model is

directly translated into C# source code and compiled as the system. This is not the most optimal test basis. Using this FMDD as a basis for testing will test the translator of this model, and not the system design. Furthermore, the level of detail of this FMDD is very high and requires lots of abstractions to be verifiable with model checking (see 5.4). A more abstract description, is provided by the "Service Document" (see [7]). This provides a solid basis for a system model and is therefore chosen as the specification document to base the formal model on. It contains if-then-else statements which describe the behavior of the system using the incoming events, as well as general comments on the IT architecture. As stated before, respondents to the conducted interviews have mentioned (see appendix I) that this document should be the basis, and test basis, of the system, but that this document is not completely up to date. This creates an extra opportunity to apply model checking, as several anomalies that have existed in the system might still be in place in this document.

## 5.2 Model checker

As stated in the introduction, the approach chosen is that of model checking, in stead of model based testing. Model checking is a mature and proven technology to reason about system specifications. Therefore, there are many tools and languages available to use, such as CWB-NC [69], DMC [70], Evaluator [71], Goanna [72], Kronos [73], MCMAS [74], mCRL2 [75], Mec 5 [76], Mur$\phi$ [77], NuSMV2 [78], PRISM [79], SAL [80], SLAB [81], SPIN [39] andUPPAAL [82] First, the prerequisites for the model checker are listed.

First of all, the model checker needs to be able to handle concurrent processes, as the different services of TSL operate independently. Furthermore, preferably, the descriptive language of the model checker contains support for data structures, as several objects (citizen and household) can be identified from the system description. Finally, this descriptive language should be relatively easy to use. This can ensure a fast adoption rate in the usage of formal methods. State space optimisation and reduction techniques are also preferable for a quicker verification of the model.

As stated in the introduction (see chapter 1), the Belastingdienst is considering the usage of formal methods in their development process. In order to be usable within the organisation, sufficient and mature tooling is needed. As model checking originates from the acadamic world, and most academic tools are known to be error prone [83], only very mature model checkers have been considered. From literature, the following tools were found to be mature enough for consideration: SPIN, UPPAAL, Mur$\phi$ and SAL. These tools are well established within the academic world and have made their way into industry. Table 5.1 shows an overview of these tools.

| | **SPIN** | **UPPAAL** | **Mur$\phi$** | **SAL** |
|---|---|---|---|---|
| **Established** | 1989 | 1995 | 1992 | 2002 |
| **Modeling language** | PROMELA | UPPAAL modeling language | Mur$\phi$ description language | SAL Language |
| **Data objects** | bit, bool, byte, short, int, unsigned, arrays, typedef | bounded integers, arrays | subranges, enumerated types, arrays, records,multiset, scalarset | subtype, subrange, array, function, tuple, and record |
| **Concurrent processes** | No upper bound | No upper bound | No upper bound | No upper bound |
| **Reduction and optimisation techniques** | Partial-Order Reduction [39], Bitstate hashing [39], Minimized automaton [84], multi-core verification [85], swarm verification [86] | symmetry reduction [87], bit-state hashing [88] | Hash compaction and state caching using probabilistic verification [89], Symmetry [90], Reversible rules [91], Repetition constructors [92], Multicore and swarm [93] | Bounded model checking, Hash Compaction |

Table 5.1: Comparison of model checkers

It is important to note that UPPAAL and SAL are a different kind of model checker than both SPIN and Mur$\phi$. UPPAAL uses timed automata in stead of finite state automata and SAL is considered a Satisfiability Modulo Theories (SMT) solver. An SMT solver uses symbolic methods to combat state-space explosion problems. However, within the domain of software verification, partial order methods tend to give better performance, and within the domain of hardware verification, the symbolic methods tend to perform better [84].

Using timed automata for the modeling and verification of TSL does not seem to be useful, as no clocks are specified in the specification. While the language of SAL provides excellent constructs to create multiple processes and the complex if-then-else structures of [7], the other constructs (mainly the `TRANSITION` and `RENAME` constructs) of the code make it less suitable for usage for modeling TSL, as the code of the model would differ quite a lot from [7]. Furthermore, these construct make it more difficult to quickly understand how the system works and require an extensive analysis by the reader.

Both Mur$\phi$ and SPIN seem to be equally suitable for the case at hand. Prior knowledge of SPIN and PROMELA has led to the choice to use SPIN and PROMELA for the case study. A system specification in PROMELA is easy to read and understand. The language has basic support for data objects. SPIN is ideal for checking concurrent processes and has a large set of optimisation and reduction techniques to limit the needed resources and speed up the verification. It furthermore offers an intuitive, program-like notation in the form of PROMELA, which aims to specify design choices unambiguously [39]. SPIN offers a powerful, concise notation for expressing general correctness requirements [39]. Finally, SPIN offers a methodology for establishing the logical consistency of the design choices and the matching correctness requirements [39].

Now that both the specification document and the tooling have been decided on, a formal model of the system can be created.

## 5.3 Model

Before modeling the system, it is important to consider the way the SPIN model checker works. Using the proper constructs of PROMELA and options of the SPIN model checker can optimise the verification runs of the model checker [94]. As modeling and verification of the model have been a time intensive process, this section will show in detail how the architecture (5.3.1), services (5.3.2) and environment (5.3.3) have been implemented in PROMELA. This will create a future reference for the *Belastingdienst* to be able to quickly apply the proposed method.

In PROMELA, each statement is atomic. This means that a statement is indivisible, so within a single statement no other actions can start (interleave). Because each step in this PROMELA code can be converted into a seperate state (e.g. due to interleaving of other processes), SPIN offers statements to overcome this behavior by marking a sequence of statements as indivisible [94]. This is called atomicy. This atomicy can be achieved

via the `atomic` and `d_step` constructs from PROMELA. By marking the sequence as indivisible, less interleaving and variable changes are noticed by the model checker, reducing the number of states needed for a verification run. This is therefore an useful approach to cope with the state space explosion problem mentioned earlier in section chapter 1.

These statements cannot be applied at random within the code of the model. Detailed knowledge of the working of the system is required to be able to place these statement. Before going into detail on the placement of these statements, an explanation of the differences in statements is provided. The `atomic` statement groups a sequence of statements and marks them as indivisible. Statements within this `atomic` statement are allowed to block. In SPIN, blocking occurs when an `if` or `do` statement do not contain an `else` clause and all alternatives evaluate to `false`. Due to interleaving processes, the `atomic` behavior is lost when the sequence blocks [84, 95], as the values of (global) variables can no longer be guaranteed to be unchanged.

`d_step` is similar to the `atomic` statement. However, the internal code of `d_step` is not allowed to block and deterministic behavior is not allowed. If deterministic behavior is encountered, the first alternative is chosen [94]. The application of these two constructs in the processes of the model is explained in 5.3.2.

### 5.3.1 Global architecture

As stated in section 3.3, TSL is made up out of different, independent services. These different services communicate through events on an ESB (see 3.4). Independent behaviour in PROMELA is modeled using processes. From this point on, services and processes are considered as being identical. PROMELA offers three ways to communicate between processes: through global variables, rendevous channels and buffered channels. Global variables are variables that can be accessed and modified by every process from the model. Channels are a more convenient way to communicate between processes. In rendezvous channels, messages are sent to the receiving service immediately, and the process will block if the receiving process is not ready to receive. Buffered channels will store the message if the channel is not full. The sending process will not block if the receiving process is not ready, that message will be stored on the channel, and the sending process can continue. This buffered channel therefore resembles the ESB used for NTS the most: it can contain multiple events, as long as the channel is not full and processes can continue to pass events to the channel.

Events contain a name, information on the citizen (*burger* in Dutch) and possibly on the household (*huishouden*) [7]. For events PROMELA offers the mtype construct [84]. This `mtype` construct is used to give mnemonic names to values [95]. The list of NTS event names, defined in the mtype can be found in listing J.1 in appendix J. Citizen and household datastructures (`BURGER` and `HUISHOUDEN` respectively can be found in listing J.2 in appendix J).

The global architecture of NTS contains an ESB. As stated earlier, the buffered channel construct of PROMELA resembles this ESB entity. In the definition of a channel, the

structure of the message is also defined. The definition of the ESB as a buffered channel, as well as the event structure, is listing in 5.1.

```
chan tsl = [CHANNELSIZE] of { mtype, byte, BURGER, HUISHOUDEN, byte };
```

Listing 5.1: ESB and event definition

The arguments are explained in order of appearance: `mtype` contains the name of the event, the first `byte` argument contains the service id (see 5.3.1.1), `BURGER` contains information on the citizen, `HUISHOUDEN` contains information on the household and final argument `byte` contains the number of tries attempted for the event (see 5.3.1.2).

#### 5.3.1.1 Subscription

In a SOA/EDA environment, services subscribe to events [54] published to the ESB. Once an event is published that a service is subscribed to, this service takes this event off the ESB and processes the information from that event. The earlier mentioned blocking feature of PROMELA can implement such a structure. This means once an event that a PROMELA process is subscribed to via the blocking feature, the process continues and can handle the event and the information from that event. However, in PROMELA, if several processes subscribe to the same event, only one of the processes will "wake up" and process the information. The language offers a construct to leave the event on the ESB and let other processes read this event via the *copy* and *polling* constructs [95]. Usage of this approach requires that, eventually, one of the processes removes the event from the ESB. Otherwise processes can infinitely handle the same event over and over again. To overcome this behaviour, an event subscription administration is implemented. Once an event is published, it is placed on the ESB for each service (or in this case process) subscribed. Listing 5.2 displays the basic setup for this mechanism. The listings (5.3-5.9) list the functions (called `inline` in PROMELA) and PROMELA data structures created for usage in the model. For each listing, a brief explanation of the code is given.

```
1     registerEvent(EVT_A, TEMPLATE_SERVICEID);
2     registerEvent(EVT_B, TEMPLATE_SERVICEID);
3
4     services_inited++;
5
6 endtemplate:if
7         listenForEvent(EVT_A, TEMPLATE_SERVICEID)
8         listenForEvent(EVT_B, TEMPLATE_SERVICEID)
9         fi
10 }
```

Listing 5.2: Register to events and wait for published events

Lines 1 and 2 of listing 5.2 represent the registration to event `EVT_A` and `EVT_B` for the service. The constant `TEMPLATE_SERVICEID` is uniquely defined for each service of TSL, ranging from 0 to 8. This is used to subscribe to only one of the instances of an event on the ESB. The (global) variable `service_inited` is used to administrate that the service has registered its incoming events. This is used to stop the environment from sending events before all services are ready to receive (see section 5.3.3). Lines 4 to 6 make the PROMELA process block and wait for events to be published on the ESB. Note the `endtemplate` on line 4. This is a label. Once the process is done handling an event, it will jump to this label (see listing 5.10) and is ready to handle a new event. The name is special: starting a label with `end` means that it will be marked as a valid end state in the model. So every process that is waiting for an event, is in a valid state and the model checker will not mark the system as being in a state of deadlock [95].

```
inline registerEvent(event, id) {
  setOne(eventSubscribers[event].service, id);
}
```

Listing 5.3: Definition of function `registerEvent`

Listing 5.3 shows the definition of the function `registerEvent`. It uses the numeric representation of the event, as `mtype` is used to give mnemonic names to values [95]). This value is used as a key to access an item in the array `eventSubscribers`. The definition of this array is found in listing 5.6, the definition of `setOne` is shown in listing 5.4.

```
inline setOne(a, p) {
  a = setBit(a,p)
}
```

Listing 5.4: Definition of function `setOne`

As in PROMELA an array of booleans is translated into an array of bytes, it is far more efficient to use a list of bits [94, 95]. This is done with the function `setOne`, depicted in listing 5.4. It assigns the result of the macro `setBit` (see listing 5.5) to the variable `a`.

```
#define setBit(data, p) (data | 1 << p)
```

Listing 5.5: Definition of macro `setBit`

The macro `setBit` sets bit `p` of the variable `data` to `1`.

```
1 typedef EVENTSUBSCRIPTION {
2   unsigned service : MAX_SERVICEID+1;
3 }
4
5 EVENTSUBSCRIPTION eventSubscribers[255];
```

Listing 5.6: Definition of eventsubscription datastructures

In listing 5.3, an array was accessed. The definition of this array and its underlying datastructure is shown in listing 5.6. `EVENTSUBSCRIPTION` is defined as a list of bits, as an array of bits (or booleans) is not efficiently translated by SPIN. As `mtype` has a maximum value of `254`, `255` is used as the array length, as arrays in PROMELA start with key `0`.

```
#define listenForEvent(event, serviceid) :: tsl ?? event, serviceid,
    incoming_burger, incoming_hh, retryCount -> active_event = event
```

Listing 5.7: ESB and event definition

Once a service is publishing an event, the subscribers are looked up and for each subscriber, the event is placed on the ESB. This is depicted in listing 5.8.

```
1  inline generate_event(E, b, hh) {
2      int j;
3      for(j : 0 .. MAX_SERVICE_ID) {
4        if
5        :: isOne(eventSubscribers[E].service, j) == 1 ->
             assert(nfull(tsl));
6                                              tsl ! tmp, j, b, hh, retryCount
7        :: else -> skip
8        fi
9      }
10 }
```

Listing 5.8: Definition of function `generate_event`

If a service wants to publish an event to the ESB, it calls the function `generate_event`, as listed in listing 5.8. It loops through all service ids with the variable `j` (line 3) and checks if the service with id `j` is subscribed to the event the service wants to publish (line 5). If a service is subscribed, and the ESB is not full (line 5), the event is published to the ESB (line 6). If the service with id `j` is not subscribed, nothing happens (line 7).

```
#define isOne(data, p) (data >> p & 1)
inline isOne(data, p) (data >> p & 1)
```

Listing 5.9: Definition of macro `isOne`

The macro `isOne` returns the value of the bit on position `p`, so either `1` or `0`.

This concludes the subscribe and publish mechanism for the SOA/EDA environment of TSL, as implemented in PROMELA for modeling TSL.

#### 5.3.1.2 Round robin

In an EDA/SOA environment, events behave freely. It could very well be the case that a stop event for a particular citizen situation arrives at a service before a start event does. The *Belastingdienst* has designed a round robin feature for these cases. If a stop event arrives at a service, and the start situation is not known at that time, an event is "parked" and injected again later. This will not happen infinitely often, the amount of retries is registered in the service. If the maximum amount of retries is reached, an error will occur. This round robin behaviour is modeled as follows: the number of retries is administrated in the event. This is the last argument in the sending (line 6 of listing 5.8) and receiving (line 2 of listing 5.7) of an event. It is also the last argument in the definition of the ESB and its event in listing 5.1. Further actions regarding round robin occur within steps of the service itself and are described in 5.3.2.

### 5.3.2 Services

Each service of TSL is structured in a similar way. This section will describe the global structure of a service in the form of a `template` service in PROMELA, using the earlier described subscribe and publish mechanism (see 5.3.1.1) and other concepts from the documentation. This will display the general application of PROMELA for services in the EDA/SOA environment of the *Belastingdienst*. This provides a structure to go on and apply the earlier described `d_step` and `atomic` constructs to improve model performance and reduce the state space significantly.

The internal architecture of the services of TSL is best described in the Software Architecture Document TSL [8]. For a single business function, the data is handled as depicted in figure 5.1. These 3 steps form the basic structure for the modeling of the business processes in PROMELA.

The system starts in a state called the "*begintoestand*". Within the model, this is stored in a global variable, as multiple instances of the same service should be able to access the same data source. This resembles the underlying database of a service. This global variable is an array that is accessed with the service id. It contains the specifics of the

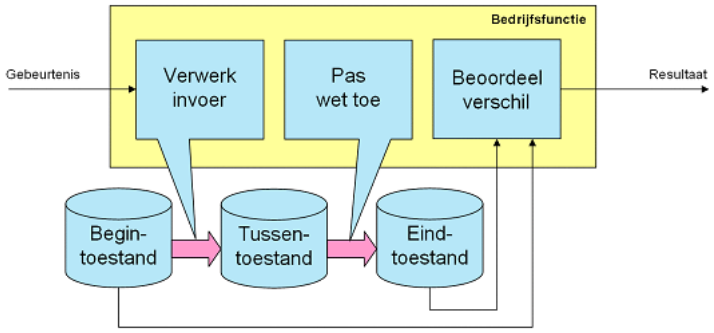


Figure 5.1: Handling of data for business functions [9]

citizens and households for that service. See listing 5.11 for its definition. When an event for a service arrives, the service unblocks and processing continues. As blocking occurs in the last sequence of `template_VerwerkBericht` (see lines 4-7 of listing 5.2), processing will continue with `template_Mapping` according to lines 13 and 14 of listing 5.10.

