Ontwerp en ontwikkeling van generieke simulatiecomponenten voor het evalueren van ontologiegebaseerde applicaties

Maarten van Dessel

Promotor: prof. dr. ir. Filip De Turck

Begeleiders: dr. Femke Ongenae, Jan Van Ooteghem, dr. Stijn Verstichel, Gregory Casier, prof. dr. Jan Cnops

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Daniël De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2014-2015

UNIVERSITEIT GENT

Toelating tot bruikleen

Ontwerp en ontwikkeling van generieke simulatiecomponenten voor het evalueren van ontologiegebaseerde applicaties

Maarten van Dessel

Promotor: prof. dr. ir. Filip De Turck

Begeleiders: dr. Femke Ongenae, Jan Van Ooteghem, dr. Stijn Verstichel, Gregory Casier, prof. dr. Jan Cnops

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Daniël De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2014-2015

UNIVERSITEIT GENT

Woord vooraf

Alvorens u zich door de technische details mag worstelen, is een woord van dank op zijn plaats. Deze thesis werd namelijk uitgevoerd met de hulp van verschillende personen. Elk hebben ze hun bijdrage geleverd die mee tot het eindresultaat heeft geleid.

Op de eerste plaats wil ik mijn begeleiders bedanken: Jan Van Ooteghem, Femke Ongenae, Stijn Verstichel en Gregory Casier. Voor al mijn vragen kon ik steeds bij hen terecht. Dankzij de vele vergaderingen en hun uitgebreide feedback is deze thesis geworden wat het is. Ook wil ik de interne promotoren, Marleen Denert en Jan Cnops, bedanken voor de begeleidende rol die zij hebben gespeeld.

Verder wil ik mijn externe promotor, Filip De Turck, bedanken voor het mogelijk maken van dit onderzoek. Het was een zeer boeiend onderwerp dat de mogelijkheid bood om mij te verdiepen in technologieën die minder uitgebreid aan bod komen tijdens de lessen. Onduidelijkheden hieromtrent werden meestal uitgeklaard met een vraag aan mijn begeleiders. Lukte dit niet, dan waren er altijd anderen waarbij ik terecht kon. Zo wil ik Koen Casier bedanken voor het interessante gesprek over de simulator, Tom De Caluwé van Amaron voor de uitgebreide uitleg over *call activities* en Pieter Bonte die hulp bood bij het opzetten van de testservers van iMinds. Ook Femke De Backere wil ik bedanken voor het aanleveren van de papers rond toepassingen van ontologiegebaseerde systemen. Aan dit lijstje mag de SPARQL-community worden toegevoegd, waarbij ik een oplossing vond voor vele problemen.

Tot slot wil ik iedereen bedanken die mij gedurende mijn thesisjaar steun heeft geboden: de collega masterstudenten op de Zuiderpoort, mijn kotgenoten, familie en vrienden.

Bedankt!

Samenvatting

Ontwerp en ontwikkeling van generieke simulatiecomponenten voor het evalueren van ontologiegebaseerde applicaties

Maarten van Dessel

Bedrijfprocessen kunnen op verschillende manieren geoptimaliseerd worden. Een mogelijke aanpak is door bronnen op een intelligente te beheren met behulp van een kennisgebaseerd systeem. Dit is software dat in het algemeen bestaat uit een ontologie en een bijhorende regelbank. Ondanks de uitgebreide mogelijkheden van een dergelijk systeem, kan er nooit met zekerheid gezegd worden dat dit zal bijdragen tot een verbeterde uitvoering van het proces. Hiertoe dient het systeem eerst getest te worden, bijvoorbeeld door het te simuleren in een virtuele omgeving. Hoewel processimulatie een mogelijke piste is, voorziet het niet direct methoden om een kennisgebaseerd systeem te testen dat gebruikt wordt voor het beheer van bronnen in een proces. Daartoe werd in deze thesis een systeem ontwikkeld dat dit mogelijk maakt voor een willekeurig kennisgebaseerd systeem. Een implementatie van dit systeem werd ontwikkeld als uitbreiding op een bestaand simulatieframework. Als laatste werd deze implementatie getest op schaalbaarheid en bruikbaarheid.

Kernwoorden: Semantische technologieën, Regelbanken, Discrete Event Simulatie, Processimulatie

Summary

Design and Development of Generic Simulation Components for the Evaluation of Ontology-based Applications

Maarten van Dessel

Business processes can be optimized in various ways. One way is to perform intelligent resource management based on a knowledge based system. Typically an ontology and a complementary rule system is used. Although this allows for complex decisions, it does not necessarily improve the efficiency of the business process. Therefore the system should be tested, e.g. by running simulations of the business process in which the system is used. Process simulation isn't new, but it does not provide methods to test an arbitrary knowledge based system used for resource allocation. This study strives to close this gap. The designed system allows to test a knowledge based system used for resource management. This was made possible through process simulation. An implementation of this system was made as an extension for an existing process simulation framework. Finally the implementation was tested on scalability and usability.

Keywords: Semantic Technologies, Rule Systems, Discrete Event Simulation, Process Simulation

Design and Development of Generic Simulation Components for the Evaluation of Ontology-based Applications

Maarten van Dessel

Supervisor(s): prof. dr. ir. Filip De Turck, dr. Femke Ongenae, Jan Van Ooteghem, dr. Stijn Verstichel, Gregory Casier, prof. dr. Jan Cnops

Abstract—**Business processes can be optimized in various ways. One way is to perform intelligent resource management based on a knowledge based system. Typically an ontology and a complementary rule system is used. Although this allows for complex decisions, it does not necessarily improve the efficiency of the business process. Therefore the system should be tested, e.g. by running simulations of the business process in which the system is used. Process simulation isn't new, but it does not provide methods to test an arbitrary knowledge based system used for resource allocation. This study strives to close this gap. The designed system allows to test a knowledge based system used for resource management. This was made possible through process simulation. An implementation of this system was made as an extension for an existing process simulation framework. Finally the implementation was tested on scalability and usability.**

The following paper describes the different components of the system and the test result. This is followed by the conclusion, as well as future work.

Keywords—**Semantic Technologies, Rule Systems, Discrete Event Simulation, Process Simulation**

I. Introduction

IN this computerized world, businesses seek ways to integrate their business logic into supportive applications[1]. Those applications then take over tasks that are usually performed manually. An example of such a task is resource management, where the correct resources (e.g. staff, equipment) are assigned to certain tasks in the process. Through this automation, business processes are sought to improve. One way to implement this, is by using a knowledge based system. An example is the ACCIO project[1][2] where a system was designed that assigned nurses to calls based on their location, competences and more context specific information. All this information is modelled through an ontology (see subsection II-A.1) and the allocation occurs via the execution of rules.

Before such implementations are used in a practical situation, they must be tested. First, to know the impact on the efficiency of the business process, second to optimize the system per implementation, e.g. the allocation of a nurse in a cancer division is slightly different than in the urgency devision (e.g. taking into account the relationship with the patient). The decisions of the system should reflect these differences.

Although simulation seems appropriate, there is currently no convenient way to test such systems (using an ontology and associated rules) through process simulation. One could build simulation software that emulates the logic of the knowledge based system, but this is an inefficient and time consuming workflow[3]. For this reason we developed a generic way to use a knowledge based system during process simulation for the allocation of resources. The knowledge model is interpreted by the system. As a consequence the system should not be altered if the knowledge model changes.

An implementation of this system was made as an extension for an existing process simulation framework[4]. It was then tested on usefulness, scalability and usability.

This article is structured as follows. The first section describes the architectural model which has been used, the following sections the different components of this model. In the final section the test results are discussed and some conclusions are presented.

II. Architecture

The architecture used in our approach exists of three main components: the process simulator, the knowledge based system and a bridge component which connects those two separate systems (see Figure 1). The three components are each discussed in a separate subsection.

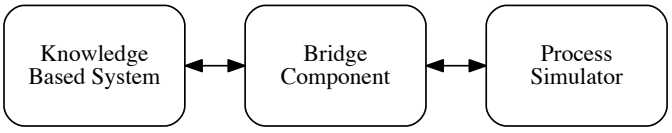


Fig. 1: The three main components of the architectural model.

A. Knowledge based system

The knowledge based system is designed to manage the resources of a specific business process. The impact of this component on the business process is what needs to be tested. The system itself consists of a knowledge base and a set of rules. Both parts are discussed in their respective subsections.

A.1 Knowledge base

The knowledge base is used to model the domain in which the process executes. It exists of entities (e.g. `Patient`, `Room`) with attributes and relations between those entities (e.g. a `Patient` is assigned to a `Room` via the relationship `belongsTo`). This model is supplemented with certain individ-

M. van Dessel is with the Industrial Engineering Department, Ghent University (UGent), Gent, Belgium. E-mail: Maarten.vanDessel@UGent.be .

[1]iMinds ACCIO project, `https://www.iminds.be/nl/projecten/2014/03/04/accio` (last checked 05/06/2015)

uals which are instantiations of those entities (e.g. `Alice` is an individual of the type `Patient`).

Although a typical knowledge base is structured this way, most implementations used for resource allocation require more complexity. This can be achieved through an ontology, in which *axioms* can be added to the knowledge base. These are statements that describe knowledge that is not explicitly modelled. This knowledge can be inferred from the existing knowledge by reasoning upon it. Applications that perform this task are called *Reasoners*. An example axiom could be: every `Person` with the relationship `hasChild` is a `Parent`. Entities satisfying this condition would be categorized as a `Parent` and receive the relationships belonging to it.

A.2 Rules

The second part of a knowledge based system consists of rules. These are statements that execute an action for entities that satisfy a certain condition. Most of the time this action implies the assertion of knowledge in the model (an example of a rule is given in Figure 2).

In this study, these rules are specifically used to allocate resources. If a resource is needed, the conditions of the rules are checked one by one. This operation continues, until one or more entities of the knowledge base satisfy the conditions of the rule. The body of the rule is then asserted in the ontology, which results in the allocation of the resource.

In a real-world example, this operation would usually be complemented with a real-world action (e.g. a nurse who is assigned a specific task would be informed through his/her personal device).

B. Process simulator

The process simulator is used to test the utilisation of the knowledge based system in a virtual environment. This virtual environment is created from a formal process definition, which allows it to be interpreted by software. The process definition is used by the framework to create many instances of the process. For each instance, the required resources are allocated to the different tasks in the process.

In order to measure the efficiency of the process, performance indicators are calculated during the simulation (e.g. workload, mean duration of a task). Afterwards these indicators are used to calculate the key performance indicators of the process. These values can then be used to compare other executions of the process (e.g. without intelligent resource allocation).

C. Bridge component

The knowledge based system and the simulator are two independent components. In order to make them collaborate, a separate component is used for the communication betwoon those two systems. If the simulator needs a resource, it will delegate this request to the bridge component. The bridge component will then perform certain actions on the knowledge based system and eventually return a resource (when available). This way the simulator and the knowledge based system remain independent from each other.

III. Implementation

The designed system was implemented as an extension for an existing process simulator. The existing framework[4] is capable of simulating formal process definitions (modelled with Business Process Modeling Notation). Additionally, a simple mechanism[2] is available to specify resources for tasks in the process. In order to extend these capabilities with those of a knowledge based system, two questions have to be answered:

- Which ontology language will be supported?
- Which rule language will be supported?

Web Ontology Language (OWL) was chosen as the supported ontology language. It is defined as a recommendation by World Wide Web Consortium (W3C) for the description of ontologies[5]. It is also commonly used in implementations of resource management tools[2], [6], [7].

While most ontologies are written in OWL, the language choice for rules can vary depending on the systems requirements. To keep the designed system as generic as possible, we decided to support the Rule Interchange Format (RIF). This language is specifically designed to enable interoperability among rule languages in general[8].

Although RIF is also recommended by the W3C, it is hard to implement. Only a few implementations were found which could directly interpret RIF[9], [10], but none of those work on top of an OWL based knowledge model. Because of this, it was decided to translate RIF into SPARQL, an RDF[3] based query language. Even though SPARQL may not be designed as a rule language, it provides a rich syntax which allows to write queries as rules[11]. Furthermore it is a widely supported standard of which multiple implementations exist. Since OWL is compatible with RDF, SPARQL can then directly be used to query the knowledge base. The translation itself is based on [12]. The execution of the queries happens via Apache Jena[4].

A. Rule execution

As previously explained (subsection II-A.2), the execution of a rule effectively allocates a resource in the ontology. Due to the behaviour of a rule, it is possible that multiple entities may satisfy the conditions of a rule. All those entities are then allocated. In the ACCIO-project, multiple nurses would receive a call on their personal device. However, only one would accept the call and perform the task. Due to the nature of the simulation framework, this behaviour (the acceptance of a call) cannot be emulated. In the virtual environment of process simulation, resources are directly assigned to the execution of a task. Therefore only one of the available resources should be affected by the rule execution. Because of this, the allocation process is divided into three steps:

[2]The original mechanism allows to specify different types of resources and their amount. Those are then managed using the mechanisms of a queue.

[3]Resource Description Format (RDF) is a primitive language used to model knowledge about a certain domain. OWL is defines as an extension to RDF which enables more expressiveness and functionality.

[4]Apache Jena, `https://jena.apache.org` (last checked 05/06/15)

1. find all available resources based on the conditions of the rule
2. select one resource from all available resources
3. allocate the selected resource

Different approaches were tested that followed this procedure. Each one involved the execution of different types of SPARQL-queries based on the translated rules[5]. Eventually one seemed to be considerably faster (based on tests described in section IV). It first detects all available resources using a SPARQL SELECT query. From these results a random, available resource is selected. This resource is then allocated by executing the (translated) rule on the ontology using a SPARQL UPDATE query. In order to affect only the selected resource, the variable representing the resource identifier is substituted in the final query. An example of a translation is presented in Figure 2.

```
ForAll ?person
  "metadata"["resource-identifier" -> "person"]
{
    If   ?person hasStatus "free"
    Then ?person hasStatus "occupied"
}
```

(a) An example RIF rule, used for resource allocation.

```
SELECT *
WHERE {
    ?person hasStatus "free" .
}
```

(b) The translation of the rule in SPARQL, used to identify available resources.

```
DELETE {
    ?person hasStatus ?status .
}
INSERT {
    ?person hasStatus "occupied" .
}
WHERE {
    ?person hasStatus "free" .
    ?person hasStatus ?status .
}
```

(c) The translation of the rule in SPARQL, used for the allocation of a specific resource (`?person` is substituted by the identification of the selected resource).

Fig. 2: The translation of a RIF rule to a SPARQL query, when used during simulation to allocate a resource.

B. OWL compatibility

Since the implementation uses ontologies modelled in OWL, additional functionality was provided.

Firstly it is possible to use a reasoner in combination with the ontology. Our implementation uses the reasoner provided by Jena, which is flexible enough to allow the use of a custom reasoner if needed.

[5]Approaches were tested which combined two or more allocation steps in a single SPARQL-query. Those turned out have a negative impact on the average simulation time.

A second addition is the support of functional properties in OWL. These are properties for which only one may exist per subject in the ontology. Problems may occur if a rule asserts statements containing a functional property. In that case the old statement should first be deleted. This is taken care of by using the DELETE clause of SPARQL, which can remove statements from the ontology before adding new ones.

IV. Tests

The implementation was tested. Firstly the effect of different configurations on the duration of a simulation was studied. Afterwards, the usability was tested. Both tests are described in the following paragraphs.

A. Scalability tests

To test the scalability of the system, different values for different parameters were tested. These parameters include the process size[6], the size of the ontology and the shortage of resources[7]. For each configuration, the average simulation time was determined and used as a key to compare to other configurations. The results were also used to compare different approaches for allocating resources (subsection III-A).

The parameter that had the most significant effect on the duration of a simulation is the shortage of resources. If only a few resources are available for many process instances, the simulator continuously looks for available resources. Although the simulator has the capabilities to predict the deallocation of a resource, those are not put to use. Since the execution of queries is expensive in comparison with other tasks in the simulator, this results in significantly longer simulation times. Figure 3 shows the impact of resource shortage on the average simulation time for two different allocation methods.

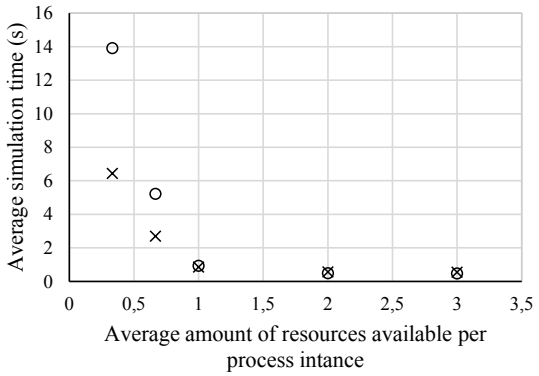


Fig. 3: The impact of the shortage of resources in a business process on the average duration of a simulation for two different allocation methods.

B. Usability tests

A second test was executed to check the practical usage of the designed system. Therefore it was used with a real process describing a patient which had to be operated on. This

[6]The amount of tasks in a linear business process

[7]This occurs if a limited amount of resources serve a lot of process instances.

involved the modelling of the business process, the modelling of the resources[8] and the creation of simulator specific configuration files. These tasks were performed based on informal process descriptions originating from interviews.

During this test it became clear that too many steps involved the manipulation of XML[9] files. This results in a time-consuming and error-prone workflow. A complementary GUI[10] could be built to simplify and automate some of these steps.

V. Conclusion

In this study we sought to create a system which allows to test an arbitrary knowledge based system used for resource management tasks. This was achieved by using process simulation which creates a virtual environment in which the knowledge system can operate. An implementation was made for an existing process simulation framework as a proof of concept. It is capable of interpreting knowledge based systems composed of an OWL ontology and a complementary RIF rule system. However, in order to be of practical use, more effort should be put into the creation of supported tools such as a GUI. Furthermore, the procedure used to allocate resources can be optimized in order to increase the system's performance.

References

[1] Amy Van Looy, "Does it matter for business process maturity? a comparative study on business process maturity models," in *LECTURE NOTES IN COMPUTER SCIENCE*, Robert Meersman, Tharam Dillon, and Herrero Pilar, Eds. 2010, vol. 6428, pp. 687–697, Springer.

[2] Femke Ongenae, Dries Myny, Tom Dhaene, Tom Defloor, Dirk Van Goubergen, Piet Verhoeve, Johan Decruyenaere, and Filip De Turck, "An ontology-based nurse call management system (oncs) with probabilistic priority assessment," *BMC Health Services Research*, vol. 11, no. 1, 2011.

[3] Laurens Van Poucke and Pieter Demyttenaere, "Techno-economische evaluatie van ehealth-diensten met behulp van ontologiegebaseerde operationele procesoptimalisatie," M.S. thesis, UGent, 2014.

[4] Gregory Casier, Jan Van Ooteghem, Koen Casier, and Sofie Verbrugge, "Application of a discrete event simulator for healthcare processes," in *BMSD (accepted paper)*, Milan, Italy, July 6-8 2015.

[5] W3C, *OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax*, second edition edition, december 2012.

[6] Femke De Backere et al., "The ocareclouds project: toward organizing care through trusted cloud services," *Informatics for Health and Social Care*, pp. 1–19, oktober 2014, PMID: 25325572.

[7] F. De Backere, F. Ongenae, F. Van den Abeele, J. Nelis, P. Bonte, E. Clement, M. Philpott, J. Hoebeke, S. Verstichel, A. Ackaert, and F. De Turck, "Towards a social and context-aware multi-sensor fall detection and risk assessment platform," *Computers in Biology and Medicine*, 2014.

[8] W3C, *RIF Overview*, second edition edition, februari 2013.

[9] Adrian Marte, "Rif4j - a reasoning engine for rif-bld," M.S. thesis, Semantic Technologies Institute (STI), Leopold–Franzens–Universität, Innsbruck, maart 2011.

[10] Marco Marano, Philipp Obermeier, and Axel Polleres, "Processing rif and owl2rl within dlvhex," in *Web Reasoning and Rule Systems*, Pascal Hitzler and Thomas Lukasiewicz, Eds., vol. 6333 of *Lecture Notes in Computer Science*, pp. 244–250. Springer Berlin Heidelberg, 2010.

[11] Axel Polleres, "From SPARQL to rules (and back)," in *World Wide Web Conference Series*, 2007, pp. 787–796.

[12] Oumy Seye, Catherine Faron-Zucker, Olivier Corby, and Corentin Follenfant, "Bridging the gap between rif and sparql: Implementation of a rif dialect with a sparql rule engine," in *Artificial Intelligence meets the Web of Data workshop, ECAI*, montpellier, France, augustus 2012, vol. 20, pp. 19 – 24.

[8] The ontologies created during the ACCIO-project[2] were used as a basis for the models created in this study.

[9] XML, eXtensible Markup Language

[10] GUI, Graphical User Interface

Inhoudsopgave

Lijst van figuren

Lijst van afkortingen

GHz	Gigahertz
MB	Megabyte
MHz	Megahertz
API	Application programming interface
BGP	Basic Graph Pattern
BPMN	Business Process Modeling Notation
DDR3	Double Data Rate type Three
DES	Discrete Event Simulatie
IBCN	Internet Based Communication Networks and services
IRI	International Resource Identifier
JVM	Java Virtual Machine
JVM	Java Virtual Machine
KPI	Key Performance Indicator
oNCS	Ontology-based Nurse Call management System
OWL	Web Ontology Language
PI	Performance Indicator
RDF	Resource Description Framework
REST	REpresentational State Transfer
RIF	Rule Interchange Format
RIF-BLD	RIF Basic Logic Dialect

RIF-FLD	RIF Framework for Logic Dialects
RIF-PRD	RIF Production Rule Dialect
SDRAM	Synchronous Dynamic Random-Access Memory
SPARQL	SPARQL Protocol And RDF Query Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

Hoofdstuk 1

Inleiding

Steeds meer bedrijfsprocessen worden ondersteund door software. Deze software neemt functies over die normaal manueel uitgevoerd worden. Een voorbeeld van een dergelijke functie is het toewijzen van bronnen aan taken in een bedrijfsproces. Bronnen zijn hier elementen die slechts beperkt beschikbaar zijn in het proces (bv. personeel en apparatuur). Wanneer gevraagd wordt naar een bepaalde bron, maar deze niet meer beschikbaar is, dan zal deze taak tijdelijk gepauzeerd moeten worden. Hierdoor neemt de doorlooptijd van het proces toe, wat ongewenst is.

Ook andere factoren geven een beeld van de efficiëntie van een proces. Men noemt deze factoren *Key Performance Indicators* (KPI's) en ze worden gebruikt als maatstaf voor de gemiddelde prestatie van een proces. Voorbeelden van KPI's zijn de verdeling van de werklast en de tijdspanne waarin een klant geholpen kan worden. Door bronnen op een meer intelligente manier toe te kennen, kan men de waarden van deze KPI's verbeteren.

Intelligente softwaresystemen kunnen gebruikt worden om deze taak op zich te nemen. Meer specifiek worden steeds vaker *kennisgebaseerde systemen* gebruikt voor het toekennen van bronnen in een proces [15]. Dit zijn systemen die beslissingen nemen op basis van een bepaald kennismodel. Het kennismodel bevat informatie over de omgeving en het domein waarin het systeem werkt. De beslissingen van het systeem worden gemodelleerd met behulp van regels. Dit zijn uitdrukkingen die gebruik maken van het kennismodel om op basis daarvan een specifieke actie uit te voeren.
Een veelgebruikte term is die van *situatie-bewuste systemen*, omdat hun gedrag afhankelijk is van de huidige situatie. Telkens de omgeving wijzigt, zal het systeem zijn kennis hierover bijwerken wat een gevolg heeft op de manier waarop beslissingen genomen worden.

Alvorens een kennisgebaseerd systeem in de praktijk gebruikt kan worden, moet men ervan overtuigd zijn dat het bijdraagt tot een verbeterde uitvoering van het proces [17]. Men doet hiervoor beroep op de KPI's van het proces. Bijkomende optimalisaties zijn nodig indien het

systeem gebruikt zal worden in verschillende omgevingen. In deze omgevingen (bv. verschillende afdelingen van een organisatie) zijn er andere prioriteiten en worden beslissingen op een andere manier genomen. Het systeem dient in deze situatie worden aangepast zodat er overeenstemming is met de werkelijke wereld.

De testen en optimalisaties die hiervoor nodig zijn zal men zo veel mogelijk uitvoeren in een virtuele omgeving onder de vorm van simulaties. Dit heeft meerdere voordelen. Het laat toe om op een snelle manier meerdere versies van een systeem te testen in een gecontroleerde en veilige omgeving. Men kan analyses uitvoeren op de resultaten van de simulaties die vervolgens gebruikt kunnen worden om optimalisaties voor te stellen.

Hoewel processimulatie niet nieuw is, ligt de simulatie van kennisgebaseerde systemen niet voor de hand. Momenteel dient men de logica van deze systemen te vertalen naar de logica die gebruikt wordt in de simulator. Een eenmalige omzetting is aanvaardbaar. Wanneer men echter verschillende versies van een dergelijk systeem wenst te simuleren, wordt dit een tijdrovende en inefficiënte manier van werken.

In deze studie worden daarom simulatiecomponenten ontwikkeld die deze stap dienen te vereenvoudigen. De componenten laten toe dat een kennisgebaseerd systeem gebruikt kan worden tijdens een processimulatie voor het toekennen van bronnen. De logica van het systeem wordt geïnterpreteerd door de simulator waardoor een manuele omzetting vermeden wordt.

Om de componenten zo generiek mogelijk te houden, werd ervoor gekozen om een regeltaal te ondersteunen die specifiek ontwikkeld werd voor haar generieke karakter. Hierdoor moet een regelset enkel vertaald worden naar de generieke regeltaal (in tegenstelling tot een vertaling naar programma-code). Tijdens de uiteindelijke simulaties zal deze generieke regeltaal als basis gebruikt worden voor de beslissingen die moeten genomen worden.

Er werden ook enkele uitbreidingen voorzien op de implementatie die toelaten om analyses uit te voeren op de werking van het kennisgebaseerd systeem, bijvoorbeeld het achterhalen van de regels die gebruikt werden voor de toekenning van een bron.

1.1 Doelstelling

Het ACCIO-project is een typisch voorbeeld waarbij een kennisgebaseerd systeem (het *Ontology-based Nurse Call management System* (oNCS)) gebruikt wordt voor het beheer van bronnen in een proces [14]. Medisch personeel wordt hier toegekend aan oproepen van onder andere patiënten. Indien men een dergelijk systeem wenst te testen in een reële situatie, dan komt daar

veel bij kijken (bv. installatie van nieuwe systemen, opleiding personeel, etc). Daarbij kunnen er dingen mislopen onder meer omdat het systeem nog niet op punt staat.

Om deze problemen te vermijden, kan men zowel de omgeving als het kennissysteem nabootsen met behulp van simulatiesoftware. Dit laat toe in een veilige omgeving het effect van de nieuwe software te testen. Een nadeel is dat men wijzigingen moet aanbrengen in de simulatiesoftware indien men de logica van het kennissysteem wenst te wijzigen. Daarom wordt er in deze thesis een antwoord gezocht op de volgende onderzoeksvraag:

> **Onderzoeksvraag:** Hoe kan processimulatie gebruikt worden om kennisgebaseerde systemen die instaan voor het beheer van bronnen, te testen en evalueren op een generieke manier?