Within the `template_Mapping` function, the data is copied (line 5 of listing 5.13) to a local variable via the mapping function (see listing 5.12), as multiple instances of the same service can be active and the service should be able to create an intermediate state (remember figure 5.1). These local variables are defined on lines 2 and 3 of 5.10) and relevant data from the event is mapped to the local variable (line 14 of listing 5.10). More details on `mapping` can be found in listing 5.13. This mapping leads to a new, intermediate, state ("*tussentoestand*"), on line 16 of listing 5.10. Legislation is applied to this intermediate state. This legislation is modeled in `template_bf1-template_bfN`, using the if-then-else constructs from [7]. This leads to the final state ("*eindtoestand*"). If data was changed in mapping or during the application of legislation ("*Beoordeel verschil*"), a new event is published (see line 36 of listing 5.10). Note that terminate events (`Tevt_X`) are not published, as they only exist to see that processing has ended. Within the actual information system, these events are used in the *Logische meetpunten administratie* (LMA). Also, manual events (`Hevt_X`) are not published within the model. In TSL, these events are published to the *Kantoortoedeler* for manual handling in the backoffice. Manual handling of events is outside the scope of this research.

```
1  proctype template() {
2    BURGER burgers[MAXBURGERS], incoming_burger; // local burger
3    HUISHOUDEN hh, incoming_hh; // local huishouden
4
5    mtype active_event; // what event to handle
6    mtype publish_event; // event to publish
7
8    byte retryCount;
9    bool data_changed = 1;
10   bool rr = 0;
11
12   atomic {
13     template_VerwerkBericht();
14     template_Mapping();
15   }
16
17   if
18   :: rr -> goto endtemplate;
19   :: else -> skip
20   fi;
21
22   d_step {
23     template_bf1();
24     template_bf2();
25     ...
26     template_bfN();
27   }
28
29   d_step {
30     if
31     :: data_changed == 0 -> skip
32     :: else -> sync_data(TEMPLATE_SERVICEID)
33     fi;
34
35     if
36     :: publish_event == Evt_end_template ->
         generate_event(publish_event, burgers[incoming_burger.BSN], hh)
37     :: publish_event == Tevt_template_gereed -> skip;  // ready to
         handle next event
38     :: publish_event == Hevt_template_uitval -> skip; // ready
39     fi;
40   }
41   goto endtemplate; // event handling done, listen for incoming events
```

Listing 5.10: Template of TSL service structure

Each piece of code is explained briefly below, to explain the underlying functions of listing 5.10:

```
typedef TSLGS {
  BURGER burgers[MAXBURGERS];
  HUISHOUDEN hh[MAXHH];
}

TSLGS tslgs[MAX_SERVICEID+1];
```

Listing 5.11: Datastructures for service information storage

The type `TSLGS` contains an array of citizens (`burgers`) of size `MAXBURGERS`. Information of these citizens is accessed based on the identification number (`BSN`, *Burger Service Nummer* or Citizen Service Number). The same holds for households, only households are accessed via a unique identification number, not via a `BSN`. `tslgs` defines an array of `TSLGS`. Services access this array via their service id.

```
inline template_Mapping() {
  mapping(TEMPLATE_SERVICEID);
}
```

Listing 5.12: Definition of function `template_Mapping`

The `template_Mapping` function calls the general `mapping` (see listing 5.13 function with the service id of the service). This to access the right structure in `tslgs`.

```
1  inline mapping(service_id) { // no non-determinism in this inline,
       because mapping is called from within each service!
2    data_changed = 1;
3    rr = 0; // reinit
4
5    get_data(service_id);
6
7    if
8    :: active_event == Evt_A -> burgers[incoming_burger.BSN].fieldA =
         incoming_burger.fieldA
9    :: active_event == Evt_start_B -> if
10                                :: burgers[incoming_burger.BSN].fieldB == 1 ->
                                       data_changed = 0 // no new information
11                                :: else -> burgers[incoming_burger.BSN].fieldB
                                       = 1
12                                fi
13   :: active_event == Evt_stop_B ->  if
14                              ::
                                    burgers[incoming_burger.BSN].inkomsten_uit_werk
                                    == 0 -> workflowRoundRobin(service_id)
15                              :: else ->
                                    burgers[incoming_burger.BSN].inkomsten_uit_werk
                                    = 0
16                              fi
17     ...
18   :: active_event == Evt_Z -> burgers[incoming_burger.BSN].fieldZ =
         incoming_burger.fieldZ
19   fi
20 }
```

Listing 5.13: Definition of function `mapping`

The `mapping` function retrieves the data from the global variable via `get_data` (line 5 of listing 5.13). For the active event (lines 7-19 of listing 5.13), relevant fields of the incoming data are copied (mapped) to the local data structure of the citizen relevant to the event (`burgers[incoming_burger.BSN]`). The relevant fields are identified from [7]). For start events (line 9 of listing 5.13), a check for new information is in place (lines 9-12 of listing 5.13). If no new information is found, the boolean `data_changed` is set to `0`. For stop events (line 13 of listing 5.13), behavior is different, the earlier mentioned round robin behavior is implemented here (see section 5.3.1.2). If the start is unknown, the event is parked and will be injected again later (see listing 5.16).

```
1 inline get_data(id) {
2   int i;
3   for(i : 1 .. MAXBURGERS) {
4     copy_burger(burgers[i], tslgs[id].burgers[i]);
5   }
6 }
```

Listing 5.14: Definition of function `get_data`

When fetching data from the global variable `tslgs` with the function `get_data`, information of all citizens is copied. This action has to take place, because due to partnerships and other events, not only information on the citizen the event belongs to is needed, but more information needs to be accessed by the service. As stated earlier, this information needs to be available locally.

```
1 inline copy_burger(localB, externalB) {
2
3     localB.fieldA = externalB.fieldA;
4     localB.fieldA = externalB.fieldA;
5     ...
6     localB.fieldZ = externalB.fieldZ;
7 }
```

Listing 5.15: Definition of function `copy_burger`

The `copy_burger` function copies all fields from the `BURGER` datastructure from the global variable (`externalB`) to the local variable (`localB`).

```
1 inline workflowRoundRobin(service_id) {
2   if
3   :: retryCount < MAX_RR_RETRIES -> retryCount = retryCount + 1;
4                           rr = 1;
5                           tsl !! active_event, service_id,
                                incoming_burger, incoming_hh, retryCount
6   :: retryCount == MAX_RR_RETRIES -> data_changed = 0;
7   fi
8 }
```

Listing 5.16: Definition of function `workflowRoundRobin`

If an event cannot be handled, the function `workflowRoundRobin` is called. In the case that the maximum number of retries is reached, the event is dropped and no data is changed (line 6 of listing 5.16). This is analog to the real system behavior, where an event that has maxed out the number of retries is send to the error handling service for manual recovery. If the number of retries has not been reached yet, the `retryCount` is

incremented (line 3 of listing 5.16), the variable `rr` is set to `1` to register the round robin in the service and stop further processing (line 4 of listing 5.16, see lines 17-20 of listing 5.10 for the stopping of the processing) and the event is placed on the ESB. Note that in this case, the `generate_event` is not used. This because this function generates events for all subscribed services, while the active event only failed for the active service.

---

**N.B.** As encountered during the creation of the model, PROMELA has some peculiarities

For example, programmers might want to replace the line

```
retryCount = retryCount + 1; rr = 1; tsl !! active_event, service_id, incoming_burger, incoming_hh, retryCount
```

by

```
rr = 1; tsl !! active_event, service_id, incoming_burger, incoming_hh, retryCount + 1
```

or

```
rr = 1; tsl !! active_event, service_id, incoming_burger, incoming_hh, retryCount++
```

Note that this is not possible! The value of `retryCount` will not get incremented due to PROMELA semantics. Therefore, the RoundRobin behavior will not stop if the start event is never received.

---

### 5.3.3 Environment

The business process "process notifications" (the process which showed that TSL is an important component, see section 3.3) depicted in figure A.1 shows that events for TSL originate from either the backoffice ("*Kantoorportaal*") or FRS. The events that originate from these sources are the (valid) inputs for the model.

```
1  do
2  :: fill_channel == 0 -> if
3                   :: send_event[Evt_A] < MAX_EVT_COUNT ->
                        send_event[Evt_A]++; generate_event(Evt_A, b, h);
4                   ...
5                   :: send_event[Evt_Z] < MAX_EVT_COUNT ->
                        send_event[Evt_Z]++; generate_event(Evt_Z, b, h);
6                   :: else -> fill_channel = 1;
7                   fi
8  :: else -> break;
9  od;
```

Listing 5.17: Sending events to TSL

The first line of listing 5.17 defines a loop. This loop is exited when `fill_channel` is not equal to `0` (line 8). As long as `fill_channel` is equal to `0`, events that have not reached their maximum value of `MAX_EVENT_COUNT` are generated non-deterministic way: the model checker is free in the choice for each of the valid alternatives. When an event is selected, its counter is raised so only `MAX_EVT_COUNT` events are generated. The values for b and h have already been determined. This is done as listed in 5.18.

```
1  inline huishoud_situations(h) {
2    nonDetermine(h.fieldA);
3    ...
4    nonDetermine(h.fieldZ);
5  }
6
7  inline burger_situations(b) {
8
9    if
10   :: true -> b.fieldA = 1
11   ...
12   :: true -> b.fieldA = N
13   fi
14
15   nonDetermine(b.fieldB);
16   ...
17   nonDetermine(b.fieldZ);
18 }
```

Listing 5.18: Random citizen and household values

A citizen can contain two types of fields: byte, short or int fields (ranging to `N` depending on the type of variable) and boolean (or bit) fields which can be either 0 or 1. By determining the value of each variable non-deterministically, all possible combinations of values is tested by the model checker. For byte, short or int fields, this non-determination is shown on lines 8-12 of listing 5.18. Other fields are determined by the function `nonDetermine` (lines 2, 4, 14 and 16 of listing 5.18). More on this `nonDetermine` function is found in listing 5.19.

```
inline nonDetermine(nd) {
  if
  :: true -> skip // nd = 0 (default value)
  :: true -> nd = 1
  fi
}
```

Listing 5.19: Definition of function `nonDetermine`

The function `nonDetermine` selects the value 0 or 1 for the variable nd.

Due to the implementation of `generate_event`, all services will have to be initialised before events are sent to the ESB. For each service, this is done in the following way:

```
  run template();
  services_started++;
```

Listing 5.20: Init service

The process for the service is started, and a counter `services_started` is incremented. This counter is used to stop the environment from sending any events to the ESB before all services are waiting for events (see listing 5.21).

```
  // block, wait for all services to init, before sending out events
      (less interleaving, so less states);
  if
  :: services_inited == services_started -> skip;
  fi;
```

Listing 5.21: Block until all services are initialised

The environment will block until `services_inited` is equal to `services_started`. This is the case when all Promela processes have passed the registration for events part of the process (see listing 5.10).

```
init {
  byte services_started = 0;
  BURGER b, b1, b2;
  HUISHOUDEN h;
  byte retryCount,send_events;
  bool send_event[255] = 0;

  bool fill_channel = 0;

  init services

  block

  send events

}
```

Listing 5.22: Environment initialisation

## 5.4 Abstractions

To reduce the amount of space needed for a state, and to lower the total number of states, several abstractions were made. This section will describe these abstractions. The abstractions made involved the BSN, income and the time model of TSL.

The first abstraction made was that of BSN. This service number normally is made up out of 9 digits. As it is not needed to model all citizens (the model checker will check all possible situations), a shorter number can be used to identify a citizen. On a side note, only one or two citizens are insufficient to oversee all relationships that can exists between citizens, for example a mother, father and child already require three citizens.

A byte variable, which can contain a number up to 255, is large enough for this, as a maximum of 255 citizens can be identified this way. This will save at least 24 bits per state, as all variables are included in the state vector. Given that for citizens the number of states is over $2^{64}$ (64 fields which can contain at least two values), this is a significant amount, while still being able to uniquely identify a citizen.

Another second abstraction that was made was regarding addresses. These are also modelled using a byte variable, in stead of for example a zip code containing four digits and 2 letters. This again provides an unique identification number, while lowering the number of bits needed.

As stated before, the benefits are income related. Representing these incomes will drastically increase the state space due to the enormous number of values such a variable can have. The business functions of TSL translate these incomes to boolean values, such as income above limit, household income zero. In stead of using incomes, these boolean expressions are attributes of the citizen. The same holds for several other values, such as hours of childcare received.

A rather important feature of the system is that of "time travelling". This concept is used for retroactive calculation if a notification of a situation in the past is provided to the system. To be able to use this concept in a model would require an extensive administration of current and previous situations of a citizen. In a model checker, this would lead to an enormous explosion in state space. As this study is a pilot study for the usage of formal methods, only current situations are administrated in the model. As the same business functions are used in retroactive and current calculation, the method will show these business functions are complete and error-free. This will show the added value of formal methods to the development process. More on the added value of formal methods is described in chapter 7.

Finally, events from civil servants (G-events, AB-events, B-events) have not been modelled. These AB events jump over the processing of the service, as these events already contain a result for this service, decided by the civil servant. These events have no other impact on the KOT chain. Same holds for the H-events as well as decisions on objections, these events only occur in special cases where a civil servant needs to verify or check certain data from the citizen or when a citizen has objected against a decision. These are outside the scope of this research.

## 5.5 Bundling services

In the model described in the previous sections, each business function is created as a separate service. This is analogous to the initial system design. However, during the first system tests, this design imposed some serious performance issues, mainly on the ESB. The proposed solution was a bundling of services, to reduce the number of messages being transferred via the ESB. This bundling merged the four AWIR services into an AWIR service and the services for *grondslagen* (*draagkracht*, *huishoudsamenstelling* and *lasten*),

*beslissen* and *beschikken* into a GBB service. In stead of 9 seperate services within TSL, the KOT chain now only contains two services.

This bundling of services has been embedded in the original model. Following the instruction from [10–12], changes have been made to the model. Whenever a change is made, the original code is kept intact via the following construct:

```
#ifdef UNBUNDLED
  original code
#else
  new code
#endif
```

Listing 5.23: Bundled and unbundled structure

## 5.6 Validation of the model

Before verifying the model with SPIN, it needs to be ensured that be model is build correctly. In order to visualize the working of the model, SPIN offers the possibility to create a Message Sequence Chart (MSC). Such a MSC displays the movement of messages between the different processes, or in the case of the model of TSL: the movement of events between services. However, the default MSC created by SPIN for the created model is not readable (see figure 5.2), due to the high number of variables inside a message (event). Luckily, the MSC is created as a PostScript file, and is therefore plaintext. As the content of the message is not relevant to the overview of the events, this content is removed from the events. When generating a MSC, SPIN creates a bounding box around the events and its content. As the event contained many variables, this bounding box was very large. By removing the content from the event, this box has become redundant. Therefore, the bounding box is removed from the MSC. This makes the MSC readable and uncluttered, see figure 5.3.

Furthermore, if one MSC would be created for all incoming events at one, the MSC will significantly grow. Therefore, for readability, a MSC is created for all incoming events seperately. This MSC is compared with the overview provided by [13]. The modification procedure can be found in the appendix K. The validation of the model has shown that the model behaves as the system and is therefore valid.

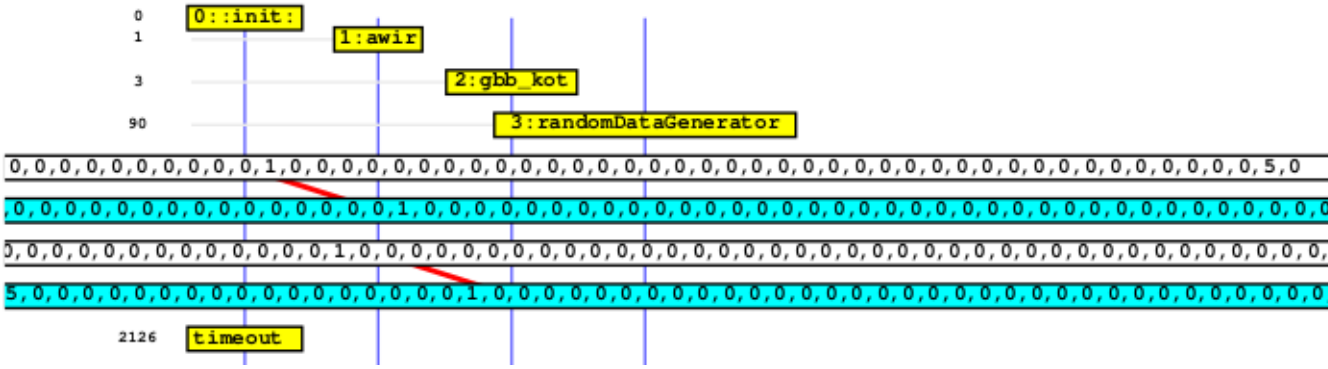


Figure 5.2: Default MSC

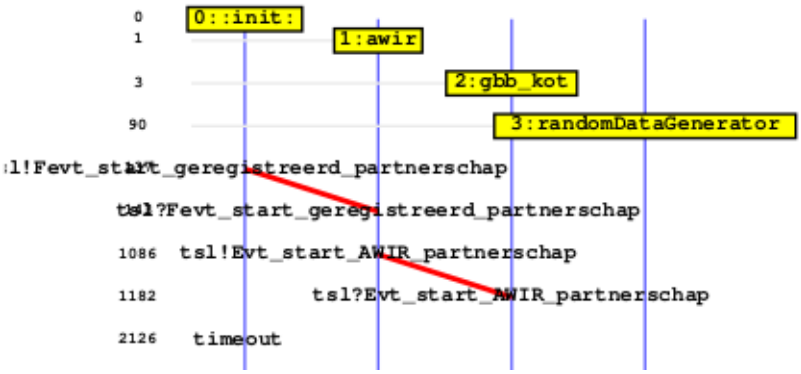


Figure 5.3: Improved MSC

## 5.7 Verification of the model

Verifying a PROMELA [96] model with SPIN is a two-step process. First, the model is translated into a pan verifier (see [97]) describing the system in C source code (pan.c and related files) . This C source code is then compiled into a runable verifier (pan). This first step is done by calling `spin` with the command line parameter `-a`:

```
spin -a model.pml
```

This generates several C source code and header files. These files are compiled into an executable verifier `pan`:

```
gcc -o pan pan.c
```

Several command line options can be given to the compiler to improve the performance of the verifier. Options include reduced memory usage through compression, minimized automaton, hashtables and bitstate searching. One of the options that this model requires is the `VECTORSZ` option. Because of the large number of variables, the default size of the state vector is insufficient for this model. A vectorsize of 16384 bytes was found to be sufficient to fit the state vector.

```
gcc -DVECTORSZ=16384 -o pan.c
```

Because of the large number of states (due to the high number of messages and variables) and the large state vector, memory requirements are very high. To help to cope with these memory requirements, several options can be used to reduce these requirements. The `COLLAPSE` (memory compression) and `MA` (minized automaton) options are used for this:

```
gcc -DVECTORSZ=16384 -DMA=8300 -DCOLLAPSE -o pan.c
```

For a optimized verfication run, the compiler can be called with an optimization flag. This increases the compilation time, but improves the performance of the verification run. As the verification run is improvement significantly, this option is advisable. The optimization flag is used as follows:

```
gcc -O2 -DVECTORSZ=16384 -DMA=8300 -DCOLLAPSE -o pan.c
```

The `SAFETY` option optimizes the code of the verifier for the case where no cycle detection is needed. This is the case when no linear temporal logic (LTL) properties or never claims are used.

```
gcc -O2 -DSAFETY -DVECTORSZ=16384 -DMA=8300 -DCOLLAPSE -o pan.c
```

The compilation and verification of the model takes up a lot of time. A single compilation takes around 30 minutes and verification runs for several days, due to the high number of states. SPIN offers several approaches to speed up this verification. These approaches are discussed in the section 5.7.1.