Door een generiek systeem te ontwikkelen, kan eender welk kennissysteem getest worden met dezelfde software. Een wijziging van het kennissysteem mag er dus niet toe leiden dat men het simulatieframework moet aangepassen.

Er werd een implementatie gemaakt van het systeem voor de processimulator van onderzoeksgroep *Internet Based Communication Networks and services* (IBCN).

1.2 Structuur

In het volgende hoofdstuk worden de verschillende technologieën toegelicht die gebruikt werden in deze thesis. Het eerste deel geeft meer informatie over een kennisgebaseerd systeem. De verschillende componenten hiervan worden toegelicht, waarna er enkele implementaties van deze componenten worden besproken. In het tweede deel van de technologiestudie wordt discrete event simulatie besproken en de link met processimulatie. Vervolgens worden in hoofdstuk 3 de eisen gegeven die hebben geleid tot het architecturaal model. Nadien wordt dit model toegelicht en de implementatie hiervan besproken (hoofdstuk 4). Deze implementatie werd getest, zowel op schaalbaarheid als op bruikbaarheid. De testopstelling alsook de resultaten hiervan staan in hoofdstuk 5. Hierna wordt het besluit van de thesis gegeven (hoofdstuk 6) waarbij er enkele uitbreidingen worden voorgesteld die kunnen bouwen op het onderzoek verricht in deze thesis.

Hoofdstuk 2

Technologiestudie

Zoals eerder vermeld is het doel van deze thesis om kennisgebaseerde systemen te kunnen gebruiken gedurende simulaties voor het toekennen van bronnen, en dit op generieke manier. In dit hoofdstuk zal er eerst worden toegelicht wat kennisgebaseerde systemen concreet zijn (paragraaf 2.1) en uit welke componenten ze bestaan. Vervolgens worden de implementaties van deze componenten besproken die gebruikt werden in deze thesis (paragraaf 2.1.3). In een volgend deel (paragraaf 2.2) wordt besproken wat discrete event simulatie is en hoe dit gebruikt kan worden bij processimulaties. Als laatste wordt een voorbeeld van processimulatiesoftware besproken dat werd gebruikt in de uiteindelijke implementatie.

2.1 Kennisgebaseerde systemen

Kennisgebaseerde systemen worden gebruikt in de artificiële intelligentie en vallen onder de noemer expertsystemen. Ze bezitten kennis over een bepaald domein dat gemodelleerd wordt in een semantisch net. Een ontologie is een specifiek soort semantisch net.

Een kennisgebaseerd systeem laat toe dat er vragen gesteld worden over het kennismodel. Op basis van deze vragen kan men vervolgens logische beslissingen nemen. Een specifieke vorm van vragen noemt men regels. Dit zijn uitdrukkingen die op basis van het kennismodel nieuwe kennis induceren. Deze nieuwe kennis kan men vervolgens toevoegen aan het kennismodel of gebruiken voor toekomstige beslissingen.

Zowel ontologieën als regels worden verder toegelicht in de volgende paragrafen.

2.1.1 Ontologie

De term ontologie is afkomstig van de filosofie waar het een synoniem voor de zijnsleer is. Het is een tak in de metafysica die onderzoekt welke dingen bestaan en hoe men deze dingen kan cate-

goriseren [21]. Het begrip ontologie in de computerwetenschappen is hiervan afgeleid. De meest gebruikte definitie wordt gegeven door Gruber [10]: een ontologie is een formele en expliciete specificatie van een conceptualisering. Meer bepaald is het een model van een domein-specifieke terminologie waarin relaties gedefinieerd zijn tussen de termen uit het domein. Door hiervan een formele beschrijving te geven, kan het model gebruikt worden in samenspraak met informatiesystemen [1].

Een ontologie kent verschillende toepassingsdomeinen, waaronder (geciteerd uit [21]):

Ontology-based search This is useful for discovery and selection of information or components.

Neutral authoring Especially helpful in cases of data exchange among systems.

Common access to information A much stronger case than Neutral Authoring, this allows meaning to be transmitted, not just matched against.

Ontology as specification Useful for describing the meaning of a domain, to guide system development.

In de literatuur werden onder andere voorbeelden teruggevonden van *Ontology-based search* [20] en *Neutral authoring* [22]. De ontologieën uit deze studie vallen onder *Ontology as specification*. Ze worden gebruikt om de omgeving en het domein (de bedrijfssector, de werknemers, etc) te beschrijven. De beslissingen van het systeem zijn op deze kennis gebaseerd [15].

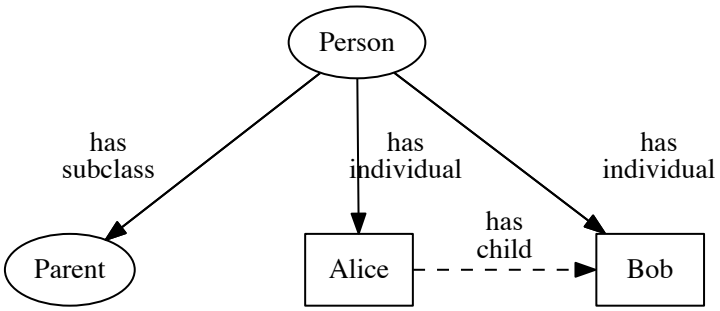
Opbouw van een ontologie

Een ontologie wordt, zoals hiervoor beschreven, gebuikt om een specificatie te geven van informatie uit een specifiek domein. In dit model beschrijft men entiteiten (bv. `Persoon`, `Ouder`, `Kind`). Elk van deze entiteiten kan verschillende eigenschappen hebben en er kunnen relaties gedefinieerd worden tussen de verschillende entiteiten uit het domein (bv. een `Ouder` heeft een relatie met zijn/haar `Kind` genaamd `heeftKind`). De verzameling van entiteiten met hun onderlinge relaties wordt de *T-Box* van de ontologie genoemd [5]. Het geeft een beschrijving van de verschillende soorten objecten en relaties uit het domein.

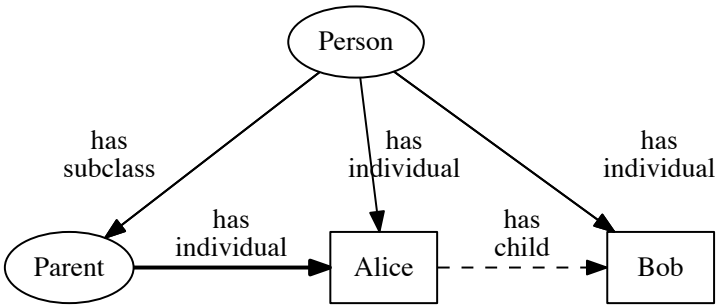
Vervolgens worden er individuen toegevoegd aan dit model die een realisatie zijn van de entiteiten in het model (bv. `Alice` is individu van het type `Persoon`). Deze individuen, samen met hun onderlinge individuspecifieke relaties, vormen de *A-Box* van het kennismodel [5].

Axioma's (de *R-Box*) vervolledigen het domeinmodel met uitdrukkingen die handelen over de elementen in de ontologie. Ze beschrijven kennis die niet expliciet beschreven staat in de ontologie, maar wel bekomen kan worden door de axioma's toe te passen op de kennis in de ontologie.

Dit soort kennis wordt ook de *inferred knowledge* van het model genoemd (dit staat tegenover de *asserted knowledge* die *wel* expliciet aanwezig is in het model). Een voorbeeld van een axioma: elke `Persoon` met een relatie `heeftKind` is tevens een `Ouder` (zie Figuur 2.1 voor een illustratie). Axioma's zijn een vorm van beschrijvende regels die verder besproken worden in paragraaf 2.1.2. Software die dergelijke uitdrukkingen kan interpreteren, wordt een *reasoner* genoemd.



(a) Asserted knowledge.



(b) Inferred knowledge.

Figuur 2.1: Verschil tussen de asserted knowledge en de inferred knowledge van een ontologie na het toepassen van het eenvoudige axioma: elke `Persoon` met een relatie `heeftKind` is tevens een `Ouder`.

2.1.2 Regels

Regeltalen worden gebruikt in kennisgebaseerde systemen om kennis te induceren uit een reeds bestaand kennismodel [12] (bv. een ontologie). De regels maken gebruik van de entiteiten, individuen en relaties in het kennismodel om op basis daarvan de nieuwe kennis te induceren. Bijvoorbeeld: 'Als X een ouder is van Y en Y is een ouder van Z, dan is X een grootouder van Z'.

In het algemeen worden regels beschouwd als uitdrukkingen met een bepaalde vorm[12]:

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n$$

Elke A_i is een atomair gegeven dat bestaat uit een predicaat p met nul, één of meer termen t_i als argument (bv. $p(t_0, \ldots, t_{n-1})$). Een term kan drie vormen aannemen: een constante, een

variabele of een functie met opnieuw termen als argumenten. De constanten verwijzen naar bronnen beschreven in het kennismodel. Een variabele kan gebruikt worden als *placeholder* om algemene regels te schrijven die van toepassing zijn op meerdere constanten. Ze worden genoteerd met een vraagteken (bv. $?var$). Functies worden gebruikt om informatie te construeren op basis van bestaande kennis. De functie $geefJaar(?datum)$ kan bijvoorbeeld gebruikt worden om het jaar te extraheren uit een datum. Variabelen noch constanten hebben een betekenis voor het systeem. Het zijn de regels die deze termen gebruiken die ervoor zorgen dat hierover geredeneerd kan worden.

Een typische regel bestaat uit twee delen: een antecedent (de conditie, Eng. *rule head*) en een consequent (de nieuwe kennis, Eng. *rule body*). Regels zonder antecedent worden feiten genoemd. Een formeel voorbeeld van de regel 'Als X een ouder is van Y en Y is een ouder van Z, dan is X een grootouder van Z' wordt gegeven door:

$$grandparent(?X,\ ?Z) \leftarrow parent(?X,\ ?Y),\ parent(?Y,\ ?Z)$$

Deze regel bevat drie atomaire gegevens: $grandparent(?X,\ ?Z)$, $parent(?X,\ ?Y)$ en $parent(?Y,\ ?Z)$. Ze maken gebruik van twee verschillende predicaten (*grandparent* en *parent*) en drie verschillende termen ($?X$, $?Y$ en $?Z$), in dit geval variabelen.

Zoals aangehaald kan een kennisgebaseerd systeem de regels gebruiken om beslissingen te nemen op basis van het kennismodel. Voor elke beslissing worden hiertoe de overeenkomstige regels overlopen, tot het antecedent van een regel werd gevonden in het kennismodel. Het resultaat (het consequent van de regel) wordt dan opgenomen in dit kennismodel of gebruikt als basis voor verdere beslissingen.

De regels die tot nu toe beschreven zijn, worden *beschrijvende regels* genoemd. Ze introduceren nieuwe kennis indien indien een conditie waar is. Een tweede soort regels zijn *productieregels*. Zij voorzien een bepaalde actie die uitgevoerd dient te worden indien een voorwaarde voldaan is. Deze actie kan feiten toevoegen, wijzigen of verwijderen, of een actie zijn die extern gedefinieerd werd.

2.1.3 Specificaties

Er zijn verschillende specificaties waarmee men ontologieën en regels kan beschrijven. Elke specificatie heeft zijn voor- en nadelen die worden afgewogen afhankelijk van projectspecifieke eigenschappen. SWRL [23] is bijvoorbeeld een regeltaal die werd ontwikkeld als uitbereiding voor OWL [3], Jena Rules laten toe om zowel *forward rules*, *backward rules* als *hybrid rules* te modelleren [2] en RIF-Core werd speciaal ontwikkeld als tussentaal bij omzettingen naar andere

regeltalen[28]. In het belang van deze studie worden hieronder de specificaties toegelicht die werden gebruikt in het implementatiegedeelte van deze studie.

Resource Description Framework

Een van de meest algemene manieren om informatie voor te stellen is met behulp van het *Resource Description Framework* (RDF). Het is een framework om informatie over bepaalde bronnen weer te geven [35, 34]. De definitie voor bron staat hier vrij voor interpretatie (documenten, mensen, objecten, concepten). Het doel is om informatie te kunnen delen tussen verschillende applicaties, zonder dat de betekenis ervan verloren gaat. RDF kan ook gebruikt worden in applicaties die gebruik maken van *Linked Data*. Een bepaalde RDF-graaf verwijst dan naar een bron die gedefinieerd wordt in een andere graaf. Dit mechanisme laat toe om een gedistribueerd kennismodel te maken, zonder elke afzonderlijke component te kennen.

Een RDF-document bestaat uit een reeks RDF-uitdrukkingen met een vaste vorm. Ze bestaan steeds uit drie componenten:

```
<onderwerp> <predicaat> <voorwerp>
```

Per uitdrukking wordt een relatie beschreven tussen het onderwerp en het voorwerp, beide een bron. De aard van de relatie wordt beschreven met het predicaat en is steeds directioneel. Verschillende uitdrukkingen of *triples* worden samen een RDF-graaf genoemd en kunnen worden voorgesteld door een directionele geëtiketteerde multigraaf. Om de componenten van een RDF-uitdrukking te beschrijven maakt men gebruik van drie soorten RDF-data: *International Resource Identifiers* (IRI's), *literals* en *blank nodes*.

Een IRI[1] identificeert een bron of relatie op een unieke manier. Deze identificatie kan gebruikt worden in externe grafen om te verwijzen naar de bron. Literals worden enkel gebruikt voor het voorwerp. Ze representeren nog steeds een bron, maar kunnen niet gebruikt worden buiten de graaf. Literals kunnen verschillende soorten waarden aannemen zoals tekst, een getal, een datum of een logische waarde. Als laatste maakt men gebruik van blank nodes om relaties te beschrijven, zonder de bron ervan te noemen. Codevoorbeeld 2.1 geeft een illustratie van een RDF-document.

Web Ontology Language

Zoals vermeld is RDF een zeer algemene vorm voor het beschrijven van data. Voor bepaalde doeleinden, zoals het gebruik van kennissystemen, is dit niet voldoende en eist men een meer

[1]Een IRI is een *Uniform Resource Identifier* (URI) waarin het gebruik van non-ASCII karakters is toegelaten.

```
<http://example#Bob> <http://example/people#hasAge> 25
<http://example#Alice> <http://example/people#parent> <http://example#Bob>
<http://example#Bob> <http://example/people#parent> <http://example#Charly>
```

Codevoorbeeld 2.1: Een voorbeeld van een RDF-document.

expressieve taal. *Web Ontology Language* (OWL)[2] is een taal die specifiek ontwikkeld is voor het beschrijven van ontologieën. Hoewel een OWL-ontologie volledig uit te drukken is in RDF, definieert ze specifieke constructies om de extra functionaliteit mogelijk te maken [26, 25, 24]. Applicaties kunnen deze constructies gebruiken om een diepere kennis te bekomen over het domein. Bijvoorbeeld: in [14] wordt een OWL-ontologie gebruikt om verschillende aspecten van de zorgsector te modelleren.

OWL maakt gebruik van axioma's, entiteiten en uitdrukkingen voor het beschrijven van een ontologie. De axioma's zijn uitdrukkingen die feiten beschrijven over het domein. De entiteiten zijn een weerspiegeling van objecten uit het domein. De uitdrukkingen worden gebruikt om complexere entiteiten te vormen met behulp van verschillende andere entiteiten. Bijvoorbeeld: de conjunctie van de klassen *fietser* en *muzikant* kan gebruikt worden om alle fietsende muzikanten te beschrijven [24].
OWL-entiteiten zijn een weerspiegeling van individuen, klassen en/of attributen. De klassen worden gebruikt om gelijksoortige individuen te klasseren. De attributen definiëren metadata over de individuen. Individuen zijn concrete voorbeelden van objecten uit een bepaalde klasse.
Op basis van de kennis in een OWL-ontologie kan er geredeneerd worden. Verschillende OWL-uitdrukkingen kunnen namelijk andere uitdrukkingen insinueren. Hierdoor kan men gevolgen trekken op basis van bestaande uitdrukkingen in de ontologie. Applicaties die deze redeneringen kunnen maken worden *reasoners* genoemd.

Rule Interchange Format

Een regeltaal die ontwikkeld werd door het *World Wide Web Consortium* (W3C) is het *Rule Interchange Format* (RIF) [30, 31]. De taal is verschillend van overige regeltalen omdat ze speciaal ontwikkeld werd om als tussentaal te dienen voor omzettingen naar andere regeltalen. RIF is eigenlijk een verzameling van verschillende dialecten, waarvan er momenteel drie zijn opgenomen in de W3C aanbeveling: *RIF Basic Logic Dialect* (RIF-BLD), *RIF Production Rule Dialect* (RIF-PRD) en RIF-Core. Elk dialect voorziet bepaalde constructies die gebruikt worden voor

[2] 'The acronym OWL was proposed as an easily pronounced acronym that would yield good logos, suggest wisdom, and honor William A. Martin's One World Language knowledge representation project from the 1970s.' (bron: `https://en.wikipedia.org/wiki/Web_Ontology_Language` (laatst geraadpleegd op 15/05/15))

specifieke soorten regels. RIF-Core is hierbij een gemeenschappelijke subset van de constructies uit RIF-BLD en RIF-PRD. RIF-BLD breidt RIF-Core uit met functies, gelijkheid, gesloten lijsten en meer. Dit dialect wordt gebruikt voor het beschrijven van *logic programs*, waarbij nieuwe kennis geconcludeerd kan worden op basis van reeds bestaande kennis. RIF-PRD is bedoeld voor het beschrijven van productieregels (gebruikt in actieve regelbanken). Andere RIF-dialecten kunnen hieraan toegevoegd worden. Speciaal hiervoor werd het *RIF Framework for Logic Dialects* (RIF-FLD) dat hiervoor de nodige concepten beschrijft [29].

Schrijfwijzen Er zijn drie schrijfwijzen gedefinieerd voor het noteren van RIF. De standaard schrijfwijze voor alle dialecten maakt gebruik van *eXtensible Markup Language* (XML) en wordt vooral gebruikt in softwaretoepassingen. Voor de voorbeelden in deze thesis wordt de *Presentation Syntax* gebruikt die bedoeld is voor RIF-BLD (bv. `B :- A`). Een aparte schrijfwijze werd ontwikkeld voor het RIF-PRD-dialect: de *Abstract Syntax* (bv. `If A Then B`).

RIF-bouwstenen RIF voorziet, afhankelijk van het gebruikte dialect[3], constructies voor alle onderdelen van een regeltaal zoals beschreven in paragraaf 2.1.2. Voor het noteren van constanten wordt er beroep gedaan op de IRI-notatie die terug te vinden is bij RDF. Het gebruik van prefixen en een standaard prefix laat toe dat de IRI's afgekort kunnen worden.

Variabelen worden gedefinieerd met behulp van *quantifiers*. Er zijn twee soorten quantifiers: `ForAll`, de universele quantifier, en `Exists`, de existentiële quantifier. Ze worden respectievelijk gebruikt om een regel te schrijven over meerdere of ten minste één bron uit het kennismodel.

Bijkomende constructies worden gebruikt om verschillende regels te groeperen en samen te voegen tot een document. Documenten kunnen vervolgens geïmporteerd worden in andere documenten. Codevoorbeeld 2.2 geeft een voorbeeld van een RIF-document.

SPARQL Protocol and RDF Query Language

SPARQL Protocol And RDF Query Language (SPARQL) is geen regeltaal, noch een taal voor het beschrijven van ontologieën. Wel is het een verzameling van talen en protocollen voor het bevragen (Eng. query) en het wijzigen van een RDF-graaf [32]. De SPARQL-specificaties voorzien hiervoor structuren die sterk lijken op constructies uit standaard SQL-talen die gebruikt worden in relationele databank systemen. De twee talen die SPARQL specificeert zijn SPARQL 1.1 Query Language en SPARQL 1.1 Update Language. Zij worden besproken in de volgende paragrafen.

[3]Bijvoorbeeld: negatie wordt wél ondersteund in RIF-PRD, maar niet in RIF-BLD.

```
Document(
    Base(<http://example#>)
    Prefix(people <http://example/people#>)

    Group(
        Forall ?X ?Y ?Z (
            people:grandparent(?X ?Z) :-
                And(people:parent(?X ?Y) people:parent(?Y ?Z))
        )

        people:parent(<Alice> <Bob>)
        people:parent(<Bob> <Charly>)
    )
)
```

Codevoorbeeld 2.2: Voorbeeld van een RIF-document.

SPARQL 1.1 Query Language Om gegevens uit een RDF-graaf te extraheren, wordt er gebruik gemaakt van standaard SPARQL-uitdrukkingen (Eng. *SPARQL-query*). De gevraagde gegevens (een deelgraaf van de bevraagde RDF-graaf) worden beschreven met behulp van een patroon, het *query pattern*. De gegevens die voldoen aan het patroon, worden gebruikt om het resultaat te construeren.

Het patroon zelf maakt gebruik van de triple-notation van RDF. Hierin worden bronnen geïdentificeerd via hun (eventueel afgekorte) IRI. Variabelen worden gebruikt op de plaats waar men geen exacte gelijkenis wenst. Meer complexe uitdrukkingen kan men bekomen door gebruik te maken van *expressions* en *filters*. Ze maken het mogelijk om data te manipuleren en voorwaarden op te leggen aan de waarde van variabelen.

Een verzameling van SPARQL-triples met eventueel expressions en filters wordt een *Basic Graph Pattern* (BGP) genoemd. De oplossingen van meerdere BGP's kunnen samengevoegd worden door gebruik te maken van een UNION-constructie.

De SPARQL 1.1 Query Language voorziet vier verschillende soorten uitdrukkingen die elk op een andere manier een resultaat construeren:

SELECT Dit type query geeft de waarden terug die ingevuld werden in de variabelen uit het query pattern. Door de waarden te substitueren in het query pattern, bekomt men een deelgraaf van de bevraagde RDF-graaf. De SELECT query is het meest voorkomende type SPARQL-query.

Codevoorbeeld 2.3 geeft een voorbeeld van een eenvoudige SPARQL SELECT query.

ASK Een ASK query wordt gebruikt om te controleren of er voor het opgegeven query pattern een oplossing gevonden kan worden. Indien de bevraagde RDF-graaf een deelgraaf gevat die voldoet aan het patroon, zal de logische waarde *waar* worden teruggegeven.

CONSTRUCT Bij CONSTRUCT queries worden de waarden van de variabelen gesubstitueerd in een opgegeven sjabloon. Alle oplossingen samen worden vervolgens gecombineerd in een nieuwe RDF-graaf die het resultaat op de query vormt.

DESCRIBE Om een beschrijving te krijgen van bronnen die voldoen aan het patroon, kan er gebruik gemaakt worden van de DESCRIBE query. Een eenvoudig voorbeeld wordt gegeven in Codevoorbeeld 2.4.

```
PREFIX people: <http://example/people#>
SELECT ?X ?Y ?Z
WHERE
{
    ?X people:parent ?Y .
    ?Y people:parent ?Z .
}
```

Codevoorbeeld 2.3: Voorbeeld van een eenvoudige SPARQL SELECT query.

```
PREFIX people: <http://example/people#>
DESCRIBE ?Z
WHERE
{
    ?X people:parent ?Y .
    ?Y people:parent ?Z .
    ?Z people:birthDate ?birth .
    FILTER (year(?birth) < 1950)
}
```

Codevoorbeeld 2.4: Voorbeeld van een eenvoudige SPARQL DESCRIBE query.

SPARQL 1.1 Update Language SPARQL 1.1 Update is een aanbeveling van het W3C voor het beschrijven en het uitvoeren van aanpassingen aan een RDF-graaf. De taal hergebruikt de

meeste constructies die gedefinieerd zijn in de SPARQL 1.1 Query Language en voegt hieraan extra functionaliteit toe.
Er worden twee soorten updates beschreven: enerzijds zijn er *Graph Management* queries die het mogelijk maken om grafen te creeëren, verwijderen, verplaatsten of te kopiëren naar een andere graaf. Anderzijds zijn er *Graph Update* queries. Zij voorzien functionaliteit om de gegevens uit een enkele graaf aan te passen. Voorbeelden hiervan zijn het verwijderen van gegevens die voldoen aan een bepaald patroon of het toevoegen van gegevens op basis van een bepaald sjabloon. Codevoorbeeld 2.5 toont een voorbeeld van een SPARQL 1.1 Update query.

```
PREFIX people: <http://example/people#>
INSERT
{
    ?X people:grandparent ?Z .
}
WHERE
{
    ?X people:parent ?Y .
    ?Y people:parent ?Z .
}
```

Codevoorbeeld 2.5: Voorbeeld van een eenvoudige SPARQL 1.1 Update query.

2.2 Discrete event simulatie

In het algemeen worden simulaties gebruikt om het gedrag van een bepaald systeem te analyseren in specifieke situaties. Afhankelijk van het type systeem wordt er een bepaalde soort van simulatie voorgeschreven. Enerzijds zijn er systemen wiens toestand beschreven wordt door continue veranderlijken (bv. het weer). Anderzijds bestaan er systemen wiens toestand op discrete tijdstippen verandert. Het zijn deze discrete systemen die men zal simuleren met behulp van Discrete Event Simulatie (DES) [13].

Een discrete event simulator beschrijft een systeem aan de hand van een reeks gebeurtenissen die zich voordoen op een specifiek tijdstip. De simulaties bestaan uit het afhandelen van deze gebeurtenissen op een chronologische manier. Doordat elke gebeurtenis zich voordoet op een specifiek tijdstip, wordt er verondersteld dat tussen deze tijdstippen de toestand van het systeem constant blijft. Hierdoor kan een discrete event simulator springen in de tijd naar de eerstvolgende gebeurtenis van het systeem. Discrete event simulaties werken daardoor veel sneller dan continue tegenhangers.

Een belangrijke toepassing van DES is de analyse op het gebruik van bronnen. Bronnen zijn, zoals reeds beschreven, dingen die slechts beperkt beschikbaar zijn in het systeem. Ze kunnen toegekend worden aan een gebeurtenis in het systeem. Hierdoor wordt de uitvoering van een gebeurtenis uitgesteld indien niet alle bronnen beschikbaar zijn. De manier van toekennen en het aantal bronnen kan bijgevolg bepalend zijn voor de efficiëntie van een systeem.

2.2.1 Processimulatie

DES kan gebruikt worden voor de simulatie van bedrijfsprocessen. In deze paragraaf wordt beschreven wat processen zijn en hoe zij gesimuleerd kunnen worden.