The model is verified by invoking the pan executable. This verifier contains several command line options. Two of these options were used in the verification process: `-n` to hide listing of unreached states and `-q` to require an empty channel in valid end states. This means that an end state is only valid if all events are handled.

```
./pan -n -q
```

This is the default way SPIN can be used for verification of the stated properties of model. However, there are several extensions to SPIN which can help to speed up the verification or reduce the state space. These are discussed in section 5.7.1.

On a side note, there are Windows interfaces to SPIN available, which improve the usability of SPIN.

### 5.7.1 Other approaches

SPIN has several other approaches to cope with memory issues. First of all, there is the option of Stack Cycling (`SC`), which lowers the amount of memory used by the verifier. This has however, influence on the performance, as part of the stack is stored on disk. Furthermore, this approach is only useful for verifications that require an unusually large depth-limit, which is not the case for the model of TSL.

Another option is the usage of multiple cores or processors of a system, via the `MULTICORE` option. However, this option requires that the user running the verifier is able to raise the amount of shared memory of the system. On Linux, this requires root priviliges. As these priviliges were not available on the machine used for verification (4 quadcore processors with 128G RAM), this option was tried unsuccesfully.

Another option that can speed up the verification process, is a bitstate search (`BITSTATE`). It should be noted that such a search does not provide a full state search, and a full prove over the model is therefore not garantueed. By using Swarm [86, 98], the coverage over the model can be raised, by creating several executables with alternating variations of the `REVERSE` and `T_REVERSE` options and including randomize options `P_RAND` and `T_RAND`. Using this method, errors in the model such as assertion violations are found relatively quickly, but runs without errors can execute for a very long time ($>$ day).

### 5.7.2 Extensions to SPIN

From literature, several extensions to SPIN are available to improve performance or lower memory requirements. Some interesting extensions that provide improvements to the verification runs are discussed.

TOPSPIN [99] uses computational group theory to determine a group of component symmetries. It automatically modifies the model checking algorithm to exploit these symmetries during verification. This can result in significantly reduced memory consumption, and a faster verification time. However, the latest version of this tool originates from 2010, and does not feature the new SPIN features employed in the model. This makes it impossible to use this tool.

DTSpin [100] is another extension to Spin. It extends Spin with discrete time. This enables Spin to be used for verification of concurrent systems that depend on timing parameters. These timing aspects are not relevant for the verification of the system as timing aspects are not part of the specification.

CPOR-Spin [101] exploits the hierarchy of the verified system for more efficient verification. The tool features an improved version of the Partial Order Reduction algorithm called Clustered Partial Order Reduction. The tool is however not compatible with some of the newer features of Promela that are used in the model and does not seem to be actively maintained.

LTSMin [102] is a toolset for manipulating labelled transition systems and model checking. It allows to reuse existing tools with new state space generation techniques. LTSMin uses SpinJa [103] as an interface for the Promela languge. Several features employed in the model (`typedef`, random receive and d_step communication) are not yet available in the tool. The tool is therefore not suitable for usage.

### 5.7.3 Properties

For a verification run, several properties can be created in the model that the verifier can check. Default properties include safety and liveness properties. These properties are created by adding valid end states to the model. These valid end states have already been shown in listing 5.2.

Within the model, assertions can be added, which are checked at runtime. These assertions can check the values of variables. If the comparison fails, this is reported. An application of this approach is shown in chapter 6. Other assertions include that after the processing of the event and the legislation, one or more events should be published by the service.

Another interesting property is that there exists a possibility to be eligible for a benefit. In the state space this would require a path to a state where a benefit is given. Or mathematically: $\exists \diamond benefit == 1$. However, this is a property that is part of Computation Tree Logic (ctl). Spin only supports Linear Temporal Logic (ltl). This ltl is best used in fairness properties, defined as constraints on cyclic executions, which state, for example, that every cyclic execution either must traverse or may not traverse specific types of states infinitely often [84]. ctl can express properties which state that from every state there exists at least one execution to an accepting state. Fairness in tsl is already expressed in the model, via the empty channels and end states.

While there has been an attempt to add ctl to Spin [104], there is currently no support for this.

## 5.8 Findings

During the creation of the model several findings for the system design and specification [7] have come forward. First of all, the structure of the document has made it very difficult to fully understand the design of the system, for example the decision of the AWIR partnership. The functions for deciding this AWIR partnership are described using an article from the legislation on AWIR, to which a large list of bullets is added. These bullets describe extra requirements for the function. Because it is an extensive list, this makes it difficult to fully understand the working of this function.

In the document, the names of events is not consistent. For example, `Evt_hh_kinderopvangtoeslag`, `Evt_HH_kinderopvangtoeslag` and `Evt_huishouden_kinderopvangtoeslag` are used to represent the same event. Furthermore, not all business functions are completely specified in [7] (see the listing below). The following listing shows findings from [7]:

- Textual
  - "some details are mentioned"

    Such sentences should not be part of a specification. It is very unclear if the specification is correct and complete. Furthermore, it leaves room for the programmer to give his or hers own interpretation.
  - "this function has the same behavior as function X, with the following exceptions"

    Such a specification should be avoided. It highly depends on the specification of function X. While the specification of X might be a complete specification, specifying a function this way leaves lots of room for human error. Furthermore, if function X is changed, a lot of rework might have to be done.
  - "the `Hevt_X` is replaced with a terminate event"

    It is important to fully specify behavior, including event names. As terminate events are using in the LMA, it is important to have the correct event names in the specification.
  - "`Evt_X` or `Evt_Y` (both) and `Evt_Z`"

    This is an unclear specification. `Evt_Y` is known to have two variations, `Evt_Y1` and `Evt_Y2`. What is meant with "both"? Is it $X||(Y_1\&\&Y2)$ or $X||Y_1||Y2$. These are not logically equivalent, and as a programmer can give its own interpretation to the specification, the system can show unwanted behavior.

- Verification
  - Through verification of the model, it has come forward that for a `Cevt_tijdstip_beschikken` event, no behavior is specified for the case that no concept depositions have been created. A verification run showed an assertion

violation for this case. This problem occurred in the disposition (*Beschikken*) service.

These findings have not shown any issues regarding concurrency. Therefore, the stated hypothesis: "Concurrency is the cause of the anomalies in KOT and model checking can detect these anomalies" is not validated.

Furthermore, the documentation contains an extensive list of change requests. This can imply that the specification has been incomplete or contained errors. However, to substantiate this statement, a detailed analysis of the changes to the specification document [7] is needed. However, change requests have only come into existence after the production date. Therefore, many changes to the document have not been properly documented in change requests. This makes is hard to see if anomalies as incompleteness have been removed from the specification.

## 5.9 Conclusions

This chapter was guided by questions 6, 9, 7, 8, 10 and 11 of 1.4. Question 6, where is the system for KOT described, can not be answered unequivocally. There are several levels on which the system is described. However, the Service Document [7] is regarded as the basis of the system.

The next question, 9, what characteristics should the specification of business processes or systems have to be suitable for model checking, is easier to answer. Findings from the case study show that the specification should be clear and unambigious, and should contain the abstract behavior of the system or business function.

Question 7, what specification language and tool is best suited for the modeling and verification of KOT, is answered with PROMELA and SPIN. This pairing is chosen due to the C-like structure of PROMELA and the maturity of SPIN.

What level of abstraction is to be used for the modeling of the system supporting KOT, question 8, is difficult. The current abstractions made, make it possible to see a working model the behavior of the events. But the aspects of the Time Object Model (regarding the start and end time of data) is removed. This abstraction lowers the state space, but also reduces the funtionality of the model, as only the current situation of the citizen is stored. Time travelling is not possible. This has some influence of the validity of the model.

The errors found by model checking are diverse. The errors found include ambiguity in the specification and incompleteness of the specification. This answers question 10.

The final question this chapter aimed to answer was question 11: to what extend does model checking improve the specification the supporting systems? By using model checking ambiguity and incompleteness is removed, the working of the system can be visualised

and simulated. This helps designers to see what they have created and what issues can occur.

# Chapter 6

# Analysis of known errors

Within NTS, several errors are known. These errors are registered as Candidate Known Errors (CKE) or Known Errors (KE). This chapter will show that these errors can be detected with model checking and could therefore have been prevented. Note that as these errors have already been identified, workarounds are available and the errors can therefore not be exploited.

For the chain used in this study, KOT, a total of over 10 (C)KE were identified. After careful investigation, most errors originate from databases behavior. This is outside the scope of this study. One KE, 111, and one CKE, 190, were selected for investigation as the description includes ordening of events and calculation errors.

## 6.1 KE 111

Known Error 111 is an error that leads to an event ending up in the Error Handling Service (EHS). The event that ends up in this EHS is an event regarding the ending of a partnership: `Evt_einde_AWIR_partnerschap`. The analysis in this KE describes the situation and shows that the error occurs when a start event overtakes an end event. The end event is given to the EHS because the partnership the event is trying to end is not found. Listing L.1 in appendix L shows the initial situation for the error (lines 6-31) and the data for the events (line 36-53). According to the description of the error, only two services, namely the AWIR partner service and the household service play a role in this error. To improve speed and rule out behavior by other services, only the AWIR and household service are started for the analysis of this Known Error. The initializing of these service in PROMELA is depicted in listing 6.1.

```
run F_bepalen_AWIR_partner_gevolgen();
services_started++;

run F_vaststellen_huishoudsamenstelling_kinderopvangtoeslag();
services_started++;
```

Listing 6.1: Starting relevant services for KE 111

To show that the system shows correct behavior when the events arrive in order, an analysis with a first-in, first-out (FIFO) ESB is performed. This is done by modifying the listening to events part of the receiving service, in this case the household service (*Vaststellen huishoudsamenstelling*) to make the queue FIFO. In stead of using random receive [95] (The "??" construct in PROMELA) for the implementation in the model), receive ("?" in PROMELA) is used to create a FIFO ESB, as the initial idea of the designers was that the ESB would function in a FIFO way. See listing 5.7 for the initial implementation of the random receive.

```
#define listenForEvent(e,s) :: tsl ? e, s, incoming_burger, incoming_hh,
    retryCount -> active_event = e
```

Listing 6.2: FIFO ESB

The existence of the error is shown by adding an assertion to the mapping function:

```
assert(burgers[incoming_burger.BSN].AWIR_partnerschap ==
    incoming_burger.AWIR_partnerschap);
```

Listing 6.3: Assertion for KE 111

This assertion checks if the partnership the events is ending, is indeed the current partnership of the citizen known to the service that is processing the request.

A full state space search of the model, given the input described in the KE document, shows no violations of this assertion. The model, and therefore the design, have defined correct behavior with respect to these events and the ordening of events.

However, when adding an extra household service (see listing 6.4), the error does occur due to the possible simultaneous retrieving of the events from the ESB. This is in line with the stated hypothesis in section 1.1 that anomolies in KOT occur due to concurrency. Listing 6.4 shows how this extra service is started.

```
run F_vaststellen_huishoudsamenstelling_kinderopvangtoeslag();
  services_started++;
```

Listing 6.4: Starting an extra service

A verification run shows that the assertion in the model now violated, even if no extra household service is in place.

### 6.1.1 Extension of the error

Looking at the underlying behavior of the AWIR partnership, it shows that several levels of this partnership exist. The partnership that is ended is a partnership based on paragraph 1C of article 3 of AWIR, while the new partnership is based on paragraph 1A of article 3 of AWIR (see [7]). Partnerships based on paragraph 1B of article 3 (see [7]) will also end the partnership based on paragraph 1C of article 3. This means that the statement that this error only occurs in a specific case is possibly false. Therefore input for the model is extended to check partnerships based on paragraph 1B. While the analysis states that two input events are required (*handtekeningrelatie* and *geregistreerd partnerschap*), this extension uses single events for the cases states in paragraph 1B of article 3. A verification run shows that indeed the known error marked as KE 111 can occur for these cases: for all events that lead to a partnership based on paragraph 1B of article 3 of AWIR the error can occur.

While this extension shows that the error is larger than depicted in the Known Error report, it should be noted that in the real system this problem will only occur for similar timestamps for the events. As timestamps have been abstracted from the model, each event has an identical timestamp.

Looking into numbers regarding AWIR partnerships, there are for example currently $287,480$ AWIR partnerships[1] for people with a common child (*gemeenschappelijk kind*) that do not have a registered partnership. This means that $2 \times 287,480 = 574,960$ citizens. Highly theoretical, every one of those citizens could have started a registered partnership with another person on the same date as the birth of their child. Each of those registered partnerships could lead to known error 111, given that the birth of the child arrives before the registered partnership.

### 6.1.2 Solution

The known error 111 occurs when a stop event of the current partnership is overtaken by a start event of the new partnership. A possible solution is to use the round robin mechanism described earlier in 5.3.1.2. This is done by replacing the mapping behavior of `Evt_start_AWIR_partnerschap`. A partnership can only be registered as started when no current partnership is registered at the service. If another partnership is registered, round robin will be used to wait for the stop event.

[1] Numbers retrieved April 11, 2012

```
if
:: burgers[incoming_burger.BSN].AWIR_partnerschap ==
   incoming_burger.AWIR_partnerschap -> data_changed = 1
:: else ->  if
        :: burgers[incoming_burger.BSN].AWIR_partnerschap == 0 -> map
           fields
        :: else ->  if
                :: rr == MAX_RR_RETRIES ->
                    assert(burgers[incoming_burger.BSN].AWIR_partnerschap
                    == 0);
                                    map fields
                :: else -> workflowRoundRobin(service_id);
                fi
        fi
fi
```

Listing 6.5: Fix for KE 111

The verification run that was performed after this change to the model showed no violations of the assertion.

## 6.2 CKE 190

Candidate Known Error 190 is an error in the calculation of the AWIR partnership. It is an error that has occurred during the regression test. The situation is complex, and as stated in the description of the error, it is not likely to occur in real life. The starting situation contains 8 citizens, which have several relationships, including partnerships, shared households and children. The details of this CKE are listed in appendix L.2. The AWIR function states that when a new partnership is formed, all citizens affected by this partnership, such as previous partners, should be rechecked for a partnership. The error that has been found is that not all citizens influenced by the partnership are rechecked and not all valid partnerships are created for the given test situation.

To verify this, a ltl property has been created:

```
ltl {
!<> tslgs[AWIR_PARTNER].burgers[4].AWIR_partnership == 5
}
```

Listing 6.6: Property for CKE 190

This property means that eventually the citizen with BSN 4 should have AWIR partner 5 registered in service identified by `AWIR_PARTNER`. A verification run shows that this property is satisfied. This means that all partnerships are started and ended correctly. The conclusion from the report that this is an implementation issue is correct, given that the model is valid.

## 6.3 Conclusion

The analysis of KE 111 and CKE 190 shows that model checking can aid to prevent errors in system design. It can also help in the analysis of the error, by adding assertions as done for KE 111. The fact that these errors did not occur in a full verification run as described in chapter 5 is two-way: first of all, the assertion in the mapping of the partnership was not in place when the full verification run was performed. Furthermore, due to the usage of Swarm and bitstate verification techniques, this error might have been missed.

Furthermore, the analysis of KE 111 has shown that concurrency is the cause of this known error, given that the ESB is FIFO. This shows that the hypothesis: "Concurrency is the cause of the anomalies in KOT and model checking can detect these anomalies in the design" is valid for the given case.

# Chapter 7

# Application of formal methods within *Belastingdienst*

Chapter 5 and 6 have shown the application of model checking on the specification of KOT and in the analysis of some known errors. But how can this technique be applied within the development process of the *Belastingdienst*? This chapter will show the experience of applying model checking to the existing specification of an information system. From this experience, recommendations on the application of model checking in the development process of the *Belastingdienst* are given. These recommendations are guided by questions 2, 12 and 13 (stated in 1) regarding the requirements and changes to the development process, the knowledge and education level and the general usability of model checking.

## 7.1 Case study experience

Creating the model from the current specification has, among others, given insight into the effort needed to create such a model. These experiences are discussed in this section.

First of all, the time intensity is something that should not be underestimated. The creation of a model from an existing specification can take up a lot of time. As model checking is known to benefit from abstractions, the system has to be well known to be able to make these abstractions. The abstraction will have to be made before the system is actually modeled, because abstraction in an existing model is a hard thing to do.

Furthermore, the knowledge of the model checking tool and modeling language by the modeler determine the amount of time needed for the creation of the model. Creating such a model requires detailed knowledge of the modeling language, the verification tool and the system under investigation. For such a vast system as TSL, abstractions have to be made in order to be able to complete a verification run once the model has been fully created. But in order to create such abstractions, knowledge of both the system design under investigation and the model checker is needed to be able to make the right

abstractions. Only experts in the domain of *Toeslagen* are able to do so.

Verification of the current model is a time consuming process. The generation of C source code from Promela is fast, and takes a couple of seconds. Compiling the model to a runnable verifier is a more time consuming process. Without compiler optimizations, this process takes about twenty minutes. With this optimizations flag (`-O2`) enabled, around 30 minutes pass by until a runnable verifier is available. A run of the verifier can take up to several days, depending on the number of starting instances of citizens chosen. Even with optimizations enabled and using Swarm, the final verification runs have taken up to several days (runs have shown runtimes of $5.63 \times 10^5$ seconds, checking $1.469532 \times 10^8$ states) for a single run.

Using Swarm for parallization of the search has shown to be significantly quicker in the detection of errors. However, as Swarm uses four different verifiers, the compile time increases slighty. But this small overhead is paid back when looking at the time saved in verification. Errors are found relatively quickly compared to the single, full state space search (around 1 hours versus several days). If no errors are found, this process can also take up to several days, and the seperate runs of the verifiers will eventually end up checking the same states.

## 7.2 The position of Model Checking in the development process

From literature [105, 106], the position of model checking in the traditional waterfall method can be identified. This is shown in figure 7.1.

Verification of the model takes places after the design phase in the model, while debugging is done in the analysis and design phases. Note that while figure 7.1 shows debugging after the Code phase as well, this is not part of model checking, as the code is not part of the model. The position of model checking makes perfect sense, as a model is, as stated before, (a representation of) the design.

Although the *Belastingdienst* uses the V-model and not the waterfall development model for its development process, these analysis and design phases can still be identified (remember figure 1.1). Looking at the V-model (see figure E.1 of appendix E) of development process the *Belastingdienst* and the description of this development process in 4.1, these analysis and design phases are part of this V-model as "*Opstellen globaal ontwerp*" (Analysis) and "*Detailontwerp Service*" (Design). These phases are the best fit for model checking to be embedded in the development process of the *Belastingdienst*.

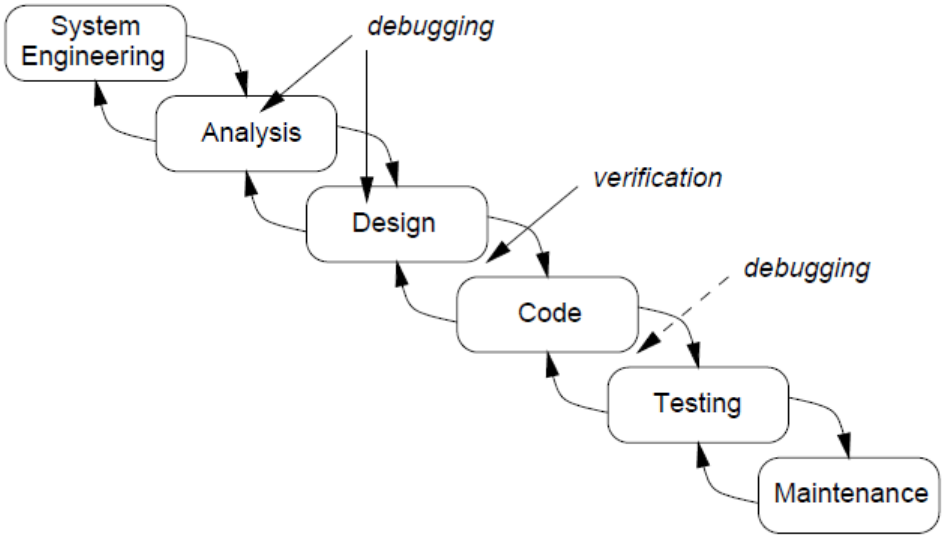


Figure 7.1: Position of model checking in the traditional waterfall process for software development [105, 106]

## 7.3 Level of usability

Within the first phases of the development process of the *Belastingdienst*, products are reviewed as part of the quality process. This can be seen in the V-model (see figure E.1 in appendix E)This is depicted as *Toetsen* in the image, the Dutch term for such reviews.

As PROMELA models are for the most part easy to read and understand, these formal models can be reviewed in the development process, as is currently done with all documentation that is created in the first phases. The important difference between the current documentation and the formal model is that the level of ambiguity is decreased, due to the languages used. While the model language PROMELA, like many other programming languages, has features to add comments to code, these comments should be kept very brief. These comments should be a short clarification of the code, longer texts will not add clearification as ambiguity of the comment is likely to rise.

In addition to the reviewing of documents as part of the development process, verification of the model with SPIN can be added to this development process. This implies that the currently identified phases of the development process do not need to be altered. An extra step, the verification of the model, can be added. To be able to perform such a verification, it is important that the properties that need to be checked, are available and have been reviewed and approved by others. Only domain experts can do so. This to ensure that the properties are valid and specify a situation that needs to be checked. The creation of the properties requires knowledge of Linear Temporal Logic. As LTL is currently mainly used in academic and higher education areas, it is wise to say that it requires a higher or academic level to be able to use this logic.

Finally, if the system suffers from an anomaly, the model can aid in the analysis of this anomaly. If provided a start condition, relevant event and condition of the anomaly, these events and conditions can be entered into the model. Verification will lead to one or more error paths to the error. If this is the case, an error in the design is identified. If no errors are found, the most likely cause is an error in the software, as the design did not include the error. Using this technique can speed up the analysis of errors, less manual work is needed for the analysis. This will eventually lower the personnel costs, as external personnel is used for this analysis.

## 7.4 Level of knowledge and education

As the case study has shown, detailed knowledge of the architecture is needed to create a model of the system. Knowledge of the architecture alone will create a very abstract version of the system. Details of the services will have to be added as well, which requires in-depth knowledge of the business logic of the services. Both knowledge of the architecture and of the business logic of the services is available in the organisation, as the system specification [7] already contained this information.

To be able to create a model, a low level degree of programming knowledge is required. While PROMELA is in fact not a programming language, its syntax resembles the C/C++ language. Prior experience with such a language can help to create the model. As PROMELA itself as several peculiarities, a set of best practices is to be shared among thosee employees that will create model, in order to escape these peculiarities.

Specifying properties is another important aspect to consider. Properties need to be specified that can be checked in the verification of the model. Knowledge of the architecture, model and application domain is needed to do this. Without knowledge of the domain, it is difficult to create valid or important properties, as exit criteria need to be known. For example, a citizen is not allowed to receive child care benefits for year $T$ if this citizen has been in detention during the entire year $T$. This knowledge is only known by domain experts.

Finally, knowledge of the SPIN and its command line and compilation options is required to generate and compile the optimal model for verification. Without these options, verification is less likely to complete or succeed in a timely fashion.

## 7.5 Implementation in the development process

To embed model checking in the development process, several things are required. First of all, people using the technique will require time to get to know the language and tool, as the learning curve is long. The case study has shown that the transformation of current documentation into a formal model as an useful approach in this learning process.

As knowledge is spread throughout the organisation, several people will work on the model simultaneously. This requires extra efforts in version management, as well as communication between the modelers with respect to the abstraction level and data objects.

Finally, an extra step, the verification of the model, will have to be added to the development process. As the "Detailontwerp service" is the final step of the V-model in which model checking is applied, the verification can be performed after this phase.

The modeling will be performed by the architects and analysts from these phases: Lead architect, Tactical architect, Project & ict architect, Strategic architect and Functional analyst (see table G.1 in appendix G), together with domain experts, forming a multi-disciplinairy team. The responsibilty for the verification will be in the hands of B/CAO (see F.1)

## 7.6 Conclusion

This chapter was guided by three questions from 1.4. These questions have been answered in the previous sections. The most important aspects for each question are mentioned in this conclusion.

The prerequisites and changes needed in the development process of the *Belastingdienst* for a successful usage of model checking (question 2 of section 1.4) are the usage of the formal language in the system design and the addition of a verification step next to the review process.

The needed education level and knowledge for model checking (question 12 of 1.4) is high. Modeling a system requires a lot of knowledge of the architecture, domain and business logic of the services. For verification, detailed knowledge of the model checker, as well as Linear Temporal Logic is needed. This requires a higher or academic level of education.

The usage of model checking (question 13 of section 1.4) is wide. Formal methods can be used in the analysis and design phases of the development process, to specify system behavior. This resulting formal design is to be used to verify behavior and properties of this design with model checking. While experience from the case study has shown that modeling is a time consuming process, it does not seem to be more time consuming then the time needed for the creation of the documentation in the current format. As the system design has been extensively described in the specification, this has been a time consuming process as well. Using a formal language as PROMELA will therefore not increase the time by a relevant factor. However, the formal model can aid in the analysis of errors from the software, as well in the prevention of errors through verification.

# Chapter 8

# Related and future work

This chapter will show the added value of this research to the current literature. Recommendations on future work in the research areas are also suggested. The research is divided into three key areas: TSL and administrative systems, software design verification and the embedding of new technology in the development process.

## 8.1 TSL

TSL and the underlying technology have been the subject of several scientific studies. The FMDD technology has been investigated in [107, 108]. Both [107] and [108] have focussed on the language aspects of the FMDD.

Other administrative systems seem to have received little attention with regard to formal methods. The research performed on a pension administration system [19] shows that formal methods can be applied to such a system. This research has not analysed the development process nor gathered experiences on the used development process.

One of the important aspects of TSL that is interesting to investigate is the Time Object Model. This feature of TSL is an essential component of the system, and is used extensively. Because of the abstractions made (see 5.4), this TOM was not part of the model, due to state space explosion and complexity. By adding the TOM, the state space is likely to rise. But the Swarm technique has proved itself useful to cope with the problems this brings in. The larger similarity the model will have when TOM is included, can help to increase the coverage of the software tests performed.

## 8.2 Software design verification

Software design verification through model checking is an area that has received a lot of attention by researchers. Its application include protocol verification, control software

[109], compilers [110], operating system kernels [111] and e-voting system. These are all embedded systems or systems with a defined input range.

Model checking merely reasons over a model, without any regard for the software implementation of the specification. It is however possible to link model checking to the software implementation. For example, model checking can be used to generate test cases [112]. Furthermore, SPIN can include C code to link the model to a software implementation. This is a simple form of Model Based Testing, a method closely related to model checking (described in chapter 1). MBT is another interesting area of research for the *Belastingdienst*, as it can help to further automate the testing process and increase the test coverage.

## 8.3 Embedding new technology in a development process

The application of formal methods in industry has received little attention. Some recent studies [113–115] have looked into aspects of the usage of formal methods in an industrial setting, but there are many things to explore in this area. Therefore, another point of investigation is the embedding of the model checking method in the development process.This report positions the model checking technique in the process and contains recommendations on the usage of this technique. However, these recommendations are global and abstract. Further research is required on the change management aspects of embedding this technology in the organisation to give detailed recommendations regarding the usage of the technique within the *Belastingdienst*.

# Chapter 9

# Conclusion

This research was guided by a main question and hypothesis. In support of this question, several subquestions were created. The results of these questions will be discussed and the main question will be answered. This main question was:

> What steps are required for a successful implementation of model checking within the development process of the *Belastingdienst*'s *Toeslagen* program?

Before providing an answer to this main question, the subquestions will be discussed briefly.

*1 What is the organisational structure of the Belastingdienst?*
The *Belastingdienst* is part of the Ministry of Finance. The *Belastingdienst* is divided into several units. The Central Office supports the other units of the *Belastingdienst*.

*2 What prerequisites and changes are needed in the development process of the Belastingdienst for a successful usage of model checking?*
In order to be able to use model checking in the development process of the *Belastingdienst*, the formal language needs to be used in the system design. To utilize the model checking technique, an extra step needs to be added to the development process. This can be done in the review process of the detailed design of the system, to which a verification step is added.

*3 What departments and units are involved in the Toeslagen program at the Belastingdienst?*
*Belastingdienst/Toeslagen* is the execution body of government for benefits (*Toeslagen*), while these benefits are established and organised by three different ministeries, which do not include the Ministry of Finance.

Within the Central Office (B/CA), most units have involvement in processes of *Toeslagen*.

*4 Who was involved in the development process of the Toeslagen program, what role did they have and how have they experienced this development process?*
A lot of people have involvement in the development process of the *Toeslagen* program. Their experience was gathered via interviews and the results were structured along Critical Success Factors from literature. The results from the interviews indicate that the Critical Success Factors play an important role in the NTS project and several improvements have been identified. Keeping documentation up to date, multidisciplinairy teams to ensure easy communication and the learning curve of the new system are important factors mentioned.

*5 What business processes are involved in* KOT*?*
The relevant processes for *Toeslagen*, looking at KOT are: processing notifications, defaulters, residence factor, automatic continuation, decision on objection, appeal, mass supervision and final awarding. The most relevant business process was also identified: Processing notifications (*Verwerken meldingen*).

*6 Where is the system for* KOT *described?*
There are several levels on which the system is described. However, the Service Document [7] is regarded as the basis of the system.

*7 What specification language and tool is best suited for the modeling and verification of* KOT*?*
For this case, PROMELA and its tool SPIN are best suited for the modeling and verification of the system of KOT. This pairing is chosen due to the C-like structure of PROMELA and the maturity of SPIN.

*8 What level of abstraction is to be used for the modeling of the system supporting* KOT*?*
The current abstractions made in the case study, make it possible to see a working model of the behavior of the events. But the aspects of the Time Object Model (regarding the start and end time of data) is removed. This abstraction lowers the state space, but also reduces the funtionality of the model, as only the current situation of the citizen is stored. Time travelling is not possible. This has some influence of the validity of the model. For a model which contains full system behavior, a higher level of detail is needed.

*9 What characteristics should the specification of business processes or systems have to be suitable for model checking?*
The specification should be clear and unambigious, and should contain the abstract behavior of the system or business function.

*10 What kind of errors does model checking detect?*
The errors found by modeling and model checking are diverse. The errors found include ambiguity in the specification and incompleteness of the specification.

*11 To what extent does model checking improve the specification of the supporting systems?*
By using model checking ambiguity and incompleteness is removed, the working of the

system can be visualised and simulated. This helps designers to see what they have created and what issues can occur.

*12 What is the education level and knowledge needed for model checking?*
The needed education level and knowledge for model checking is high. Modeling a system requires a lot of knowledge of the architecture, domain and business logic of the services. For verification, detailed knowledge of the model checker, as well as Linear Temporal Logic is needed. This requires a higher or academic level of education.

*13 What are the general usability, costs, time intensity for model checking within the Toeslagen program at the Belastingdienst?*
Model checking can be used in the analysis and design phases of the development process, to specify system behavior. While experience from the case study has shown that modeling is a timely process, it does not seem to be more time consuming then the time needed for the creation of the documentation in the current format. As the system design has been extensively described in the specification, this has been a time consuming process as well. Using a formal language as PROMELA will therefore not increase the time by a relevant factor. However, the formal model can aid in the analysis of errors from the software, as well in the prevention of errors through verification.

*14 Does model checking provide added value to an organisation, taking into account costs and benefits?*
Considering the effort needed for a change request to be drawn up and implemented, the organisation can benefit from the model checking technique. Every prevented change request will save on the organisational costs, as these request require a lot of time for analysis and review which is expensive. Of course, not all errors due to incompleteness can be prevented, for example unknown features. But of known features the checking the completeness of the design via model checking is helpful. Furthermore, the technique can aid in the analysis of errors found in the system. And finally, using the formal specification language helps to clearify the specification documents.

*15 What view does model checking deliver of the supporting system of* KOT*?*
Model checking an abstract, formal model of the supporting system of KOT has shown that the specification contains several anomalies. However, these anomalies are relatively small compared to the system and were already known to the organisation. This shows that the supporting system of KOT is currently a stable system. This is shared by the organisation, as the number of problems that occur in the production system is low.

*16 Do stakeholders involved in* KOT *share the view delivered by model checking?*
Stakeholders share the findings from the case study. The specification is known to contain anomalies due to time shortage. However, these anomalies from the specification do not have an effect on the production software. Most problems have been eliminated during pre production and testing phases.

The analysis of Known Errors has shown that concurrency can indeed be the cause of the anomalies in the system of KOT. This means that the hypothesis that guided the case study: "Concurrency is the cause of the anomalies in KOT and model checking can detect

these anomalies in the design" upholds. Model checking could have prevented problems due to concurrency as currently occur within the communication between the AWIR and household service. As this anomaly has a wide range of potential occurrences (over 500,000 citizens involved the stated situation of the error), it is important to overcome these anomalies.

To sum up, and answer the main question, the following steps are required for a successful implementation of model checking in the development process of the *Belastingdienst*:

1. Working in multidisciplinairy teams with domain experts, architects en analysts

   Multidisciplinairy teams ensure easier communication. The domain experts are needed to embed specific knowledge in the specification of the system

2. Educate architects and analysts on the usage of the formal language and accompanying tools

   Detailed knowledge of the specification language, model checking tool and domain knowledge is needed for a successful usage of the technique. This requires architects and analysts to obtain knowledge over the language and tools.

3. Educate others involved in the basics of the formal language, so they can review the specification

   The review process within the development process ensures that the system is specified to behave as all users expect. To be able to review the specification, knowledge on the formal language is required

4. Educate domain experts in the drawing up of conditions that the specification should uphold. These conditions are verified by the model checker

   A model checker can verify stated conditions. This is a powerfull aspect of a model checker, but the language it requires is not an easy concept. Domain experts should be taught on this language, as they have the needed domain knowledge to specify the conditions

5. Schedule verification time, so the specification can be verified by the model checker. If problems are found during verification, their should be enough time scheduled to analyse and repair these problems

   Model checking can be a time consuming operation. But as it can aid in the detection of errors, it is important to verify the specification, before starting the software implementation of this specification

Further research is needed on quantification on the gain provided by model checking. It is clear that formal methods provide added value to the development process: once the long learning curve has passed, the system is specified in an unambiguous language, that

can be verified by model checking. This improves the system specification. The model checking tool can also aid in the analysis of software errors from the production phase, speeding up the analysis, which can also reduce the costs of the analysis.

# Bibliography

## — Internal documents —


[1] Somers, H. and van Rooyen, J. Model based testen - Pilot voor Definitief toekennen. Onderzoek opzet (in Dutch).

[2] *Belastingdienst*. Organisatiestructuur Belastingdienst/Centrale administratie, Jul 2011.

[3] *Belastingdienst/Toeslagen*. MLTP Toeslagen 2012-2015. Internal *Belastingdienst* document.

[4] Project Toeslagen 2009 and Foederer, R. Verzamelen raakvlakken, Sep 2007.

[5] *Belastingdienst*. BiZZdesigner Navigator Toeslagen Nieuw. B/CA intranet.

[6] Cluster IV, DGBel, Belastingdienst. Kaderdocument IV-keten - Strategie, structuur en inrichting van de IV-keten van de Belastingdienst. .

[7] Nissink, P. and Groot, M. de and Fernandez, S. and Maathuis, M. Services Toeslagen, versie 5.3, Jun 2011.

[8] Veenstra, Erik and others. Belastingdienst Toeslagen, TSL Software Architecture Document, 2010.

[9] Capgemini, A-team TSL. Bedrijfsfunctie bepaal AWIR-partnerschap, Nov 2007.

[10] Albert Chung, Sjoerd Perfors, Vincent Kappert, and Wim Wentink. Fo – bundelen services, 2011.

[11] Albert Chung, Sjoerd Perfors, Vincent Kappert, and Wim Wentink. To – bundelen services, 2011.

[12] Gert Veldhuijzen van Zanten and Erik Veenstra. Ontwerp samenvoegen services, 2011.

[13] B/CPP. Service Toeslagen (TSL) / Bedrijfsservice: Kinderopvangtoeslag, 2011.

## — Miscellaneous —


[14] Canfora, G. and Di Penta, M. Service-oriented architectures testing: A survey. *Software Engineering*, pages 78–105, 2009.

[15] Solstice Software, Robert Carmichael. Assuring Quality Business Processes through Service-Oriented Architecture Testing, 2006.

[16] Boehm, BW. Software engineering economics. *IEEE transactions on software engineering*, 10(1):4–21, 1984.

[17] Atlee, J.M. and Gannon, J. State-based model checking of event-driven system requirements. *Software Engineering, IEEE Transactions on*, 19(1):24–40, 1993.