Processen

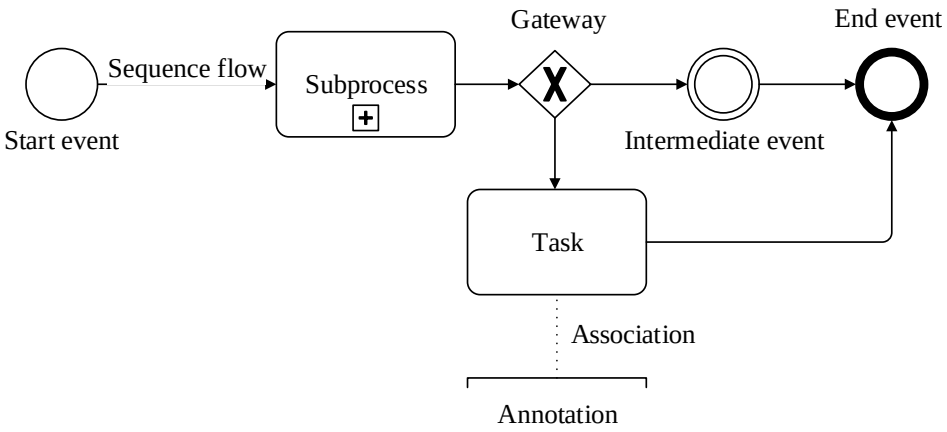
Een proces is een stappenplan van taken dat omschrijft hoe iets moet uitgevoerd worden. Per taak dienen de benodigde bronnen worden toegekend die vereist zijn voor de uitvoering van de taak. Om een proces formeel te definiëren maakt men gebruik van specifieke talen. *Business Process Modeling Notation* (BPMN) is een taal die hier speciaal voor ontwikkeld werd. Ze voorziet zowel in een grafische voorstelling als in een formele procesdefinitie (via een XML-notatie). BPMN modellen kunnen daardoor rechtstreeks gebruikt worden in computersystemen (bv. processimulaties).

De procesmodellen die gebruikt werden in deze thesis zijn beschreven in BPMN 2.0. Er wordt daarom een beknopt overzicht gegeven van de verschillende basiselementen uit deze taal[18]. Een BPMN-procesmodel bestaat uit een reeks taken (Eng. tasks) die onderling verbonden zijn via een *sequence flow*. Het begin en het einde van het proces wordt expliciet aangegeven met een *start* en *end event*[4]. *Intermediate events* worden gebruikt indien er interactie is met structuren waarover men geen controle heeft binnen het bedrijfsproces (bv. indien een bericht ontvangen dient te worden van een derde partij). De *flow* van het proces kan gesplitst worden met een *gateway*. Dit wordt bijvoorbeeld gebruikt indien taken parallel uitgevoerd kunnen worden of indien een specifiek pad gevolgd dient te worden naargelang de situatie. Taken die uit meerdere deeltaken bestaan kunnen gemodelleerd worden als een subproces. Bij de uitvoering van deze taak zal het overeenkomstige subproces worden opgestart en doorlopen worden. Ten slotte kunnen er annotaties worden toegevoegd aan het procesmodel (bv. om een situatie te verduidelijken). Een illustratie van de verschillende basiselementen is terug te vinden in Figuur 2.2.

Merk op dat BPMN geen manier voorziet voor het modelleren van bronnen in een proces. Om visueel aan te geven dat de uitvoering van een bepaalde taak bronnen vereist, worden in deze

[4]Meerdere *start* en *end events* per proces zijn mogelijk, maar dit wordt in deze studie niet gebruikt.

studie annotaties gebruikt.



Figuur 2.2: Een voorbeeld van een procesmodel in BPMN.

Processimulaties

Processimulaties worden gebruikt om meer inzicht te krijgen in de uitvoering van een proces. Formele modellen, bv. beschreven met BPMN, maken de overgang naar DES minder complex. De uitvoering van een taak in het proces wordt vertaald naar een start en eind gebeurtenis (op basis van de duur van de taak). Ook het gebruik van bronnen in een proces kan volledig vertaald worden naar de gebruiken in DES[5].

Tijdens simulaties worden verschillende meetwaarden bijgehouden zoals de duur van een taak, de wachttijd op een bron, etc. Deze waarden (Eng. *Performance Indicators* (PI's)) kunnen vervolgens geaggregeerd worden tot KPI's. Het zijn deze KPI's die uiteindelijk gebruikt worden als meetwaarden voor efficiëntie van een proces. Door bepaalde eigenschappen van het proces te wijzigen, kan men met processimulatie nagaan wat de invloed ervan is op voorgedefinieerde KPI's.

2.2.2 Processimulator van IBCN

Een specifieke implementatie van een processimulator werd verder ontwikkeld bij IBCN in het kader van het HIPS-project[6][7]. Deze simulator is in staat om procesmodellen te simuleren die beschreven zijn in BPMN. De simulator laat bijkomende functionaliteit toe, zoals toekennen van bronnen aan taken in het proces en het toekennen van een kost aan zowel taken als bronnen

[5]Hiervoor kan er geen beroep worden gedaan op BPMN.

[6]HIPS, Innovation through coordinated and optimized Patient, Supply and information flows in Hospitals, `https://www.iminds.be/hips` (laatst geraadpleegd op 05/06/15)

in het proces. Deze informatie wordt onder andere gebruikt voor het berekenen van bepaalde performance indicators.

Het doel van de simulator is om op basis van KPI's van een proces mogelijkheden tot verbetering te identificeren. Bijkomend kunnen *bottlenecks* in het proces achterhaald worden die een grote negatieve invloed hebben op de efficiëntie van een proces. De ontwikkeling van een eigen simulator laat toe dat er eenvoudig aanpassingen gemaakt kunnen worden die vereist zijn voor specifiek onderzoek.

Al tijdens het simuleren worden analyses uitgevoerd op de uitvoeringen van de verschillende procesinstanties. Op basis van deze analyses wordt een speciale stopvoorwaarde gedefinieerd: indien alle uitgevoerde analyses (bv. gemiddelde duur van een taak, keuze bij een gateway, etc) zijn geconvergeerd, stop de simulatie. Dit verzekerd op zekere hoogte dat de PI's van het proces op een representatieve manier werden berekend.

Hoofdstuk 3

Architectuur

In dit hoofdstuk wordt het architecturaal model besproken dat ontwikkeld werd gedurende de thesis. Allereerst zullen de verschillende eisen worden voorgesteld die bepalend zijn voor het model en de implementatie. De niet-functionele eisen worden toegelicht in paragraaf 3.1. Zij vormden de basis voor toekomstige beslissingen. Vervolgens worden de functionele eisen toegelicht die meer gericht zijn naar de implementatie van het model (zie paragraaf 3.2).

3.1 Niet-functionele eisen

3.1.1 Generiek systeem

Zoals vermeld in de literatuurstudie bestaat een kennisgebaseerd systeem uit twee onderdelen: een kennisbasis en een regelset. Afhankelijk van projectspecifieke eigenschappen zal een bepaalde implementatie voor beide componenten worden gekozen. De ene implementatie voorziet in een meer expressieve taal (bv. OWL Full), terwijl de andere meer geoptimaliseerd is voor snelheid (bv. OWL DL) [22]. Dit zorgt voor een grote verscheidenheid in de implementaties van kennisgebaseerde systemen.

Een van de vereisten voor deze thesis, is het voorzien van de nodige functionaliteit om deze verscheidenheid op te vangen. Dit houdt in dat de te ontwikkelen componenten zo veel mogelijk talen dienen te ondersteunen, zonder hiervoor specifieke functionaliteit te voorzien.

3.1.2 Eenvoudige optimalisatie

Het doel van de simulaties in deze thesis is tweeledig. Enerzijds wenst men het effect van een kennisgebaseerd systeem op de KPI's van een proces te analyseren en te verbeteren. Anderzijds wenst men het systeem te verbeteren, afhankelijk van de situatie waarin het gebruikt zou worden. Deze laatste soort is *case*-afhankelijk en moet bijgevolg manueel gebeuren. Deze architecturale

eis houdt in dat de nodige functionaliteit moet worden voorzien om op een eenvoudige manier regels aan te passen en deze te voeden aan de simulator.

3.2 Functionele bouwblokken

Alvorens een architecturaal model voor te leggen, worden de functionele eisen toegelicht. Ze leggen vast wat er verwacht kan worden van het uiteindelijke systeem. De te ontwikkelen architectuur wordt hier nog beschouwd als een *black box*. Om dit te illustreren werd een *Use Case Diagram* gemaakt (Figuur 3.1).

3.2.1 Gebruik van kennisgebaseerde systemen voor het toekennen van bronnen

In de technologiestudie werd uitgelegd wat het betekent om bronnen toe te kennen aan taken in een proces. De manier waarop een bron wordt toegekend, wordt meestal niet beschreven. Dit komt omdat een DES gewoonlijk de eerst beschikbare bron uit een lijst toekent, zonder bijkomende selectiecriteria.

Deze functionaliteit dient te worden uitbreid. Het moet mogelijk worden om aan te geven hoe een bron wordt toegekend in de simulator. Een van de mogelijkheden dient het gebruik van een kennisgebaseerd systeem te zijn. De aanpassing moet er voor zorgen dat reeds bestaande procesbeschrijvingen nog steeds bruikbaar zijn met de gebruikte simulatiesoftware.

3.2.2 Opvragen van bronnen

De simulator moet in staat zijn een bron te vragen aan een ontologiegebaseerd systeem. Bij het toekennen van de bron wordt verondersteld dat deze wordt teruggegeven in het formaat dat gebruik wordt door de simulator. Daarbij mag de bron niet meer beschikbaar zijn voor toekomstige toekenningen, tot de bron weer wordt vrijgegeven.

Het moet mogelijk zijn om meerdere types bronnen toe te kennen op deze manier. Ze worden dan toegekend met elk hun eigen regelsets die mogelijks inwerken op dezelfde ontologie. Deze ontologie wordt bijgevolg gedeeld onder de verschillende soorten bronnen.

3.2.3 Vrijgeven van bronnen

De simulator moet in staat zijn een bron terug vrij te geven aan het ontologie gebaseerd systeem. Bij het vrijgeven van de bron wordt verondersteld dat deze wordt afgeleverd in het formaat dat gebruik wordt door de simulator. Daarbij wordt de bron terug beschikbaar geacht voor toekomstige aanvragen.

Indien meerdere bronnen worden toegekend met behulp van een ontologie, dan moet het mogelijk zijn om eenzelfde ontologie te delen.

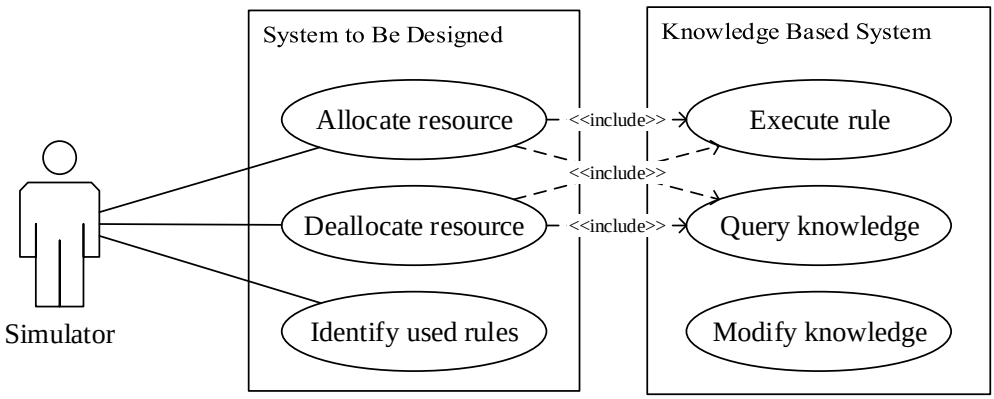
3.2.4 Regel identificatie

Zoals reeds aangehaald in de technologiestudie zal het kennisgebaseerd systeem gebruikt worden voor het toekennen van bronnen. Per type bron wordt een regelset aangeduid voor de toekenning ervan. De regels uit de set worden overlopen en uitgevoerd totdat een bron kan worden toegekend. Er is bijgevolg slechts één regel uit de set die zorgt voor de effectieve toekenning.

Naar analyse doeleinden toe is het interessant om te achterhalen welke regels resulteerden in de toekenning van de bron. Het is daarom een eis naar het ontwerp om hiervoor de nodige functionaliteit te voorzien.

3.2.5 Losse koppeling

De simulator mag niet op de hoogte zijn van de achterliggende technologie die gebruikt wordt voor het beheren van bronnen. Dit impliceert een losse koppeling tussen de simulator enerzijds en het kennisgebaseerd systeem anderzijds. Om dit mogelijk te maken dienen beide systemen te worden aangesproken via hun eigen, reeds gedefinieerde interface.



Figuur 3.1: Het use case diagram dat de verschillende taken van het te ontwikkelen systeem weergeeft.

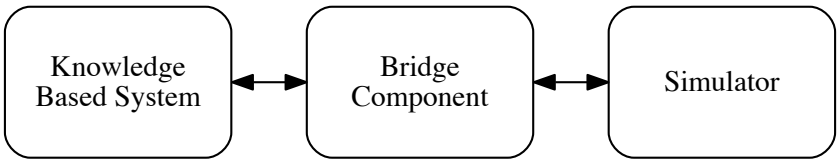
3.3 Architecturaal model

Deze paragraaf bespreekt het model van de architectuur dat werd ontwikkeld. Dit model wordt in twee delen besproken waarbij telkens dieper wordt ingegaan op bepaalde componenten. Het

hoogste niveau van abstractie wordt besproken in paragraaf 3.3.1. Hierna volgt een aparte bespreking voor elke hoofdcomponent uit het abstracte model (paragrafen 3.3.2, 3.3.3 en 3.3.4).

3.3.1 Hoofdcomponenten

Het architecturaal model bestaat uit drie hoofdcomponenten: het kennisgebaseerd systeem, een brugcomponent en de simulator. Elke component communiceert enkel met zijn naaste. Hierdoor kunnen het kennisgebaseerd systeem en de simulator onafhankelijk van elkaar ontwikkeld worden. Een illustratie van dit abstract model wordt weergeven door Figuur 3.2.



Figuur 3.2: De drie hoofdcomponenten van het architecturaal model.

3.3.2 Kennisgebaseerd systeem

Het kennisgebaseerd systeem is de eerste component die wordt besproken. Het is dit systeem dat gebruik zal worden voor de allocatie en het vrijgeven van bronnen. Zoals eerder uitgelegd bestaat het uit twee onderdelen: een kennismodel en regels.