[18] Bharadwaj, R. and Heitmeyer, C.L. Model checking complete requirements specifications using abstraction. *Automated Software Engineering*, 6(1):37–68, 1999.

[19] Bosma, T., van Rooyen, J., van der Zee, B. To prevent or to cure? In *Proceedings of the 19th International Conference on Software and Systems Engineering and their Applications*, 2006.

[20] Tretmans, J. Model based testing with labelled transition systems. *Formal methods and testing*, pages 1–38, 2008.

[21] Michelson, B.M. Event-driven architecture overview. *Patricia Seybold Group*, 2006.

[22] Laliwala, Z. and Chaudhary, S. Event-driven service-oriented architecture. In *Service Systems and Service Management, 2008 International Conference on*, pages 1–6. IEEE, 2008.

[23] Amálio, N. and Kelsen, P. and Ma, Q. The visual contract language: abstract modelling of software systems visually, formally and modularly. *Univ. of Luxembourg, Tech. Rep. TR-LASSY-10-01*, 2010.

[24] Miller, S.P. and Whalen, M.W. and Cofer, D.D. Software model checking takes off. *Communications of the ACM*, 53(2):58–64, 2010.

[25] Scheer, A.W. and Thomas, O. and Adam, O. Process Modeling using Event-Driven Process Chains. *Process-Aware Information Systems*, pages 119–145.

[26] Drury, C.G. Service, quality and human factors. *AI & Society*, 17(2):78–96, 2003.

[27] Murphy, C. Improving application quality using test-driven development (TDD). *Methods & Tools*, 13(1):2–17, 2005.

[28] Tretmans, J. A theory of model-based testing and how ioco goes eco. *Electronic Notes in Theoretical Computer Science*, 264(3):86–89, 2010.

[29] Muccini, H. *Software architecture for testing, coordination and views model checking*. PhD thesis, Universta degli Studi di Roma 'La Sapienza', 2002.

[30] Clarke, E.M. and Wing, J.M. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[31] Ostroff, J.S. Formal methods for the specification and design of real-time safety critical systems. *Journal of Systems and Software*, 18(1):33–60, 1992.

[32] Lin, F.J. and Chu, PM and Liu, M.T. Protocol verification using reachability analysis: the state space explosion problem and relief strategies. *ACM SIGCOMM Computer Communication Review*, 17(5):126–135, 1987.

[33] Alur, R. and Brayton, R. and Henzinger, T. and Qadeer, S. and Rajamani, S. Partial-order reduction in symbolic state space exploration. In *Computer Aided Verification*, pages 340–351. Springer, 1997.

[34] El-Far, I.K. and Whittaker, J.A. Model-Based Software Testing. *Encyclopedia of Software Engineering*, 2001.

[35] Pelánek, R. Fighting state space explosion: Review and evaluation. *Formal Methods for Industrial Critical Systems*, pages 37–52, 2009.

[36] Hall, A. Realising the benefits of formal methods. *Formal Methods and Software Engineering*, pages 1–4, 2005.

[37] Tweede Kamer der Staten-Generaal. Kamerstuk Tweede Kamer, vergaderjaar 2010–2011, 31 322, nr. 123. `https://zoek.officielebekendmakingen.nl/kst-31322-123.pdf`.

[38] Larsen, P.G. and Fitzgerald, J. and Brookes, T. Applying formal specification in industry. *Software, IEEE*, 13(3):48–56, 1996.

[39] Holzmann, G.J. The model checker SPIN. *Software Engineering, IEEE Transactions on*, 23(5):279–295, 1997.

[40] Holzmann, G.J. and Smith, M.H. A practical method for verifying event-driven software. In *Proceedings of the 21st international conference on Software engineering*, pages 597–607. ACM, 1999.

[41] Belastingdienst. Basic Values. `http://www.belastingdienst.nl/wps/wcm/connect/bldcontenten/standaard_functies/individuals/organisation/basic_values/`.

[42] Belastingdienst. Kamerstuk Tweede Kamer, Vergaderjaar 2011-2012, 33000-IXB nr. 24, Bijlage bij Kamerstuk 33000-IXB nr. 24, Beheerverslag Belastingdienst 2011. `https://zoek.officielebekendmakingen.nl/blg-168696.pdf`.

[43] Tweede Kamer der Staten-Generaal. Kamerstuk II, Vergaderjaar 2008-2009, 31066 nr. 61. https://zoek.officielebekendmakingen.nl/kst-31066-61.pdf.

[44] Tweede Kamer der Staten-Generaal. Kamerstuk II, Vergaderjaar 2008-2009, 31066 nr. 64. https://zoek.officielebekendmakingen.nl/kst-31066-64.pdf.

[45] Tweede Kamer der Staten-Generaal. Kamerstuk II, Vergaderjaar 2009-2010, 31066 nr. 78. https://zoek.officielebekendmakingen.nl/kst-31066-78.pdf.

[46] Tweede Kamer der Staten-Generaal. Bijlage bij Kamerstuk II - Vijfde halfjaarsrapportage vereenvoudigingsoperatie Belastingdienst, Vergaderjaar 2009-2010, 31066 nr. 82. https://zoek.officielebekendmakingen.nl/blg-47836.pdf.

[47] Tweede Kamer der Staten-Generaal. Bijlage bij Kamerstuk II - Halfjaarsrapportage Belastingdienst, Vergaderjaar 2009-2010, 31066 nr. 90. https://zoek.officielebekendmakingen.nl/blg-70107.pdf.

[48] Tweede Kamer der Staten-Generaal. Bijlage bij Kamerstuk II - Halfjaarsrapportage Belastingdienst november 2009, Vergaderjaar 2010-2011, 31066 nr. 98 Herdruk. https://zoek.officielebekendmakingen.nl/kst-31066-102.pdf.

[49] Tweede Kamer der Staten-Generaal. Kamerstuk II, Vergaderjaar 2010-2011, 31066 nr. 102. https://zoek.officielebekendmakingen.nl/kst-31066-102.pdf.

[50] Tweede Kamer der Staten-Generaal. Bijlage bij Kamerstuk II - $8^e$ Halfjaarsrapportage Belastingdienst mei 2011, Vergaderjaar 2010-2011, 31066 nr. 103. https://zoek.officielebekendmakingen.nl/blg-116908.pdf.

[51] Ministerie van Binnenlandse Zaken en Koninkrijksrelaties. Kamerstuk Tweede Kamer, vergaderjaar 2011-2012, 41490, nr. 88 - Rapportage grote en risicovolle ICT-projecten, Bijlage bij de Jaarrapportage Bedrijfsvoering Rijk 2011. https://zoek.officielebekendmakingen.nl/blg-167549.pdf.

[52] Ministerie van Binnenlandse Zaken en Koninkrijksrelaties. Toeslagen Nieuw | Rijks ICT dashboard. https://www.rijksictdashboard.nl/content/project/toeslagen-nieuw.

[53] Tweede Kamer der Staten-Generaal. Kamerstuk Tweede Kamer, vergaderjaar 2010–2011, 31 066, nr. 106. https://zoek.officielebekendmakingen.nl/kst-31066-106.pdf.

[54] Maréchaux, J.L. Combining service-oriented architecture and event-driven architecture using an enterprise service bus. *IBM Developer Works*, 2006.

[55] Martijnse, N. en Noordam, P. . Projectmanagement: lessen uit falende en succesvolle ICT-projecten. Controllers moeten vanaf de start een rol spelen. *Management Control en Accounting*, (3), April 2007.

[56] Kloppenborg, T.J. and Opfer, W.A. The current state of project management research: Trends, interpretations, and predictions. *Project Management Journal*, 33(2):5–18, 2002.

[57] Rockart, J.F. Chief executives define their own data needs. *Harvard business review*, 57(2):81, 1979.

[58] Pinto, J.K. and Slevin, D.P. Critical factors in successful project implementation. *IEEE transactions of engineering management*, (1):22–27, 1987.

[59] Pinto, J.K. and Prescott, J.E. Variations in critical success factors over the stages in the project life cycle. *Journal of management*, 14(1):5–18, 1988.

[60] Sumner, M. Critical success factors in enterprise wide information management systems projects. In *Proceedings of the 1999 ACM SIGCPR conference on Computer personnel research*, pages 297–303. ACM, 1999.

[61] El Emam, K. and Koru, A.G. A replicated survey of IT software project failures. *Software, IEEE*, 25(5):84–90, 2008.

[62] Rosacker, K.M. and Olson, D.L. Public sector information system critical success factors. *Transforming Government: People, Process and Policy*, 2(1):60–70, 2008.

[63] Shokri-Ghasabeh, M. and Kavoousi-Chabok, K. Generic project success and project management success criteria and factors: Literature review and survey. *WSEAS Transactions on business and economics*, 6(8):456–468, 2009.

[64] Pankratz, O. and Loebbecke, C. Project managers'perception of is project success factors–a repertory grid investigation. 2011.

[65] Verhoef, C. Politieke deadlines: dodelijk voor IT. *Digitaal bestuur*, January 2007.

[66] Groep, P. and Beenker, N. Studie naar succes-en faalfactoren van complexe ICT projecten.

[67] Rampersad, H.K. Total Quality Management; an executive guide to continuous im-provement. 2001.

[68] Roberts, M. *Readings in Total Quality Management*, chapter 30, pages 459–473. Dryden Press, 2 edition, 1999. Becoming customer oriented By Mary Lou Roberts.

[69] Cleaveland, R. and Parrow, J. and Steffen, B. The Concurrency Workbench: A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(1):36–72, 1993.

[70] Sipma, H. and Uribe, T. and Manna, Z. Deductive model checking. In *Computer Aided Verification*, pages 208–219. Springer, 1996.

[71] Mateescu, R. and Sighireanu, M. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *Science of Computer Programming*, 46(3):255–281, 2003.

[72] Fehnker, A. and Huuck, R. and Jayet, P. and Lussenburg, M. and Rauch, F. Goanna—a static model checker. *Formal Methods: Applications and Technology*, pages 297–300, 2007.

[73] Yovine, S. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):123–133, 1997.

[74] Jones, A.V. and Lomuscio, A. A BDD-based BMC approach for the verification of multi-agent systems. *Organizing and Program Committee*, page 253, 2009.

[75] Groote, J.F. and Mathijssen, A. and Reniers, M. and Usenko, Y. and Van Weerdenburg, M. The formal specification language mCRL2. *Methods for Modelling Software Systems (MMOSS)*, 6351, 2007.

[76] Griffault, A. and Vincent, A. The Mec 5 model-checker. In *Computer Aided Verification*, pages 248–251. Springer, 2004.

[77] Dill, D. The Mur$\phi$ verification system. In *Computer Aided Verification*, pages 390–393. Springer, 1996.

[78] Cimatti, A. and Clarke, E. and Giunchiglia, E. and Giunchiglia, F. and Pistore, M. and Roveri, M. and Sebastiani, R. and Tacchella, A. Nusmv 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, pages 241–268. Springer, 2002.

[79] Kwiatkowska, M. and Norman, G. and Parker, D. PRISM: Probabilistic symbolic model checker. *Computer Performance Evaluation: Modelling Techniques and Tools*, pages 113–140, 2002.

[80] Shankar, N. Combining theorem proving and model checking through symbolic analysis. *CONCUR 2000—Concurrency Theory*, pages 1–16, 2000.

[81] Dräger, K. and Kupriyanov, A. and Finkbeiner, B. and Wehrheim, H. SLAB: A certifying model checker for infinite-state concurrent systems. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 271–274, 2010.

[82] Bengtsson, J. and Larsen, K. and Larsson, F. and Pettersson, P. and Yi, W. UPPAAL—a tool suite for automatic verification of real-time systems. *Hybrid Systems III*, pages 232–243, 1996.

[83] Hoffmann, V. and Lichter, H. and Nyßen, A. Processes and Practices for Quality Scientific Software Projects. In *Proceedings of 3rd International Workshop on Academic Software Development Tools WASDeTT-3*, pages 95–108, 2010.

[84] Holzmann, G. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[85] Holzmann, G.J. and Bosnacki, D. Multi-core model checking with SPIN. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8. IEEE, 2007.

[86] Holzmann, G.J. and Joshi, R. and Groce, A. Swarm verification. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 1–6. IEEE Computer Society, 2008.

[87] Hendriks, M. and Behrmann, G. and Larsen, K. and Niebert, P. and Vaandrager, F. Adding symmetry reduction to uppaal. *Formal Modeling and Analysis of Timed Systems*, pages 46–59, 2004.

[88] Behrmann, G. and Bengtsson, J. and David, A. and Larsen, K. and Pettersson, P. and Yi, W. Uppaal implementation secrets. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 3–22. Springer, 2002.

[89] Stern, U. and Dill, D.L. Combining state space caching and hash compaction. *Methoden des Entwurfs und der Verifikation digitaler Systeme*, 4:81–90, 1996.

[90] Norris Ip, C. and Dill, D.L. Better verification through symmetry. *Formal methods in system design*, 9(1):41–75, 1996.

[91] Ip, C.N. and Dill, D.L. State reduction using reversible rules. In *Proceedings of the 33rd annual Design Automation Conference*, pages 564–567. ACM, 1996.

[92] Ip, C. and Dill, D. Verifying systems with replicated components in Mur$\phi$. In *Computer aided verification*, pages 147–158. Springer, 1996.

[93] Stern, U. and Dill, D. Parallelizing the Mur$\phi$ verifier. In *Computer Aided Verification*, pages 256–267. Springer, 1997.

[94] Ruys, T. Low-fat recipes for SPIN. *SPIN Model Checking and Software Verification*, pages 287–321, 2000.

[95] Ben-Ari, M. *Principles of the Spin model checker*. Springer-Verlag New York Inc, 2008.

[96] Holzmann, G.J. Design and validation of computer protocols. 1991.

[97] Holzmann, G.J. PAN: a protocol specification analyzer. Technical report, Technical Report TM81-11271-5, AT&T Bell Laboratories, 1981.

[98] Holzmann, G. and Joshi, R. and Groce, A. Swarm verification techniques. *Software Engineering, IEEE Transactions on*, (99):1–1, 2010.

[99] Alastair F. Donaldson and Alice Miller. A computational group theoretic symmetry reduction package for the SPIN model checker. In *Proceedings of the 11th International Conference on Algebraic Methodology and Software Technology (AMAST'06)*, volume 4019 of *Lecture Notes in Computer Science*, pages 374–380. Springer, 2006.

[100] D. Bošnački and D. Dams. Discrete-time promela and spin. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 307–310. Springer, 1998.

[101] T. Basten, D. Bošnački, and M. Geilen. Cluster-based partial-order reduction. *Automated Software Engineering*, 11(4):365–402, 2004.

[102] S. Blom, J. van de Pol, and M. Weber. Ltsmin: Distributed and symbolic reachability. In *Computer Aided Verification*, pages 354–359. Springer, 2010.

[103] M. de Jonge and T. Ruys. The spinja model checker. *Model Checking Software*, pages 124–128, 2010.

[104] W. Visser and H. Barringer. Ctl* model checking for spin. *Software Tools for Technology Transfer, LNCS*, 1999.

[105] Ruys, T.C. *Towards effective model checking.* PhD thesis, Universiteit Twente, 2001.

[106] R.S. Pressman. *Software engineering: a practitioner's approach.* McGraw-Hill Science/Engineering/Math, 2010.

[107] B Lamers. Een functionele aanpak voor taalcreatie & transformatie. Master's thesis, Radboud University Nijmegen, 2009.

[108] Albert Gerritsen. Functional debugging. Master's thesis, Radboud University Nijmegen, 2011.

[109] P. Kars. The application of promela and spin in the bos project. In *The Spin Verification System: The Second Workshop on the SPIN Verification System: Proceedings of a DIMACS Workshop, August*, volume 5, page 51, 1996.

[110] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[111] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220. ACM, 2009.

[112] P.E. Ammann, P.E. Black, and W. Majurski. Using model checking to generate tests from specifications. In *Formal Engineering Methods, 1998. Proceedings. Second International Conference on*, pages 46–54. IEEE, 1998.

[113] R. Alexander. Deployment of formal methods in industry: the legacy of the fp7 ict deploy integrated project. *ACM SIGSOFT Software Engineering Notes*, 5, 2012.

[114] R. Calinescu, S. Kikuchi, and M. Kwiatkowska. Formal methods for the development and verification of autonomic it systems. *Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development and Verification. IGI Global (to appear, 2011)*, 2011.

[115] M. Mazzolini, A. Brusaferri, and E. Carpanzano. Model-checking based verification approach for advanced industrial automation solutions. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8. IEEE, 2010.