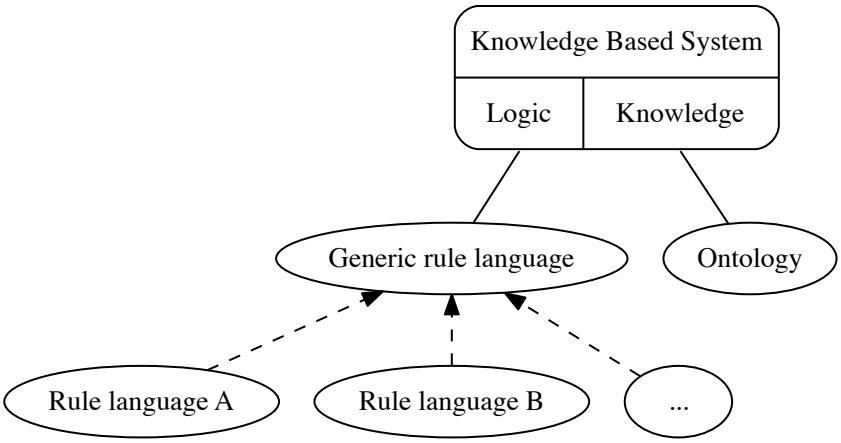
Er zal een ontologie gebruikt worden als kennismodel. Hierin worden zowel de bronnen als de domeinkennis gemodelleerd. Het gebruik van een ontologie laat toe dat ook complexere systemen gebruikt kunnen worden voor de toekenning van bronnen.

Regels kunnen in verschillende regeltalen geschreven worden. Een generiek systeem (paragraaf 3.1.1) zou al deze verschillende talen moeten ondersteunen. In dit model wordt dit opgevangen door gebruik te maken van één generieke regeltaal die een algemene structuur bezit. Hierdoor wordt het eenvoudig om andere talen om te zetten naar de generieke regeltaal. Deze omzetting kan naar de toekomst toe geautomatiseerd worden, maar valt buiten het kader van deze thesis.

Figuur 3.3 illustreert de opbouw van het kennisgebaseerd systeem.

3.3.3 Simulator

De simulator is de component die procesmodellen interpreteert en simuleert volgens de principes van discrete event simulatie. Tijdens een simulatie worden stap voor stap alle taken uit



Figuur 3.3: De verschillende onderdelen van het kennisgebaseerd systeem.

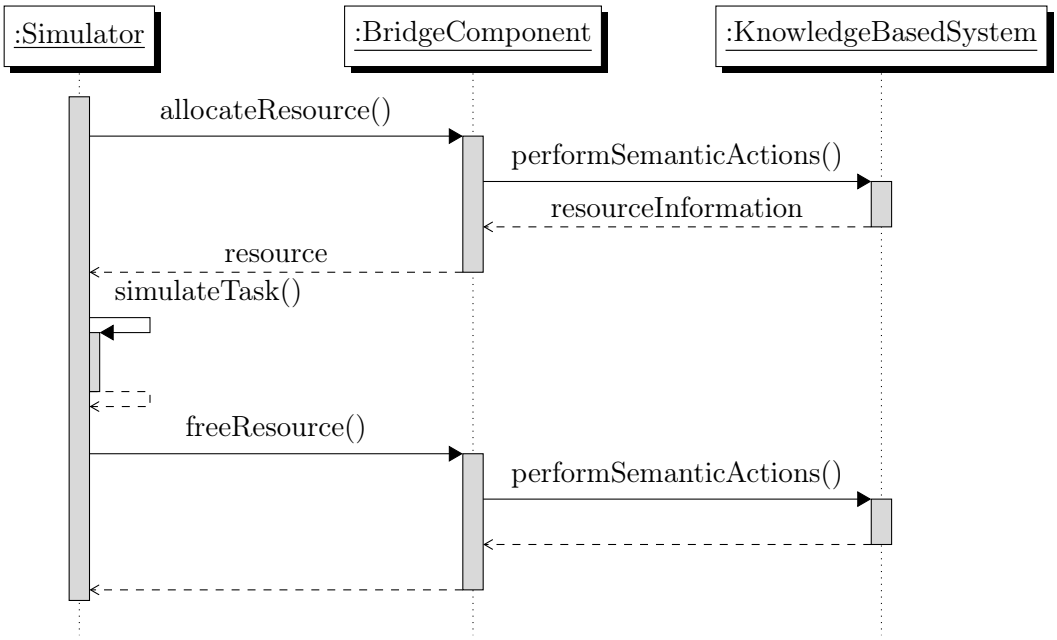
het proces overlopen. Per taak worden de nodige bronnen gealloceerd alvorens deze taak kan worden uitgevoerd. De simulator zal deze bronnen opvragen aan de brugcomponent. Indien de brugcomponent geen bron kan voorzien, zal uitvoering van de taak uitgesteld worden.

Na de afloop van een taak worden de gebruikte bronnen teruggegeven aan de brugcomponent. Ze worden dan terug als beschikbaar beschouwd.

3.3.4 Brugcomponent

Een van de kwaliteitseisen is dat er een losse koppeling heerst tussen het kennisgebaseerd systeem en de simulator. Dit laat toe dat beide systemen onafhankelijk van elkaar ontwikkeld kunnen worden. De brugcomponent maakt dit mogelijk door de communicatie tussen beide systemen op zich te nemen.

Indien een bron nodig is tijdens de simulatie zal dit door de simulator aan de brugcomponent gevraagd worden. Vervolgens wordt het kennisgebaseerd systeem aangesproken door de brugcomponent. De regels die horen bij het gevraagde type bron zullen worden uitgevoerd op de kennisbasis totdat een bron wordt gevonden. De gealloceerde bron wordt vervolgens omgezet waarna ze wordt teruggegeven aan de simulator. Figuur 3.4 illustreert hoe deze communicatie zou verlopen.



Figuur 3.4: De communicatie tussen de drie hoofdcomponenten.

Hoofdstuk 4

Implementatie

In dit hoofdstuk wordt de implementatie van de architectuur besproken. Een eerste deel (paragraaf 4.1) licht de verschillende ontwerpbeslissingen toe. Zij geven aan hoe de implementatie aansluit bij de architectuur. Vervolgens wordt er voor elk onderdeel een meer gedetailleerde uitleg voorzien (paragraaf 4.2).

4.1 Ontwerpbeslissingen

In deze paragraaf worden de ontwerpbeslissingen besproken per onderdeel van het architecturaal model.

4.1.1 Kennisgebaseerd systeem

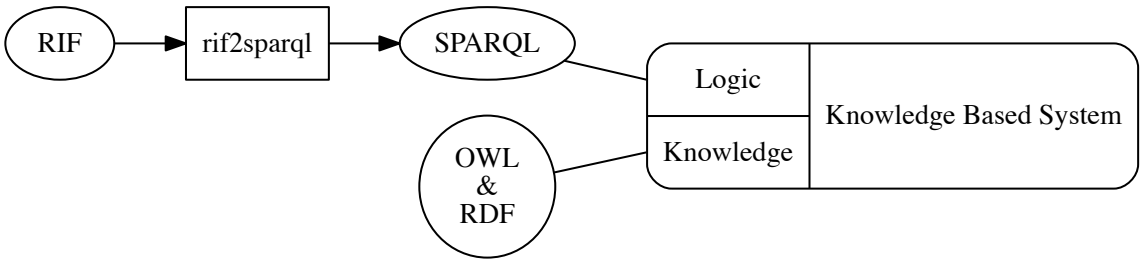
Net als het kennis gebaseerd systeem uit twee componenten bestaat, moeten er twee fundamentele beslissingen genomen worden:

- Welke taal zal gebruikt worden als generieke regeltaal?
- Welke taal zal ondersteund worden voor ontologieën?

Elk van deze beslissingen wordt behandeld in de overeenkomstige paragrafen en geïllustreerd in Figuur 4.1.

Generieke regeltaal

De eerste component uit het architecturaal model is de generieke regeltaal. In de implementatie werd ervoor gekozen om RIF te gebruiken. Het is een regeltaal die specifiek als tussentaal ontwikkeld werd (paragraaf 2.1.3). Dit maakt RIF uitermate geschikt als generieke component in de architectuur.



Figuur 4.1: De verschillende ontwerpbeslissingen voor het kennisgebaseerd systeem.

Het gebruik van RIF heeft ook nadelen. Omwille van haar algemene karakter is een implementatie van RIF moeilijk te vinden. Ze werken niet rechtstreeks met RDF-gebaseerde grafen[12, 11], of de implementatie is niet vrij beschikbaar[19]. Er werd daarom beslist om RIF enkel te gebruiken als tussentaal. De regels worden bijgevolg omgezet naar een andere regeltaal waarvan er wel implementaties bestaan. Het is deze regeltaal die uiteindelijk geïnterpreteerd zal worden.

Er werd gekozen om de RIF-regels om te zetten naar SPARQL-queries. SPARQL is als oorsprong geen regeltaal, maar de flexibele structuur laat toe dat ze hiervoor gebruikt kan worden. Op basis van dit gegeven werden reeds enkele vertalingen gemaakt die toelaten om SPARQL-queries te gebruiken als regels [16] [19]. SPARQL is tevens een wijd verspreide standaard die verschillende implementaties kent, waarvan er enkele opensource zijn (bv. Apache Jena[7]).

Voor de omzetting van RIF naar SPARQL werd een aparte tool ontwikkeld die besproken wordt in paragraaf 4.2.1.

Ontologie

Er zal in de implementatie gewerkt worden met ontologieën. Als eerste test werd er gewerkt met RDF om daarna over te stappen naar OWL. Zoals vermeld is OWL een uitbereiding op RDF, waardoor deze stap relatief eenvoudig was.

In de technologiestudie werden enkele voorbeelden aangehaald waarbij ontologieën gebruikt worden voor de allocatie van bronnen in een proces. De meeste implementaties hiervan maken gebruik van OWL voor het modelleren van de ontologie (bv. [4], [8] en [14]). OWL wordt momenteel aanbevolen voor het beschrijven van ontologieën door het W3C, wat dit uitgebreid gebruik mee verklaart.

[7] Apache Jena, `https://jena.apache.org/` (laatst geraadpleegd op 05/06/15)

4.1.2 Simulator

De tweede component die wordt besproken is de simulator. Er werd gekozen om hiervoor de processimulator van de onderzoeksgroep IBCN (zie paragraaf 2.2.2) te gebruiken. De code hiervan wordt door de vakgroep zelf beheerd, wat de voornaamste reden is waarom deze simulator wordt gebruikt. Het framework werkt bovendien zeer efficiënt, daar het in staat is om binnen enkele seconden een simulatie uit te voeren van een bepaald proces.

Programmacode

Het simulatie-framework is geschreven in Java. De belangrijkste Java-packages en hun functies worden hieronder toegelicht. Figuur 4.3 verduidelijkt de werking van de verschillende packages.

model-package De `model`-package vormt de kern van de simulator. Het wordt gebruikt om processen te beschrijven in code. De meeste klassen representeren een bepaald type element uit BPMN 2.0 (bv. een *start-event*, een bepaald type *gateway*, etc). Bijkomende abstracties worden gebruikt om op een generieke manier te kunnen werken met de procesbeschrijving in de andere delen van het framework.
Een tweede verantwoordelijkheid van de package is het beschrijven van bronnen in de simulator. Aangezien deze studie speciaal hierover gaat, worden de belangrijkste klassen hieronder toegelicht.

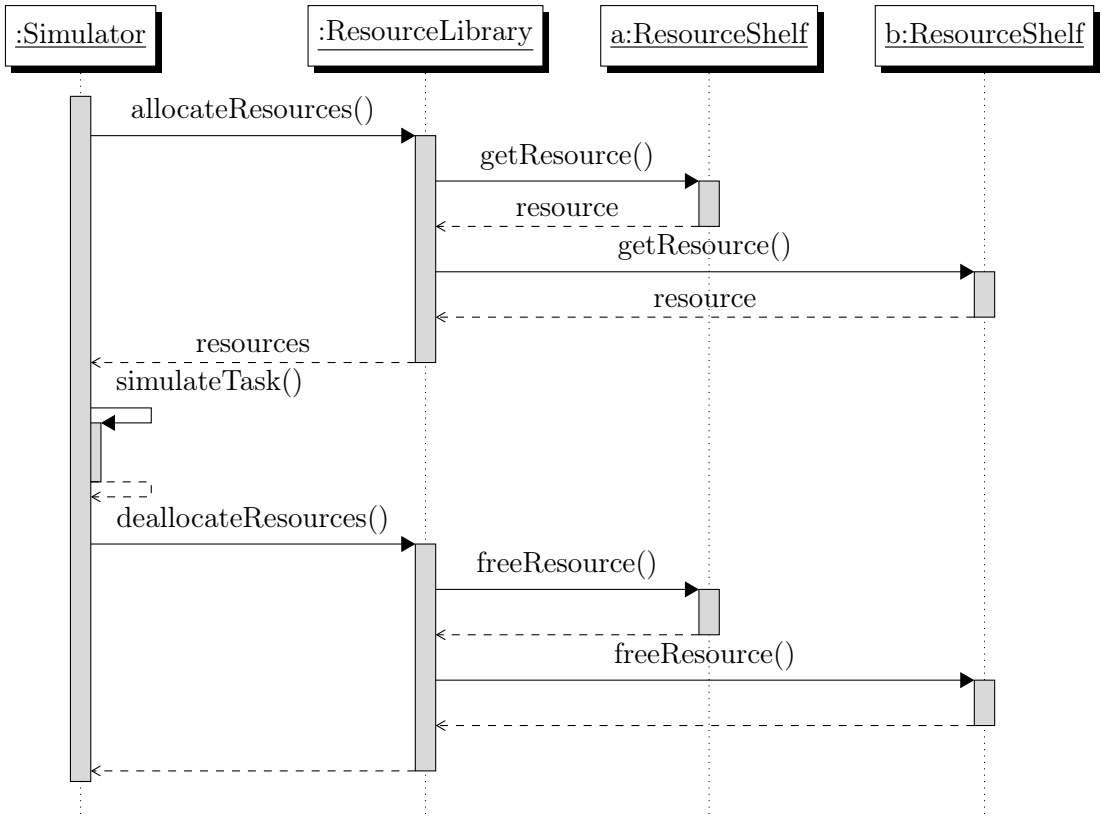
`ResourceLibrary` Deze klasse is verantwoordelijk voor het beheer van alle bronnen die gebruikt worden in het proces. De structuur voorziet methoden voor het opvragen en vrijgeven van bronnen. De bibliotheek delegeert haar aanvragen naar meerdere schabben die elk instaan voor het beheren van een bepaald type bron.