# List of Figures

# List of Tables

# List of code listings

# Lists of abbreviations

| | |
|---|---|
| AB | Architecture board |
| AWIR | *Algemene Wet Inkomensafhankelijke Regelingen* |
| B/CA | *Belastingdienst/Centrale Administratie* |
| B/CAO | *Belastingdienst/Centrum voor Applicatie-ontwikkeling en onderhoud* |
| B/CFD | *Belastingdienst/Centrum voor Facilitaire Dienstverlening* |
| B/CKC | *Belastingdienst/Centrum voor Kennis en Communicatie* |
| B/CIE | *Belastingdienst/Centrum voor Infrastructuur en Exploitatie* |
| B/CIO | *Belastingdienst/Centrum voor IV-keten ondersteuning* |
| B/T | *Belastingdienst/Toeslagen* |
| BD | *Belastingdienst* |
| BOAB | *Bedrijfsonderdeel architectuur board* (Unit architecture board) |
| CSF | Critical Success Factors |
| EDA | Event-Driven Architecture |
| ESB | Enterprise Service Bus |
| FIOD | Fiscale Inlichtingen- en OpsporingsDienst |
| FTO | *Funtionele Test Omgeving* |
| FTE | FullTime-Equivalent |

| | |
|---|---|
| FRS | *Feiten Registratie Systeem*, component of NTS |
| GBB | *Grondslagen, Beslissen en Beschikken* |
| GO | *Globaal ontwerp* (Global Design) |
| ICT | Information and Communication Technology |
| IM | Information Management |
| ISB | Information Service Board |
| KGB | *Kindgebonden budget* |
| KOT | *Kinderopvangtoeslag* |
| MBT | Model Based Testing |
| MC | Model Checking |
| MTHV | *Methoden, Technieken, Hulpmiddelen en Voorschriften* |
| NGIV | *Niet-Geautomatiseerde InformatieVoorziening* |
| PB | Portfolio Board |
| PRA | Product Risk Analysis |
| PRINCE2 | PRojects IN Controlled Environments 2 |
| NTS | *Nieuwe Toeslagen Systeem* |
| SOA | Service Oriented Architecture |
| SDLC | Software Development Life Cycle |
| SMT | Satisfiability Modulo Theories |
| SVB | Sociale Verzekerings Bank |
| TMAP | Test Management Approach |
| TSL | *Toeslagen*, component of NTS |

VTA *Verbeterde Test Aanpak* (Improved Test Approach)

WV(S) *Wijzigingsvoorstel(len)* (Proposal(s) for change)

# Appendix A

# Business process “process notifications”

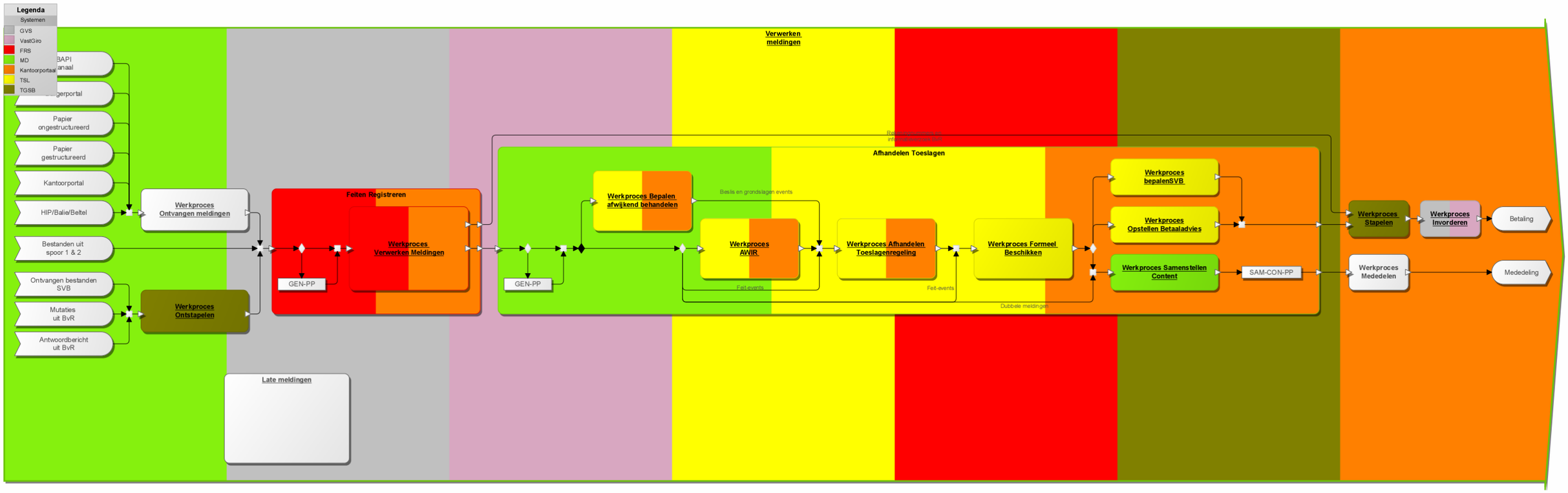


Figure A.1: Business process Process notifications [5]

Appendix B

# *Toeslagen* application architecture

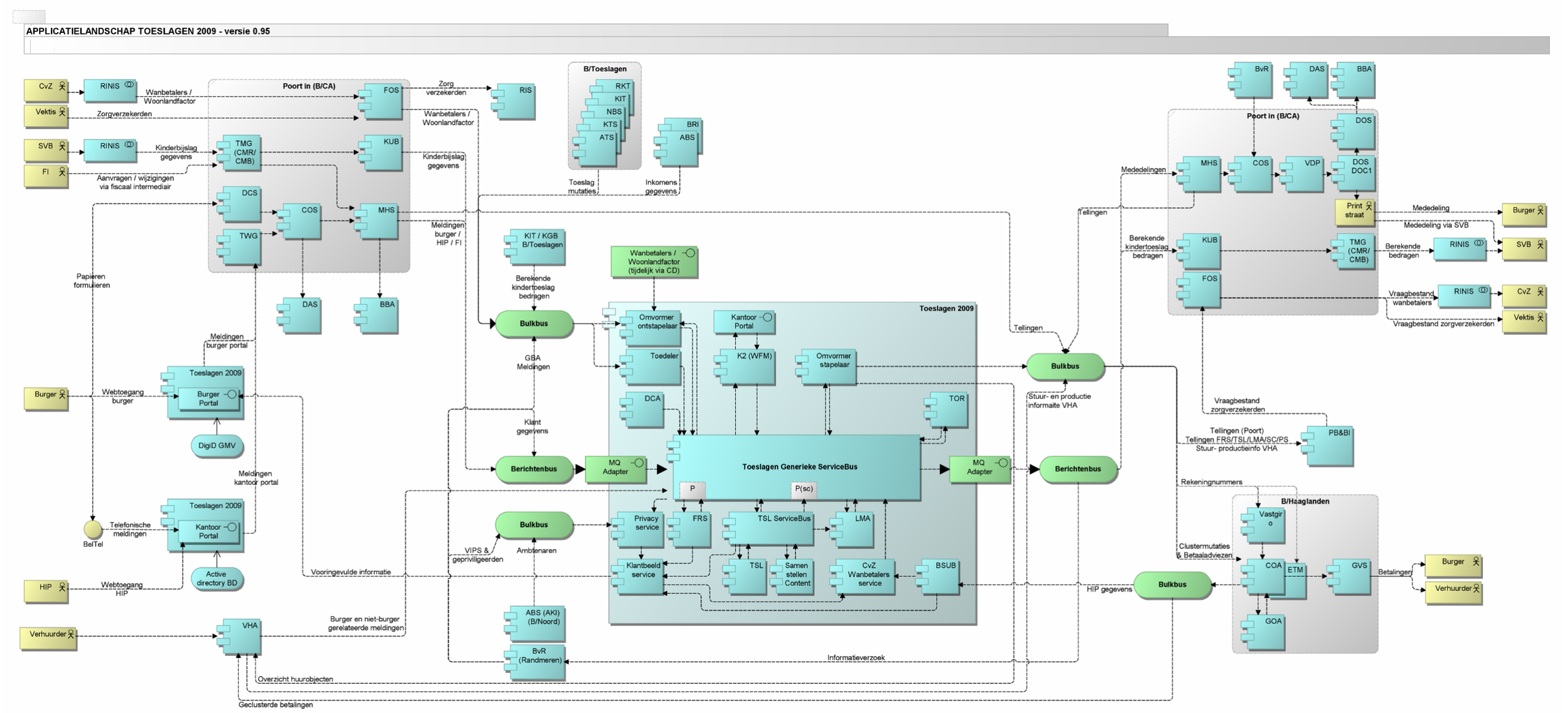


Figure B.1: *Toeslagen* application architecture [5]

# Appendix C

# Processing notifications

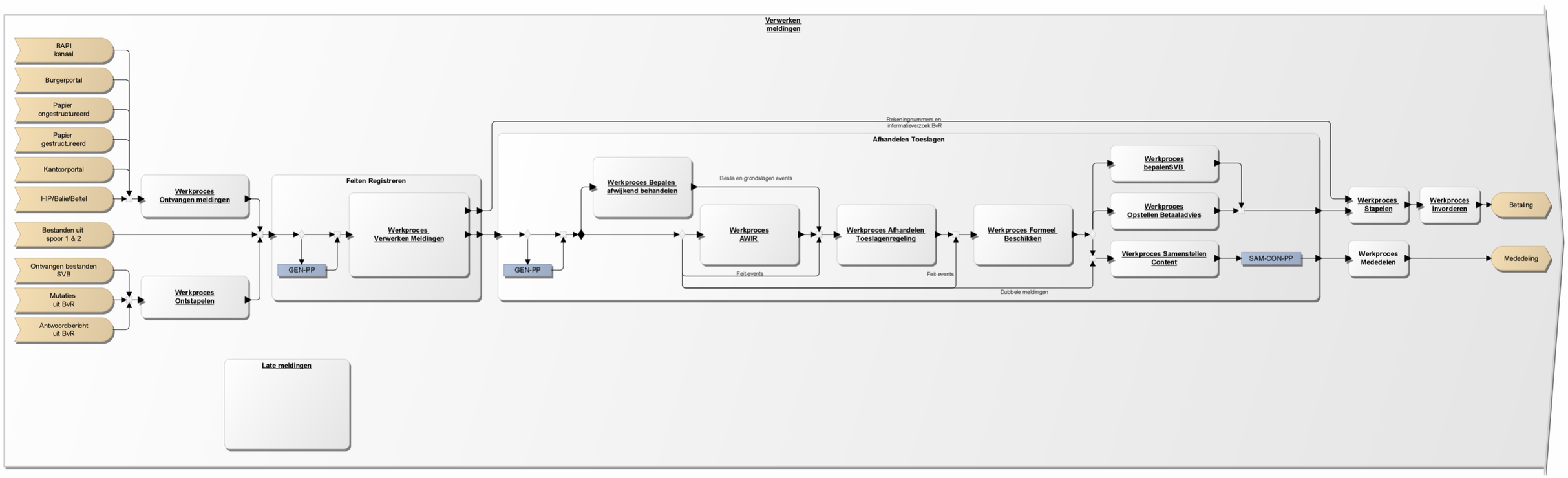


Figure C.1: Business process Processing notifications [5]

Appendix D

# Workprocess handle benefits regulations

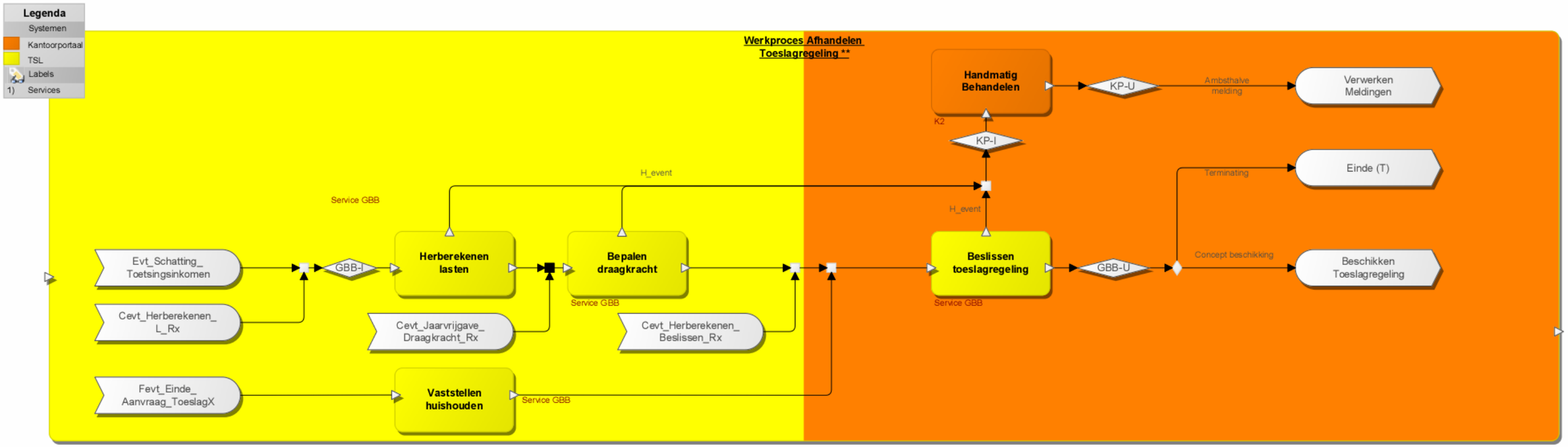


Figure D.1: Workprocess handle benefits regulations [5]

# Appendix E

# Development process at *Belastingdienst*

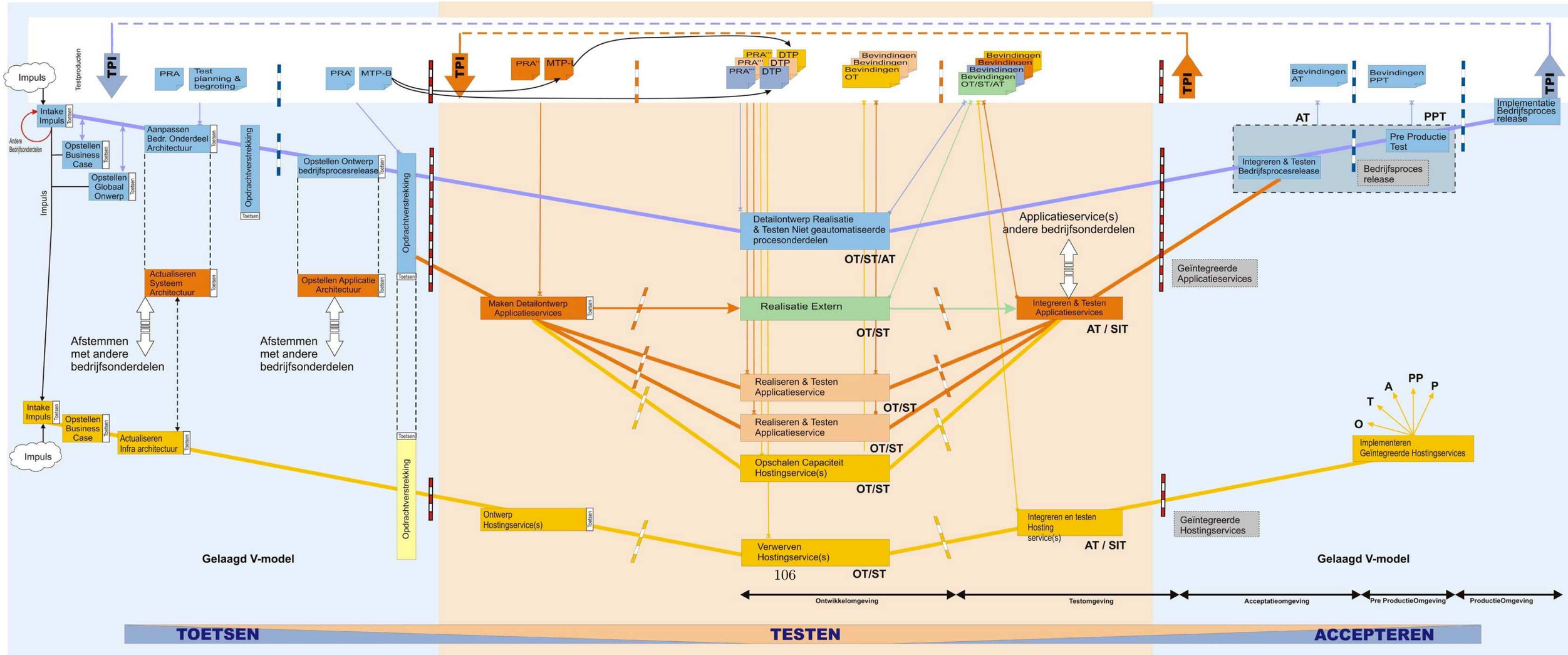




Figure E.1: Development process at *Belastingdienst* (V-model) [6]

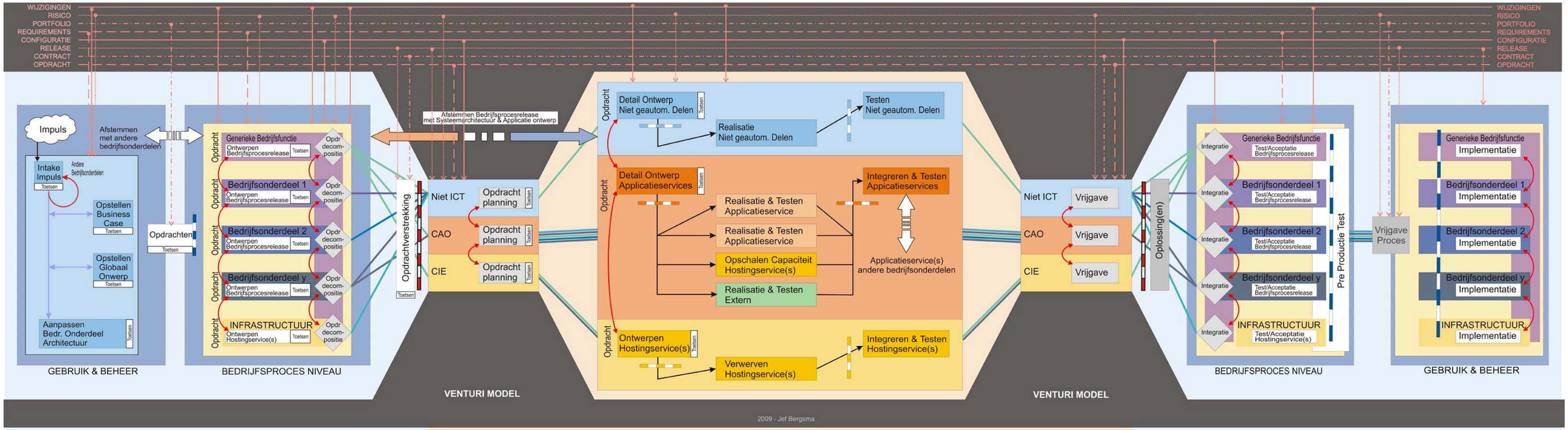


Figure E.2: Development process at *Belastingdienst* (Venturi model) [6]

# Appendix F

# RASCI table

| | IM | B/CAO | B/CIE | B/CFD | B/CKC | DGBEL | MT BD | Unit MT | Unit AB | Unit PB | AB BD | Units* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Intake impulse | RA | R | R | | | C | | | A | | | C |
| Drafting outline business case | R | R | R | | | C | A | A | I | | | |
| Drafting global design | R | R | C | | | C | A | A | | | I | |
| Update business architecture components and task portfolio | R | R | | | | | A | A | C | C | | |
| Update corporate architecture and corporate task portfolio | | | | | | R | RA | | | | | R |
| Draft design | RA | RAC | RC | | C | | | | C | C | | C |
| Draft detailed design, realise and test automated information services | RA | RA | R | | | | | | | | | |
| Upscaling hosting capacity | | | RA | R | | | | | | | | |
| Draft detailed design, realise and test non automated process components | A | | R | R | R | | | | | | | R |
| Test business process | AR | R | A | | | | | | | | | R |
| Implement operational services | | | R | | | | | A | | | | R |
| Implement business process release | R | R | R | | R | | | | | | | AR |
| Evaluate | R | R | R | | R | | | A | R | R | | R |

* Units on which the initial impulse has impact

Table F.1: Responsibility assignment matrix of the development process of the *Belastingdienst*

# Appendix G

# Distribution of respondents

| Phase | Role | Code |
|---|---|---|
| Intake impulse | Tactical architect | 002 |
| | Programme Manager | 003 |
| | Implementation manager | 004 |
| | Incident manager | 008 |
| | Strategic architect | 009 |
| | Quality monitor | 011 |
| | Service manager | 012 |
| | Consultant primary process | 013 |
| | Implementation manager | 015 |
| | Team leader development team | 016 |
| | General Programme Manager | 020 |
| Drafting outline business case | Tactical architect | 002 |
| | Programme Manager | 003 |
| | Strategic architect | 009 |
| | Quality monitor | 011 |
| | Consultant primary process | 013 |
| | Team leader development team | 016 |
| Drafting global design | Tactical architect | 002 |
| | Implementation manager | 004 |
| | Project & ICT architect | 007 |
| | Strategic architect | 009 |
| | Quality monitor | 011 |
| | Consultant primary process | 013 |
| | Team leader development team | 016 |
| Update business architecture components and task portfolio | Tactical architect | 002 |
| | Project & ICT architect | 007 |
| | Strategic architect | 009 |

| | | |
|---|---|---|
| | Consultant primary process | 013 |
| Update corporate architecture and corporate task portfolio | Tactical architect | 002 |
| | Strategic architect | 009 |
| | Consultant primary process | 013 |
| Draft design | Tactical architect | 002 |
| | Programme Manager | 003 |
| | Implementation manager | 004 |
| | Incident manager | 008 |
| | Strategic architect | 009 |
| | Quality monitor | 011 |
| | Team leader development team | 016 |
| Draft detailed design, realise and test automated information services | Quality monitor | 001 |
| | Tactical architect | 002 |
| | Project & ICT architect | 007 |
| | Strategic architect | 009 |
| | Quality monitor | 011 |
| | Team leader development team | 016 |
| | Functional analyst | 017 |
| | Lead architect TSL | 018 |
| Upscaling hosting capacity | Quality monitor | 001 |
| Draft detailed design, realise and test non automated information process components | Quality monitor | 001 |
| | Tactical architect | 002 |
| | Incident manager | 008 |
| | Strategic architect | 009 |
| | Quality monitor | 011 |
| | Analyst | 014 |
| Test business process | Quality monitor | 001 |
| | Project manager test | 010 |
| | Quality monitor | 011 |
| | Test co-ordinator and manager | 019 |
| Implement operational services | Quality monitor | 001 |
| | Programme Manager | 003 |
| | Chain co-ordinator | 005 |
| | Project & ICT architect | 007 |
| | Project manager test | 010 |
| | Quality monitor | 011 |
| | Service manager | 012 |
| | Analyst | 014 |
| | Implementation manager | 015 |
| | Test co-ordinator and manager | 019 |
| Implement business process release | Quality monitor | 001 |
| | Programme Manager | 003 |
| | Implementation Manager B/CA | 004 |

| | | |
|---|---|---|
| | Chain co-ordinator | 005 |
| | Process designer | 006 |
| | Project & ICT architect | 007 |
| | Project manager test | 010 |
| | Quality monitor | 011 |
| | Service manager | 012 |
| | Analyst | 014 |
| | Implementation manager | 015 |
| | Test co-ordinator and manager | 019 |
| Evaluate | Quality monitor | 001 |
| | Programme Manager | 003 |
| | Implementation Manager B/CA | 004 |
| | Chain co-ordinator | 005 |
| | Quality monitor | 011 |
| | Implementation manager | 015 |

Table G.1: Coverage of respondents over development process

| **English** | ***Dutch*** |
|---|---|
| Analyst | *Analist* |
| Chain co-ordinator | *Ketenregisseur* |
| Consultant primary process | *Consultant primair proces* |
| Functional analyst | *Functioneel analist* |
| Project & ICT architect | Project & ICT architect |
| Incident manager | *Incidentmanager* |
| Implementation manager | *Implementatiemanager* |
| Implementation manager b/ca | *Implementatiemanager* |
| Lead architect TSL | *Lead architect* TSL |
| Programme Manager | *Programmamanager* |
| Project manager test | *Projectmanager test* |
| Quality Monitor | *Kwaliteitsmonitor* |
| Process designer | *Procesontwerper* |
| Service manager | *Servicemanager* |
| Strategic architect | *Strategisch architect* |
| Tactical architect | *Tactisch architect* |
| Team leader development team | *Teamleider ontwikkelteam* |

Table G.2: Translation of functions in development process

# Appendix H

# List of questions

## H.1 Betrokkenheid bij Toeslagen

1. Sinds wanneer ben u betrokken bij het programma Toeslagen?
2. Welke rol vervult u binnen het programma Toeslagen?
3. Wat houdt deze rol in?
4. In welke stap of stappen uit het V-model zou u zichzelf plaatsen? *(Toon model aan geïnterviewde)*
5. In welke perioden was dit? *(Alleen in geval van meerdere stappen uit model/functies binnen het programma Toeslagen)*

## H.2 Ontwikkelproces

1. Kunt u het ontwikkelproces binnen de Belastingdienst beschrijven, terugkijkend naar het programma Toeslagen? (*Identificeerbare stappen in het proces, slagbomen, voortgangsrapportages, V-model bekend?)*
2. Wat is uw ervaring met dit ontwikkelproces binnen de Belastingdienst? (*Doorvragen naar ervaringen/voordelen/nadelen*)
3. Wat kan er verbeterd worden aan dit ontwikkelproces?
4. Welke type fouten bent u tegengekomen die zijn ontstaan tijdens het ontwikkelproces? (*Eventueel doorvragen naar kennis genomen van fouten die niet zelf is tegengekomen*)
5. Op welke manier hadden deze fouten voorkomen kunnen worden?

## H.3 Kwaliteit

1. Wat vindt u van de kwaliteit die in de verschillende fasen van het ontwikkelproces geleverd is en wordt?
2. Welke methodes of technieken kunnen deze kwaliteit verbeteren?

## H.4 Testen

1. Welke testtechnieken worden er binnen de Belastingdienst, kijkend naar het programma Toeslagen, toegepast?
2. Wat vindt u van deze testtechnieken? (*Dekkingsgraad, voldoen de technieken?*)
3. Op welk niveaus in het ontwikkelproces wordt er getest? (*Toon V-model wederom, hierin staan de stappen c.q. niveaus*)
4. Is de huidige manier van testen voldoende om de gevraagde kwaliteit te leveren?
5. Wat kan hier aan veranderd worden?

## H.5 Impact

1. Wat was de impact van de gevonden fouten op de bedrijfsvoering? *(extern/intern)*
2. Welke maatregelen zijn er genomen om deze fouten op te lossen?
3. Wat wordt er gedaan om gelijksoortige fouten te voorkomen?

## H.6 Veranderingen

1. Welke veranderingen hebben er de afgelopen jaren plaatsgevonden in het ontwikkelproces? (*Extra stappen in V-model, slagbomen, testtechnieken, VTA?*)
2. Op welke wijze zijn deze veranderingen doorgevoerd in het ontwikkelproces? (*Tijdsduur, opgelegd/inspraak*)
3. Wat had hier beter in gekund?
4. Terugkijkend naar deze veranderingen, de wijze waarop deze zijn doorgevoerd in het ontwikkelproces en uw eigen ervaringen hiermee, op welke wijze denkt u dat veranderingen in het ontwikkelproces het best en meest eenvoudig ingevoerd kunnen worden?

# Appendix I

# Transcripts

The transcripts of the interviews have been marked as confidential, and have been removed from this report.

# Appendix J

# Model

## J.1 Events

```
// List of all events
/* Internally, the values of the mtype are represented as positive byte
    values,
  so there can be at most 255 values of the type.
*/
mtype = {
    // start FRS
    // Aanvang_partnerschap, // FRS melding 3
    Fevt_start_geregistreerd_partnerschap, // Start events following FRS
        melding 3
    Fevt_start_samenlevingscontract,
    Fevt_handtekeningrelatie_lopend_partner, // End events following FRS
        melding 3

    //Beeindiging_partnerschap, // FRS melding 4
    Fevt_einde_geregistreerd_partnerschap, // Event following FRS
        melding 4
    Fevt_einde_samenlevingscontract, // Event following FRS melding 4

    //Burger_overleden, // FRS melding 5
    Fevt_overlijden, // Events following FRS melding 5
    //Fevt_eind_geregistreerd_partnerschap, // (see FRS melding 4)
    //Fevt_eind_samenlevingscontract, // (see FRS melding 4)

    //Burger_verhuist, // FRS melding 6
    Fevt_burger_inschrijven_op_adres, // Event following FRS melding 6
    Fevt_meerdere_burgers_inschrijven_op_adres, // Event following FRS
        melding 6

    //Verblijfsstatus_burger_verandert, // FRS melding 7
    Fevt_verblijfsstatus, // Event following FRS melding 7
    Fevt_einde_verblijfstitel, // Event following FRS melding 7
```

```
        //Burger_vraagt_toeslag_aan , // FRS Melding 8
        Fevt_aanvraag_kinderopvangtoeslag ,
        Fevt_handtekeningrelatie_medebewoner , // Events following FRS
            melding 8
        //Burger_zet_toeslag_stop , // FRS melding 9
        Fevt_einde_aanvraag_kinderopvangtoeslag , // Event following FRS
            melding 9

        //Burger_gaat_weer_naar_school , // FRS melding 12
        Fevt_burger_gaat_naar_school , // Event following FRS melding 12

        //Burger_gaat_niet_meer_naar_school , // FRS melding 13
        Fevt_burger_gaat_niet_meer_naar_school , // Event following FRS
            melding 13

        //Burger_ontvangt_inkomsten_uit_werk , // FRS melding 16
        Fevt_burger_ontvangt_inkomsten_uit_werk , // Event following FRS
            melding 16

        //Burger_ontvangt_geen_inkomsten_meer_uit_werk , // FRS melding 17
        Fevt_burger_ontvangt_geen_inkomsten_uit_werk , // Event following FRS
            melding 17

        //Gegevens_kinderopvang_worden_opgegeven , // FRS melding 18
        Fevt_kinderopvanggebruik , // Event following FRS melding 18

        // TODO source??
        Evt_lasten_kinderopvangtoeslag ,
        Evt_normen_kinderopvangtoeslag ,
        Cevt_dataset_proefberekening_kinderopvangtoeslag ,
        Bevt_beslissing_kinderopvangtoeslag ,

        //Wijziging_ouder_kind_relatie , // FRS melding 20
        Fevt_relatie_ouder_kind_wijzigt , // Event following FRS melding 20
        Fevt_einde_ouder_kind_relatie , // Event following FRS melding 20

        //Burger_wordt_gedetineerd , // FRS melding 27
        Fevt_burger_gedetineerd , // Event following FRS melding 27
        //Burger_is_niet_meer_gedetineerd // FRS melding 28
        Fevt_burger_niet_gedetineerd , // Event following FRS melding 28

        //Inkomen_voor_burger_is_opnieuw_vastgesteld , // FRS melding 48
        Fevt_verzamelinkomen , // Start Events following FRS melding 48
        Fevt_fiscaal_jaarloon ,
        Fevt_NINBI ,
        Fevt_beschreven_IB ,
        Fevt_beschreven_NINBI ,
        Fevt_niet_beschreven_IB ,
        Fevt_niet_beschreven_NINBI , // End Events following FRS melding 48

        //Burger_meldt_inkomenswijziging , // FRS melding 49
        Fevt_geschat_toetsingsinkomen , // Event following FRS melding 49

        //Burger_krijgt_uitstel_IB , // FRS melding 52
```

```
    Fevt_burger_heeft_uitstel_IB, // Event following FRS melding 52

80
    //Burger_krijgt_fiscaal_partner, // FRS melding 53
    //Fevt_start_fiscaal_partnerschap, // Event following FRS melding 53
    Fevt_start_fiscaal_partner,

85
    //Burger_beeindigt_fiscaal_partnerschap,  // FRS melding 54
    //Fevt_einde_fiscaal_partnerschap, // Event following FRS melding 54
    Fevt_einde_fiscaal_partner,

90
    //Burger_verjaart, // Fake, non-existing event
    //Fevt_verjaring_x_jaar, // Event following FRS melding 59
        (non-existent)
    Fevt_verjaring_5_jaar,
    Fevt_verjaring_13_jaar,
95  Fevt_verjaring_18_jaar,


    //Aanvang_duurzaam_gescheiden_partnerschap, // FRS melding 65
    Fevt_aanvang_duurzaam_gescheiden_partnerschap, // Event following
        FRS melding 65
100
    //Einde_duurzaam_gescheiden_partnerschap, // FRS melding 66
    Fevt_einde_duurzaam_gescheiden_partnerschap, // Event following FRS
        melding 66


105 //Start_gezamenlijke_schuld, // FRS melding 67
    Fevt_start_gezamenlijke_schuld, // Event following FRS melding 67

    //Einde_gezamenlijke_schuld, // FRS melding 68
    Fevt_einde_gezamenlijke_schuld, // Event following FRS melding 68
110
    //Start_partners_in_pensioenregeling, // FRS melding 69
    Fevt_start_partners_in_pensioenregeling, // Event following FRS
        melding 69

    //Einde_partners_in_pensioenregeling, // FRS melding 70
115 Fevt_einde_partners_in_pensioenregeling, // Event following FRS
        melding 70

    //Start_gezamenlijk_huishouden, // FRS melding 71
    Fevt_start_gezamenlijk_huishouden, // Event following FRS melding 71

120 //Einde_gezamenlijk_huishouden, // FRS melding 72
    Fevt_einde_gezamenlijk_huishouden, // Event following FRS melding 72

    Fevt_burger_krijgt_kind,

125 //Burger_tekent_als_toeslagpartner_of_medebewoner, // FRS melding 77
    //Fevt_handtekeningrelatie_lopend_partner, // (see FRS melding 3)
```

```
        Fevt_handtekeningrelatie_lopend_medebewoner, // Event following FRS
            melding 77


130     Fevt_handtekeningrelatie_kinderopvangtoeslag, // TODO source


        //Burger_verzoekt_wisseling_toeslagaanvrager_en_partner, // FRS
            melding 78
        Fevt_rolwisseling_kinderopvangtoeslag, // Event following FRS
            melding 78
135
        //Burger_heeft_onderhuurder, // FRS melding 79
        Fevt_start_onderhuurder, // Event following FRS melding 79

        //Burger_heeft_geen_onderhuurder, // FRS melding 80
140     Fevt_einde_onderhuurder, // Event following FRS melding 80

        //Burger_ontvangt_aanvullende_bijdrage_voor_Kinderopvang, // FRS
            melding 82
        Fevt_burger_ontvangt_aanvullende_bijdrage, // Event following FRS
            melding 82

145     //Burger_ontvangt_geen_aanvullende_bijdrage_voor_Kinderopvang, //
            FRS melding 83
        Fevt_burger_ontvangt_geen_aanvullende_bijdrage, // Event following
            FRS melding 83

        // end FRS

150     // Start AWIR F_bepalen AWIR partner gevolgen
        Gevt_start_AWIR_partnerschap,
        Gevt_einde_AWIR_partnerschap,
        Hevt_bepalen_AWIR_partnergevolgen_uitval,
        Hevt_AWIR_partner_niet_uniek_te_bepalen,
155     Tevt_bepalen_AWIR_partnergevolgen_gereed,

        //Cevt_geen_loon_vrijgegeven,
        //Cevt_genereer_schatting_toetsingsinkomens,
        Gevt_geschat_10pct_toetsingsinkomen,
160     Gevt_definitief_10pct_toetsingsinkomen,// end AWIR
            F_bepalen_toetsingsinkomen (incoming)
        Evt_geschat_toetsingsinkomen,// start AWIR
            F_bepalen_toetsingsinkomen (outgoing)
        Evt_definitief_toetsingsinkomen,
        Evt_schatting_10pct_toetsingsinkomen,
        Evt_definitief_10pct_toetsingsinkomen,
165     Hevt_bepalen_toetsingsinkomen_uitval,
        Tevt_bepalen_toetsingsinkomen_gereed, // end AWIR
            F_bepalen_toetsingsinkomen (outgoing)

        // start AWIR F_bepalen_vermogen (incoming)
        Fevt_voordeel_sparen_en_beleggen,
170     Fevt_end_voordeel_uit_sparen_en_beleggen,
```

```
        Gevt_vermogen,
        // end AWIR F_bepalen_vermogen (incoming)

        // start AWIR F_bepalen_vermogen (outgoing)
175     Evt_vermogen,
        Hevt_vermogen_uitval,
        Tevt_bepalen_vermogen_gereed,
        // end AWIR F_bepalen_vermogen (outgoing)

180


        // start F_bepalen_schatting_draagkracht_kinderopvangtoeslag
            (incoming)
        //Evt_HH_kinderopvangtoeslag,
        Evt_hh_kinderopvangtoeslag,
185     //Evt_huishouden_kinderopvangtoeslag,


        //Evt_schatting_toetsingsinkomen,
        Gevt_draagkracht_schatting_kinderopvangtoeslag,
190     //Evt_definitief_toetsingsinkomen,
        //Evt_definitief_10pct_toetsingsinkomen,
        //Evt_schatting_10pct_toetsingsinkomen,
        Fevt_geboorte_kind,
        //Fevt_burger_overleden,
195     Cevt_jaarvrijgave_draagkracht_kinderopvangtoeslag,
        // end F_bepalen_schatting_draagkracht_kinderopvangtoeslag (incoming)

        // start F_bepalen_schatting_draagkracht_kinderopvangtoeslag
            (outgoing)
        //Evt_draagkracht_schatting_kinderopvangtoeslag,
200     Hevt_bepalen_schatting_draagkracht_kinderopvangtoeslag_uitval,
        Hevt_bepalen_draagkracht_kinderopvangtoeslag_uitval,
        Tevt_bepalen_schatting_draagkracht_kinderopvangtoeslag_gereed,
        Tevt_bepalen_draagkracht_kinderopvangtoeslag_gereed,
        // end F_bepalen_schatting_draagkracht_kinderopvangtoeslag (outgoing)
205
        // start F_vaststellen_draagkracht_kinderopvangtoeslag
        Evt_draagkracht_definitief_kinderopvangtoeslag,
        // end F_vaststellen_draagkracht_kinderopvangtoeslag

210     // start F_beslissen_kinderopvangtoeslag
        Evt_draagkracht_schatting_kinderopvangtoeslag, // Gevt ?
        // Evt_draagkracht_definitief_kinderopvangtoeslag
        //Evt_normen_kinderopvangtoeslag,
        //Cevt_dataset_proefberekening_kinderopvangtoeslag,
215     //Fevt_verblijfsstatus,
        //Fevt_burger_ontvangt_inkomsten_uit_werk,
        //Fevt_burger_ontvangt_geen_inkomsten_uit_werk,
        //Fevt_burger_ontvangt_aanvullende_bijdrage,
        //Fevt_burger_ontvangt_geen_aanvullende_bijdrage,
220
        //Fevt_burger_gedetineerd,
        //Fevt_burger_niet_gedetineerd,
```

```
        //Fevt_geboorte_kind,
        //Bevt_beslissing_kinderopvangtoeslag,
225     //Fevt_burger_inschrijven_op_adres,
        //Fevt_meerdere_burgers_inschrijven_op_adres,

        // AWIR: bepalen toetsingsinkomen
        Cevt_geen_loon_vrijgeven,
230     Cevt_genereer_schatting_toetsingsinkomens,
        //Gevt_geschat_10pct_toetsingsinkomen,
        //Gevt_definitief_10pct_toetsingsinkomen,


235     // AWIR: toeslagbetrokkenheid
        Evt_burger_toeslagbetrokkene,
        Hevt_bepalen_toeslagbetrokkenheid_uitval,
        Tevt_bepalen_toeslagbetrokkenheid_gereed,

240
        // Uitgaande events F_beslissen_kinderopvangtoeslag
        Evt_beslissing_kinderopvangtoeslag,
        Hevt_beslissen_kinderopvangtoeslag_uitval,
        Tevt_beslissen_kinderopvangtoeslag_gereed,
245

        // Incoming events F_bepalen_lasten_kinderopvangtoeslag,
        // Fevt_kinderopvanggebruik,
        // Fevt_burger_inschrijven_op_adres,
250     // Fevt_meerdere_burgers_inschrijven_op_adres,
        // Fevt_geboorte_kind,
        // Fevt_verjaring_5_jaar,
        // Fevt_verjaring_13_jaar,
        // Fevt_overlijden,
255     Cevt_herbereken_1_kinderopvangtoeslag,
        Gevt_lasten_kinderopvangtoeslag,


        // Outgoing events F_bepalen_lasten_kinderopvangtoeslag
260     //Evt_lasten_kinderopvangtoeslag,
        //Evt_lasten_kinderopvangtoeslag_rekenkosten_per_uur_kc_do,
        //Evt_lasten_kinderopvangtoeslag_rekenkosten_per_uur_kc_bso,
        //Evt_lasten_kinderopvangtoeslag_rekenkosten_per_uur_go_do,
        //Evt_lasten_kinderopvangtoeslag_rekenkosten_per_uur_go_bso,
265     Hevt_bepalen_lasten_kinderopvangtoeslag_uitval,
        Tevt_bepalen_lasten_kinderopvangtoeslag_gereed,



270     Evt_burger_krijgt_behandelsoort_gezinsbijslag,

        // Incoming events F_beschikken_kinderopvangtoeslag
        Evt_ambtelijke_conclusie_kinderopvangtoeslag,
        Cevt_tijdstip_beschikken,
275     Evt_AWIR_indicatie_meerderjarigheid_jonger_dan_meerderjarigheidsleeftijd,
        Evt_verzoek_toelichting_beschikking_kinderopvangtoeslag,
```

```
        Fevt_burgerkrijgt_behandelsoort_gezinsbijslag, // TODO
        // Outgoing events F_beschikken_kinderopvangtoeslag

        Evt_beschikking_kinderopvangtoeslag,
        Evt_toelichting_beschikking_kinderopvangtoeslag,
        //Evt_beschikking_BoB_kinderopvangtoeslag,
        Evt_concept_beschikken_kinderopvangtoeslag,
        Tevt_beschikken_kinderopvangtoeslag_gereed,
        Hevt_beschikken_kinderopvangtoeslag_uitval,


        Evt_start_AWIR_partnerschap,
        Evt_einde_AWIR_partnerschap,
    //
        Evt_AWIR_indicatie_meerderjarigheid_jonger_dan_meerderjarigheidsleeftijd,
        //Hevt_bepalen_AWIR_partnergevolgen_uitval,
        //Tevt_bepalen_AWIR_partnergevolgen_gereed,


        //Evt_draagkracht, // 7.5.3



          // Outgoing events F_vaststellen_draagkracht_kinderopvangtoeslag
          //Evt_draagkracht_definitief_kinderopvangtoeslag,
          Tevt_vaststellen_draagkracht_kinderopvangtoeslag_gereed,
          Hevt_vaststellen_draagkracht_kinderopvangtoeslag_uitval


          // F_vaststellen_huishouden_kinderopvangtoeslag
          Tevt_vaststellen_huishoudsamenstelling_kinderopvangtoeslag_gereed,
          Hevt_vaststellen_huishoudsamenstelling_kinderopvangtoeslag_uitval,
        Gevt_hh_kinderopvangtoeslag,
        //Gevt_hh_kinderopvang_toeslag
    }
```