`ResourceShelf` Deze klasse is verantwoordelijk voor het beheer van één type bron. De klasse heeft methoden voor het opvragen en vrijgeven van dit type bron. Deze methoden worden aangeroepen door de `ResourceLibrary`.

`Resource` Deze klasse stelt een instantie voor van een bron. De klasse is gedefinieerd als een JavaBean[8]. Elke resource heeft een unieke identificatie, een bepaalde kost en een periode waarin ze gebruikt kan worden (bv. voor personeel met werkuren).

Figuur 4.2 illustreert de wisselwerking tussen deze verschillende componenten. Er wordt getoond hoe een taak wordt gesimuleerd, waarbij er twee verschillende soorten bronnen nodig zijn (bv. één personeelslid en één toestel). Beide bronnen worden dan aangevraagd via verschillende `ResourceShelves`.

[8] JavaBean, een Java-klasse met enkel *properties*, *getter*- en *setter*-methoden.



Figuur 4.2: Het beheer van bronnen in de processimulator van IBCN.

converter-package De `converter`-package wordt gebruikt voor het inlezen van externe bestanden. De drie voornaamste bestandstypes zijn:

`*.bpmn20.xml` Dit zijn procesmodellen beschreven in de XML-syntax van BPMN 2.0. Ze worden vertaald naar klassen uit de eerder besproken `model`-package.

`*.resources.xml` Doordat BPMN 2.0 geen methoden voorziet voor het beschrijven van bronnen in een bedrijfsproces, wordt hiervoor een apart bestand gebruikt. Per type bron wordt hier informatie opgegeven zoals:

- het aantal bronnen per type
- de variabele en vaste kostprijs voor het gebruik van een bron
- de periodes waarin bepaalde bronnen beschikbaar zijn (bv. voor personeel met bepaalde shiften)

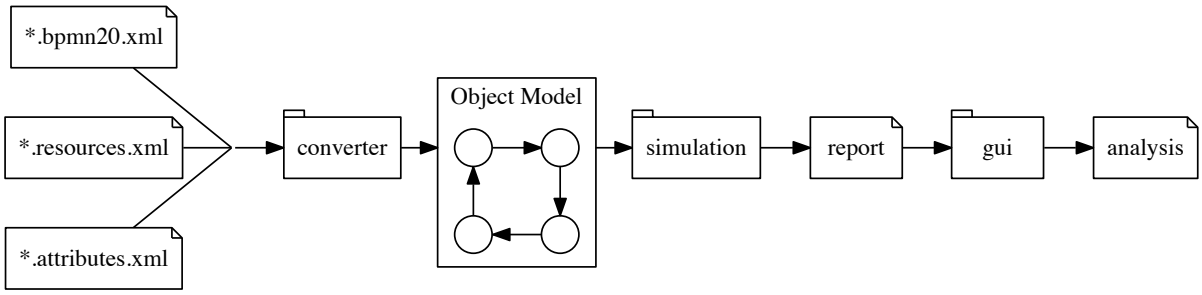
De informatie uit dit bestand wordt gebruikt voor het opstellen van de `ResourceLibrary` met de verschillende `ResourceShelves`.

`*.attributes.xml` Bestanden van dit type geven extra informatie over de verschillende taken uit het proces. Per element kan opgegeven worden welke bronnen nodig zijn.

Ook de kostprijs en duur van het element wordt hier opgegeven.

simulation-package De `simulation`-package bevat de eigenlijke functionaliteit voor het simuleren van een procesmodel (volgens DES).
Deze package bevat ook functionaliteit voor het rapporteren van informatie van een volledige simulatie. Via deze weg worden onder andere verschillende performance indicators van het proces berekend en opgeslagen.

gui-package De `gui`-package wordt gebruikt om het rapport van een volledige simulatie te bekijken. Op deze ruwe data kunnen analyses worden uitgevoerd zoals het bepalen van de werklast van een bepaald type bron.
De werking van de verschillende packages uit de simulator wordt vereenvoudigd weergegeven in Figuur 4.3.



Figuur 4.3: Werking van de componenten van de processimulator van IBCN.

4.1.3 Brugcomponent

De brugcomponent wordt hier als laatste van de drie hoofdcomponenten besproken omdat ze als enigte afhankelijk is van de twee andere. Zoals besproken in paragraaf 3.3.4, is ze verantwoordelijk voor de communicatie tussen het kennisgebaseerd systeem en de simulator.

De brugcomponent past het beste op de plaats van de `ResourceShelf`. Het aanpassen van deze klasse is niet vanzelfsprekend, omdat het framework compatibel moet blijven met reeds bestaande simulatiemodellen. Er werd daarom gekozen om het beheer van bronnen te abstraheren. De `IResourceShelf`-interface werd hiervoor geïntroduceerd en schrijft voor wat de verantwoordelijken zijn van de *shelves* die instaan voor het beheer van bronnen. Hiervan werden vervolgens twee implementatie afgeleid:

`DefaultResourceShelf` Deze shelf vervangt de implementatie die bronnen op de originele manier beheert. De effectieve bronnen worden toegekend in volgorde van hun unieke identificatie.

`OntologyResourceShelf` Deze shelf stelt de brugcomponent voor in dit architecturaal ontwerp. De allocatie en het vrijgeven van bronnen via het kennisgebaseerd systeem gebeurt op deze plaats. De effectieve werking wordt beschreven in paragrafen 4.2.2 en 4.2.3.

Apache Jena

Opdat de brugcomponent bronnen kan beheren, dient ze in staat te zijn regels (in dit geval SPARQL-queries) uit te voeren op een kennisgebaseerd systeem. Hiervoor wordt er beroep gedaan op het Apache Jena-framework (kortweg Jena). Dit opensource softwarepakket is geschreven in Java en biedt ondersteuning voor het bouwen van applicaties die gebruik maken van semantische webtechnologieën. De functionaliteit van het framework is opgedeeld in drie grote delen die hieronder worden besproken.

RDF Allereerst voorziet Jena functionaliteit voor het beheren van data in RDF. Er bestaat hiervoor een object model dat de verschillende componenten voorstelt die werden gedefinieerd voor RDF. Met behulp van voorziene parsers (bv. voor RDF/XML- en Tutrle-syntax) kunnen RDF-documenten worden ingelezen naar dit object model.

Jena laat toe dat deze RDF-grafen doorzocht worden met behulp van SPARQL-queries. Hiervoor werd de *query engine* ARQ ontwikkeld. De engine biedt volledige ondersteuning voor de SPARQL 1.1 Query Language en SPARQL 1.1 Update (paragraaf 2.1.3). Deze queries kunnen uitgevoerd worden op eender welk RDF-model. Naast de besproken W3C-standaarden is de component uitgebreid met extra ARQ-specifieke functionaliteit zoals de ondersteuning voor *free text search* en ARQ-specifieke functies (bv. $afn : min(num1, num2)$ en $afn : max(num1, num2)$ die respectievelijk het minimum en maximum teruggeven van twee expressions die een getalwaarde opleveren[9]). Het is ook mogelijk om zelf functies toe te voegen.

OWL Modellen die meer expressiviteit toevoegen aan hun data maken vaak gebruik van OWL. Speciaal hiervoor voorziet Jena specifieke functies zoals:

- het opvragen van specifieke attributen (bv. *functional properties* die uniek zijn per subject)
- het werken met OWL-klassen en hun eigenschappen (bv. *equivalent classes* die dezelfde individuen bevatten, maar uitgaan van een ander concept)

[9]Extensions in ARQ, `https://jena.apache.org/documentation/query/extension.html` (laatst geraadpleegd op 05/06/15)

- het werken met OWL-metadata

Hoewel deze functionaliteit specifiek bestemd is voor OWL-ontologieën, wordt er gebruik gemaakt van de nodige abstracties. Daardoor is deze *Application programming interface* (API) volledig compatibel met de API gebruikt voor RDF-modellen.

Een belangrijke toevoeging hieraan is het gebruik van een reasoner. Jena voorziet ondersteuning voor reasoners op een generieke manier. Hierdoor kunnen nieuwe reasoner-implementaties inhaken in het framework door de juiste *adapters* te voorzien.

Triple store Het laatste element van Jena bestaat uit een hoge performantie triple store voor het beheer van zeer grote RDF-grafen. Dergelijke triple stores kunnen vervolgens aangeboden worden als service via *Fuseki*. Deze server-applicatie laat toe SPARQL-queries uit te voeren op de triple store via HTTP-aanvragen.

Keuze Er werd voornamelijk gekozen voor Apache Jena omdat het een opensource framework is dat een uitgebreide ondersteuning biedt voor courante standaarden die gebruikt worden bij de ontwikkeling van semantische applicaties. Het wordt ontwikkeld door een levendige community zonder invloed van een derde partij.
Bovendien is het een vrij flexibel framework dat aanmoedigt om zelf functionaliteit toe te voegen (bv. het voorzien van een reasoner, het definiëren van eigen SPARQL-expressions).

4.2 Uitdieping

In dit deel wordt er dieper ingegaan op bepaalde componenten van de implementatie. Paragraaf 4.2.1 bespreekt de vertaling van RIF naar SPARQL. Dit wordt vervolgd met een gedetailleerde beschrijving van de allocatie van bronnen.

4.2.1 Omzetting van RIF naar SPARQL

Voor de omzetting van RIF naar SPARQL werd een op zich staande component ontwikkeld. Deze omzetting is gebaseerd op twee artikels. Allereerst worden beide artikels besproken, waarna de uiteindelijke omzetting wordt toegelicht.

RIF4J — A Reasoning Engine for RIF-BLD [12]

In deze studie wordt onderzocht hoe RIF-BLD-regels gebruikt kunnen worden voor reasoning-taken in een Datalog engine. Er wordt hierbij eerst een object model in Java voorgesteld voor

de programmatische verwerking van RIF-BLD-regels. Hierop volgt een vertaling van RIF-BLD-formules naar Datalog. Dit laat toe dat RIF rechtstreeks gebruikt kan worden in een Datalog engine zoals IRIS[10]. Als laatste wordt onderzocht hoe de IRIS-reasoner verbeterd kan worden op vlak van schaalbaarheid. Er wordt hiertoe een uitbreiding op IRIS voorsteld, IRIS-RDB. Deze uitbereiding maakt het mogelijk om Datalog-programma's af te handelen met behulp van relationele databanken.

In deze studie werd uitsluitend gebruik gemaakt van het RIF-BLD object model, wat verder wordt toegelicht. Het model is een representatie van de presentation syntax van RIF. De nodige abstracties werden ingebouwd om op een generieke manier te kunnen werken met de constructies uit RIF-BLD.

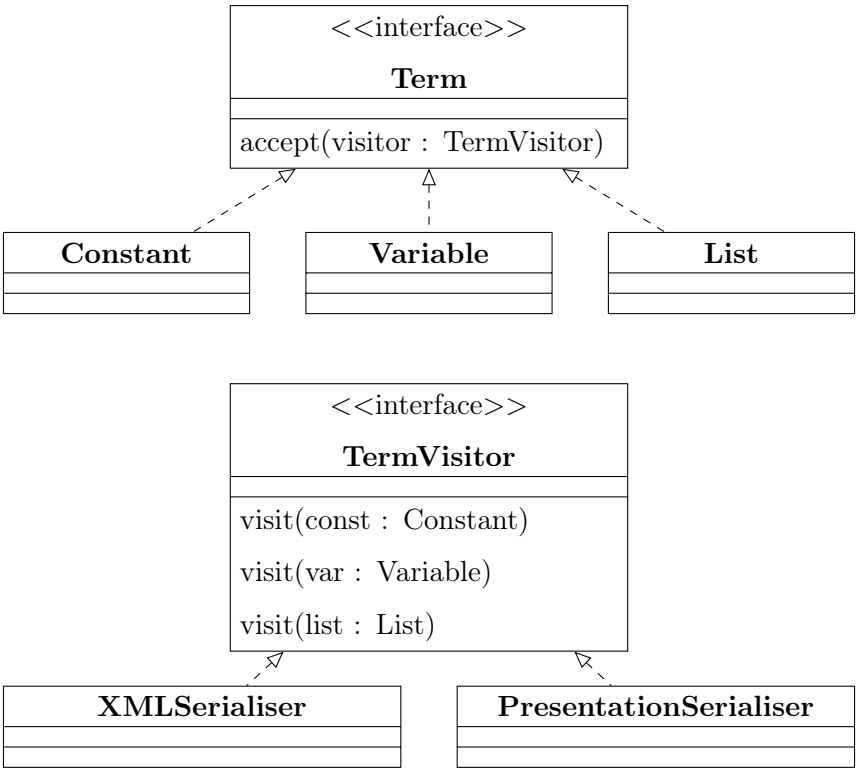
RIF4J ondersteunt het gebruik van het *Visitor Pattern*[9]. Hierdoor kunnen andere applicaties eenvoudig werken bovenop de functionaliteit van RIF4J. Het *design pattern* schrijft voor dat elke klasse in de structuur (bv. de klasse `Term` die een RIF-term voorstelt) een `accept`-methode moet ondersteunen die een `Visitor`-klasse accepteert als argument. De `Visitor` (bv. de klasse `TermVisitor`) is een interface die een `visit`-methode heeft voor elke klasse uit de structuur die het dient te ondersteunen. Op deze manier zou bijvoorbeeld een `TermVisitor` ondersteuning bieden voor de methoden `visit(Constant)`, `visit(Variable)` en `visit(List)`.
Om de klassenstructuur te doorlopen, roept men de `accept`-methode aan met als argument een `Visitor`. De klasse uit de structuur zal op zijn beurt de `visit`-methode aanroepen van de `Visitor`, met zichzelf als argument. Op deze manier kunnen meerdere visitors voorzien worden, die elk de structuur afhandelen op hun manier. Het RIF4J-framework biedt reeds functionaliteit aan via de implementatie van visitor-klassen. Zo is het mogelijk om een RIF-regel uit te schrijven in de XML-syntax en presentation syntax (Figuur 4.4). Verder worden visitors voorzien voor de normalisatie en transformatie van RIF-BLD termen.