Listing J.1: Events

## J.2 Citizen and household

```
    // Burger variables
    /*
    typedef Inkomenssoort {
      bool NINBI = 0;
      bool verzamelinkomen = 0;
      bool fiscaal_jaarloon = 0;
    }

    typedef Inkomensstatus {
      bool aangifte = 0;
      bool definitief = 0;
      bool herzien = 0;
```

```
    bool onbekend = 0;
    bool in_onderzoek = 0;
}
*/

typedef HUISHOUDEN {
  bool som_inkomen_positief = 0; // "Som van alle geschatte inkomens
      alle leden huishouden < 0"
  bool draagkracht = 0; // voorlopig bool, draagkracht van huishouden
  bool geen_recht_op_kinderopvangtoeslag = 0; // recht op
      kinderopvangtoeslag
  bool lasten_kinderopvangtoeslag = 0;
}

typedef BURGER {
  //bool partner = 0;
  byte AWIR_level = 0;
  byte AWIR_sublevel = 0;
  byte AWIR_partnerschap = 0;
  byte handtekening = 0;
  byte BSN = 0;

//  bool handtekeningrelatie = 0;


//  bool handtekening_lopend_partner = 0;
//  bool handtekening_lopend_medebewoner = 0;
  byte geregistreerd_partnerschap = 0;


  bool lopende_aanvraag_met_ander_persoon = 0; // TODO: store burgerId
      (fake BSN) ?
  bool lopende_eenpersoonsaanvraag = 0;
  bool lopende_gezamenlijke_tweepersoonsaanvraag = 0;


  byte samenlevingscontract = 0;
  bool overleden = 0;
  bool verblijfsstatus = 0;
  bool kinderopvangtoeslag = 0;


  bool nationaliteit[2] = 0; // EU or not (see TODO)
  //bool nationaliteit2 = 0; // NL of niet?

  bool gaat_naar_school = 0;


  // opvangsoort  (5.4.1 SD)
  bool dagopvang = 0;
  bool buitenschoolseopvang = 0;
  // opvangvorm (5.4.1 SD)
  bool gastouderopvang = 0;
  //bool kindercentrum = 0; //not used??
```

```
    //bool thuisopvang = 0; //not used??

     // TODO uitzoeken waar dit vandaan komt
    bool lasten_kinderopvang_bekend = 0;

    byte niveau_van_behandeling = 1;

    bool gebruikt_uren_kinderopvang = 0;


    byte huishouden = 0; // burger is (voor deel van het jaar) deel van
        een of meerdere huishoudens
    bool geschat_inkomen_bekend = 0; // geschat inkomen bekend
    bool geschat_10pct_inkomen_bekend = 0; // geschat 10% inkomen bekend

    bool definitief_inkomen_bekend = 0; // definitief inkomen bekend
    bool definitief_10pct_inkomen_bekend = 0; // definitief 10% inkomen
        bekend


    byte aanvraagnummer = 0;

    bool aanvrager = 0;

    bool uren_kinderopvang_geclaimd = 0;
    bool over_norm_max_uren_KOT = 0;

    byte adres = 0;
    bool voor_1_juli_op_adres = 0;

    byte heeft_kind = 0;
    byte heeft_kind_met = 0;
    //bool is_kind_van = 0;

    byte moeder = 0;
    byte vader = 0;

    //bool zelfde_ouder = 0;

    bool beschreven_IB = 0;

    bool beschreven_NINBI = 0;

    bool verzamelinkomen_bekend = 0;

    bool NINBI_bekend = 0;

    bool toetsingsinkomen_bekend = 0;
    bool toetsingsinkomen_negatief = 0;

    bool fiscaal_jaarloon_bekend = 0;

    bool is_onderhuurder = 0;
    bool heeft_onderhuurder = 0;
```

```
    bool uitstel_IB = 0;
    byte fiscaal_partnerschap = 0;

    bool duurzaam_gescheiden = 0;

    byte gezamenlijke_schuld = 0;
    byte partners_in_pensioenregeling = 0;

    bool gedetineerd = 0;
    bool inkomsten_uit_werk = 0;
    bool aanvullende_bijdrage = 0;
    bool ouder_dan_vijf = 0;
    bool ouder_dan_dertien = 0;
    bool ouder_dan_achttien = 0;

    bool vader_overleden = 0;
    bool moeder_overleden = 0;
    byte getrouwd = 0;



};
```

Listing J.2: Citizen and household datastructure

# Appendix K

# MSC modification

```
#!/bin/bash

# remove all variables from events
sed -i 's/\(,[0-9]\)*//g' model.pml.ps

# massive replace, easier in php
/usr/bin/php5-cgi replace.php

#create pdf
ps2pdf model.pml.ps
```

../eclipse/msc.sh

```
<?php

  $regexp = '(stroke\n){1}[\-]?([0-9]* [0-9]*
      moveto\n){1}([\-]{0,1}[0-9]* [0-9]* lineto\n){4}([0-9]\.[0-9]*
      0\.[0-9]* 0\.[0-9]* setrgbcolor AdjustColor\n){1}(closepath
      fill\n){1}([\-]?[0-9]* [0-9]* moveto\n){1}([\-]{0,1}[0-9]* [0-9]*
      lineto\n){4}';


  $content = file_get_contents('model.pml.ps');

  $content = preg_replace('/'.$regexp.'/e', '', $content, -1, $matches);

  $fp = fopen('model.pml.ps', 'w+');

  if($fp) {
    fwrite($fp, $content);
    fclose($fp);
  }

?>
```

../eclipse/replace.php

# Appendix L

# Known error settings

```
1 inline KE111() {
2
3   int i;
4   for(i : 0 .. MAX_SERVICE_ID) { // set same burger info on all service
        databases
5
6       // burger A
7       b.handtekening = 2; // handtekeningrelatie with B
8       b.BSN = 1;
9       b.AWIR_partnerschap = 2; // AWIR-partner with B
10      b.adres = 1;
11      b.ouder_dan_achttien = 1;
12      b.voor_1_juli_op_adres = 1;
13      copy_burger(tslgs[i].burgers[b.BSN], b);
14      //copy_huishouden(tslgs[i].hh, h);
15
16      // burger B
17      b.handtekening = 1; // handtekeningrelatie with A
18      b.BSN = 2;
19      b.AWIR_partnerschap = 1; // AWIR-partner with A
20      b.ouder_dan_achttien = 1;
21      b.adres = 1;
22      b.voor_1_juli_op_adres = 1;
23      copy_burger(tslgs[i].burgers[b.BSN], b);
24
25      // burger C
26      b.BSN = 3;
27      b.AWIR_partnerschap = 0;
28      b.adres = 1;
29      b.ouder_dan_achttien = 1;
30      b.voor_1_juli_op_adres = 1;
31      copy_burger(tslgs[i].burgers[b.BSN], b);
32
33  }
34
35
```

```
36        b1.handtekening = 3; // handtekeningrelatie A-C
37        b1.BSN = 1;
38        //b1.partner = 1;
39        b1.ouder_dan_achttien = 1;
40        b1.voor_1_juli_op_adres = 1;
41
42        if
43        :: true -> b2.getrouwd = 3; // partnerschap A-C
44        :: true -> b2.samenlevingscontract = 3; // partnerschap A-C
45        :: true -> b2.fiscaal_partnerschap = 3; // partnerschap A-C
46        :: true -> b2.heeft_kind_met = 3; // partnerschap A-C
47        :: true -> b2.partners_in_pensioenregeling = 3; // partnerschap A-C
48        fi;
49
50        b2.BSN = 1;
51        b2.ouder_dan_achttien = 1;
52        b2.voor_1_juli_op_adres = 1;
53
54
55
56 }
```

Listing L.1: Settings for KE 111

```
1 inline CKE190() {
2   int i;
3   for(i : 0 .. MAX_SERVICE_ID) { // set same burger info on all service
        databases
4
5         // burger 3450
6         b.BSN = 1;
7         b.AWIR_partnerschap = 2;
8         b.adres = 1;
9         b.moeder = 0;
10        b.vader = 0;
11        b.heeft_kind = 4;
12        b.ouder_dan_achttien = 1;
13        b.voor_1_juli_op_adres = 1;
14        b.fiscaal_partnerschap = 0;
15        b.handtekening = 0;
16        b.lopende_gezamenlijke_tweepersoonsaanvraag = 0;
17        b.samenlevingscontract = 0;
18        copy_burger(tslgs[i].burgers[b.BSN], b);
19        //copy_huishouden(tslgs[i].hh, h);
20
21        // burger 3462
22        b.BSN = 2;
23        b.AWIR_partnerschap = 1;
24        b.adres = 1;
25        b.moeder = 0;
26        b.vader = 0;
27        b.heeft_kind = 4;
28        b.ouder_dan_achttien = 1;
29        b.voor_1_juli_op_adres = 1;
```

```
30        b.fiscaal_partnerschap = 0;
31        b.handtekening = 0;
32        b.lopende_gezamenlijke_tweepersoonsaanvraag = 0;
33        b.samenlevingscontract = 0;
34        copy_burger(tslgs[i].burgers[b.BSN], b);
35
36        // burger 3474
37        b.BSN = 3;
38        b.AWIR_partnerschap = 0;
39        b.adres = 1;
40        b.moeder = 0;
41        b.vader = 0;
42        b.heeft_kind = 0;
43        b.ouder_dan_achttien = 1;
44        b.voor_1_juli_op_adres = 1;
45        b.fiscaal_partnerschap = 0;
46        b.handtekening = 0;
47        b.lopende_gezamenlijke_tweepersoonsaanvraag = 0;
48        b.samenlevingscontract = 0;
49        copy_burger(tslgs[i].burgers[b.BSN], b);
50
51        // 3401 (kind van 3450 en 3462)
52        b.BSN = 4;
53        b.AWIR_partnerschap = 7;
54        b.adres = 2;
55        b.moeder = 1;
56        b.vader = 2;
57        b.heeft_kind = 8;
58        b.ouder_dan_achttien = 1;
59        b.voor_1_juli_op_adres = 1;
60        b.fiscaal_partnerschap = 0;
61        b.handtekening = 7;
62        b.lopende_gezamenlijke_tweepersoonsaanvraag = 1;
63        b.samenlevingscontract = 5;
64        copy_burger(tslgs[i].burgers[b.BSN], b);
65
66        // 3413
67        b.BSN = 5;
68        b.AWIR_partnerschap = 6;
69        b.adres = 2;
70        b.moeder = 1;
71        b.vader = 2;
72        b.heeft_kind = 0;
73        b.ouder_dan_achttien = 1;
74        b.voor_1_juli_op_adres = 1;
75        b.fiscaal_partnerschap = 6;
76        b.handtekening = 6;
77        b.lopende_gezamenlijke_tweepersoonsaanvraag = 1;
78        b.samenlevingscontract = 4;
79        copy_burger(tslgs[i].burgers[b.BSN], b);
80
81        // 3425
82        b.BSN = 6;
83        b.AWIR_partnerschap = 5;
```

```
84        b.adres = 2;
85        b.moeder = 0;
86        b.vader = 0;
87        b.heeft_kind = 0;
88        b.ouder_dan_achttien = 1;
89        b.voor_1_juli_op_adres = 1;
90        b.fiscaal_partnerschap = 5;
91        b.handtekening = 5;
92        b.lopende_gezamenlijke_tweepersoonsaanvraag = 1;
93        b.samenlevingscontract = 0;
94        copy_burger(tslgs[i].burgers[b.BSN], b);
95
96        // 3437
97        b.BSN = 7;
98        b.AWIR_partnerschap = 4;
99        b.adres = 2;
100       b.moeder = 0;
101       b.vader = 0;
102       b.heeft_kind = 8;
103       b.ouder_dan_achttien = 1;
104       b.voor_1_juli_op_adres = 1;
105       b.fiscaal_partnerschap = 0;
106       b.handtekening = 4;
107       b.lopende_gezamenlijke_tweepersoonsaanvraag = 1;
108       b.samenlevingscontract = 0;
109       copy_burger(tslgs[i].burgers[b.BSN], b);
110
111       // 3???
112       b.BSN = 8;
113       b.AWIR_partnerschap = 0;
114       b.adres = 2;
115       b.moeder = 7;
116       b.vader = 4;
117       b.heeft_kind = 0;
118       b.ouder_dan_achttien = 1;
119       b.voor_1_juli_op_adres = 1;
120       b.fiscaal_partnerschap = 0;
121       b.handtekening = 0;
122       b.lopende_gezamenlijke_tweepersoonsaanvraag = 0;
123       b.samenlevingscontract = 0;
124       copy_burger(tslgs[i].burgers[b.BSN], b);
125
126   }
127
128     // event data
129     b2.getrouwd = 3; // partnerschap 3425-3474
130     b2.BSN = 6;
131     b2.ouder_dan_achttien = 1;
132     b2.voor_1_juli_op_adres = 1;
133
134 }
```

Listing L.2: Settings for CKE 191