Het opstellen van een RIF-BLD-regel kan zowel in code als door gebruik te maken van een voorziene XML-parser.

Bridging the Gap between RIF and SPARQL: Implementation of a RIF Dialect with a SPARQL Rule Engine [19]

In het tweede artikel wordt een vertaling tussen RIF-BLD en SPARQL besproken. Allereerst worden beiden talen voorgesteld. Voor SPARQL wordt tevens besproken hoe een CONSTRUCT query (zie paragraaf 2.1.3) gebruikt kan worden als *forward chaining inference rule*. Men kan namelijk nieuwe kennis construeren (met het CONSTRUCT-sjabloon) op basis van reeds bestaande

[10]IRIS Reasoner, `http://www.iris-reasoner.org` (laatst geraadpleegd op 05/06/2015)



Figuur 4.4: Een illustratie van het visitor pattern, gebruikt in RIF4J.

kennis in het model (de bevraagde RDF-graaf). Op basis van dit gegeven wordt eerst een vertaling voorgesteld van SPARQL naar RIF-BLD. Vervolgens wordt deze vertaling omgekeerd opdat RIF-regels omgezet kunnen worden naar SPARQL-queries.

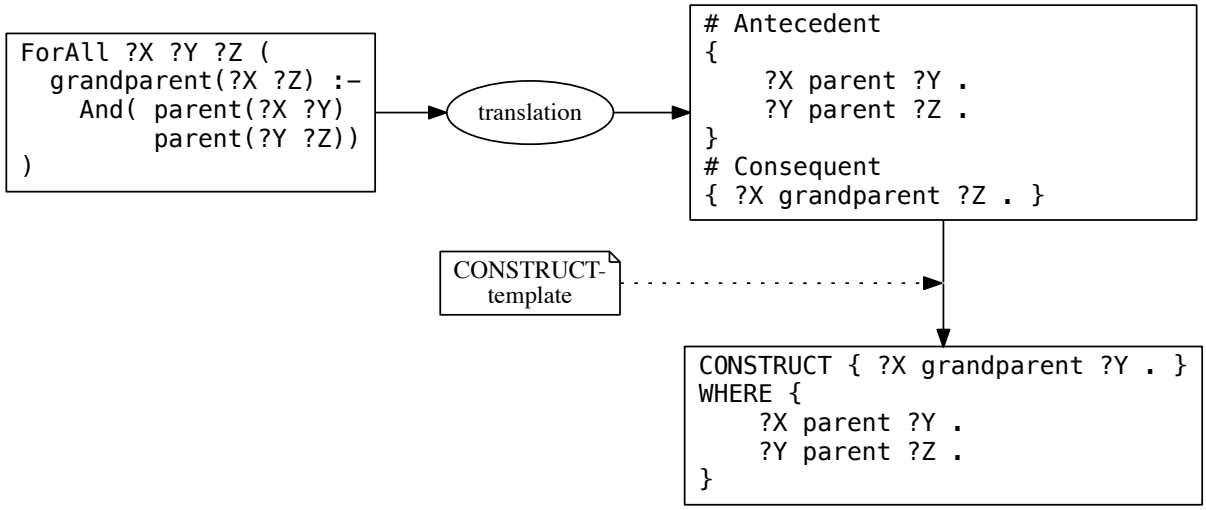
RIF en SPARQL zijn zeer verschillende talen, zowel in de manier van schrijven als het doel waarvoor ze ontwikkeld werden. Hierdoor is een volledige vertaling tussen beide niet mogelijk. Er wordt daarom een nieuw RIF-dialect voorgesteld, RIF-SPARQL, dat alle RIF-BLD constructies omvat die vertaalbaar zijn naar SPARQL. Dit dialect omvat alle RIF-BLD-constructies met de uitzondering van *universal facts* en open lijsten. Daarbij kunnen bij de *Equal*-, *Member*-, *Subclass*- en *Frame*-formules enkel RIF-termen gebruikt worden die een constante, variabele of gesloten lijst voorstellen.

De vertaalcomponent

In deze paragraaf wordt de component besproken die werd ontwikkeld om RIF-regels om te zetten naar SPARQL-queries. Deze omzetting gebeurt in twee stappen. Allereerst worden RIF-regels in XML-formaat ingelezen met behulp van het RIF4J-framework. De parser construeert hierbij het reeds besproken RIF-BLD object model. Vervolgens wordt de volledige structuur doorlopen met behulp van visitors. Op deze manier kan elk onderdeel stuk voor stuk omgezet worden

gelijkaardig aan de vertaling beschreven in het tweede artikel.

De vertaling gebeurt op een licht gewijzigde manier. In [19] wordt elke RIF-regel omgezet naar een SPARQL-query van het type CONSTRUCT. Het antecedent en consequent van een regel wordt dan respectievelijk omgezet naar het query pattern en het sjabloon van de CONSTRUCT query. Sinds de komst van SPARQL 1.1 Update is CONSTRUCT echter niet meer het enige type query dat gebruikt kan worden om regels voor te stellen in SPARQL. Om deze verscheidenheid op te vangen en om extra flexibiliteit te voorzien, werd gekozen om de RIF-regels om te zetten naar een tussenliggend formaat. Dit formaat bevat een vertaalde versie van het antecedent en consequent van de RIF-regel. Beide objecten zijn dus geen volwaardige query, maar kunnen gebruikt worden als basis voor een SPARQL-query. De component maakt daarvoor gebruik van sjablonen die elk in staat zijn een ander soort query te construeren. Eenzelfde vertaling kan bijgevolg gebruikt worden als basis voor meerdere type queries. Deze functionaliteit wordt gebruikt bij de allocatie van een bron (paragraaf 4.2.2). Het liet ook toe op een flexibele manier meerdere procedures voor de allocatie te testen (elk gebruik makende van aangepaste vertalingen, zie paragraaf 4.3). Zowel voor algemeen belang als voor het implementatiegedeelte van deze thesis werden enkele voorgedefinieerde sjablonen opgesteld en toegevoegd aan het framework. Een voorbeeld van de vertaling wordt geïllustreerd in Figuur 4.5.



Figuur 4.5: Illustratie van de vertaling van een RIF-regel naar een SPARQL-query.

De vertaling In deze paragraaf wordt de geïmplementeerde vertaling van RIF-BLD naar SPARQL besproken.

Zowel *rule-implications* (regels zonder variabelen), *universal-rules* (regels mét variabelen) als *atomic facts* (regels zonder antecedent) kunnen vertaald worden naar SPARQL-queries:

- Het consequent van een RIF-regel wordt vertaald naar een BGP zonder SPARQL-filters
- Het antecedent van een RIF-regel wordt vertaald naar een query pattern dat gebruikt wordt in de WHERE-clausule van een query. Hiervoor gelden de volgende regels:
  - Bij een conjunctie worden alle factoren vertaald en samengevoegd tot een BGP.
  - Bij een disjunctie worden alle factoren vertaald en samengevoegd met behulp van UNION-constructies.

De quantifiers uit RIF hebben geen tegenhanger in SPARQL. Bij de vertaling worden er daarom geen bijkomende constructies voorzien, al dient dit met de nodige voorzichtigheid te gebeuren. Theoretisch gezien is het niet mogelijk om de universal quantifier `ForAll` te vertalen naar een SPARQL-constructie. In de open wereld van RDF kan er namelijk nooit met zekerheid gezegd worden dat alle entiteiten werden beïnvloed door de query. Data kan in RDF namelijk afkomstig zijn van meerdere grafen die niet allemaal gekend zijn. Bij een `ForAll` quantifier moeten al deze grafen in rekening gebracht worden. De ontologieën uit deze thesis worden echter gebruikt voor het beschrijven van een specifieke domeinkennis. Er wordt hierbij verondersteld dat dit de enige kennis is die over dit domein bestaat. Dit impliceert een gesloten wereld. Daarom wordt de vertaling van universal quantifiers hier wél toegelaten.

Op een dieper niveau wordt een atomair gegeven in RIF vertaald naar een SPARQL-triple. Predicaten met meer dan twee termen als argument worden niet ondersteund. De RIF-termen zelf worden vertaald op basis van de onderstaande regels:

- Constanten worden vertaald naar letterlijke waarden in RDF. Het datatype wordt hierbij omgezet naar een overeenkomstig XML Schema datatype. Indien er geen type werd opgegeven of indien het type niet wordt ondersteund, dan wordt er gebruik gemaakt van een *string-literal*.
- RIF-variabelen worden omgezet naar SPARQL-variabelen.
- Een *class membership term* (bv. `o # c`) wordt vertaald naar een triple met `rdf:type` als predicaat (bv. `o rdf:type c`).
- Een *subclass term* (bv. c_1 `##` c_2) wordt vertaald naar een triple met `rdfs:subClassOf` als predicaat (bv. c_1 `rdfs:subClassOf` c_2).
- Een *frame* (bv. `s[`p_1 `->` o_1 `...` p_n `->` o_n`]`) wordt vertaald naar meerdere triples met hetzelfde subject (bv. `s` p_1 o_1`; ...; s` p_n o_n`;`).

- *Equality terms* die variabelen en / of constanten vergelijken worden vertaald naar gelijkheidstesten in SPARQL-filters. Gelijkheidstesten tussen andere soorten termen worden niet ondersteund.
- *Externally defined terms* worden vertaald naar overeenkomstige operatoren of predicaten in SPARQL-filters.

De voornaamste beperkingen van het geïmplementeerde RIF-SPARQL-dialect ten opzichte van RIF-BLD zijn:

- Atomaire gegevens met meer dan twee termen als argument worden niet ondersteund.
- *Positional terms*, *named arguments* en lijsten worden niet ondersteund.
- De implementatie voorziet een uitgebreide vertaling van RIF-externals naar SPARQL-functies en -predicaten. Deze vertaling kan steeds worden aangevuld.

De aangepaste grammatica van het RIF-SPARQL dialect in presentation syntax (ten opzichte van [19]) wordt dan gegeven door Codevoorbeeld 4.1.

```
RULE        ::= CLAUSE |
                'Forall' Var+ '(' CLAUSE ')' |
                'Exists' Var+ '(' CLAUSE ')'
CLAUSE      ::= ATOMIC ':-' FORMULA |
                'And' '(' ATOMIC* ')' ':-' FORMULA
FORMULA     ::= ATOMIC |
                ('And' | 'Or') '(' FORMULA* ')'
ATOMIC      ::= Atom | Frame | Member | Subclass | Equal
Atom        ::= UNITERM
UNITERM     ::= Const '(' TERM TERM ')'
Equal       ::= TERM1 '=' TERM1
Member      ::= TERM1 '#' TERM1
Subclass    ::= TERM1 '##' TERM1
Frame       ::= TERM1 '[' (TERM1 '->' TERM1)* ']'
TERM        ::= TERM1 | TERM2
TERM1       ::= Const | Var
TERM2       ::= Expr | 'External' '(' Expr ')'
Expr        ::= UNITERM
```

Codevoorbeeld 4.1: De grammatica van RIF-SPARQL die ondersteund wordt in de vertaalcomponent van deze thesis.

Apache Jena

Het Apache Jena-framework werd reeds besproken in paragraaf 4.1.3. De brugcomponent maakt gebruik van dit framework voor het beheren van de ontologie. De aanpassingen die uitgevoerd worden op dit model gebeuren via de ARQ query engine. Omwille van deze keuze, gebeurt de omzetting van het RIF4J object model naar het Apache Jena object model voor het beschrijven van SPARQL queries in code.

4.2.2 Allocatie

In deze paragraaf wordt besproken op welke manier bronnen geselecteerd worden uit de ontologie. Allereerst worden enkele bijkomende eisen besproken waaraan de implementatie moet voldoen. Vervolgens wordt de allocatie zelf toegelicht.

Eisen

Het werk in deze thesis voegt functionaliteit toe aan de processimulator van IBCN. Hierdoor zijn er bijkomende eisen waaraan de allocatie (en deallocatie) moet voldoen. Deze worden hieronder besproken.

Identificatie van een bron Indien de simulator een bron vraagt aan de brugcomponent, wordt verwacht dat deze bron een bepaalde identificatie bevat. Dit is bijvoorbeeld vereist om analyses uit te kunnen voeren op de rapporten van de simulator (bv. voor het berekenen van de werklast van het personeel). Verder is de identificatie belangrijk indien de bron wordt vrijgegeven. Enkel op basis van deze identificatie is het mogelijk dat de correcte bron wordt aangepast in de ontologie (bv. de aanpassing van een status naar *beschikbaar*).

Unieke toewijzing van een bron Indien de simulator een bron vraagt aan de brugcomponent, wordt verwacht dat er maximaal één bron wordt toegewezen aan deze taak. Een probleem kan ontstaan indien meerdere entiteiten uit het kennismodel voldoen aan het antecedent van een regel. In dat geval dient er een verantwoorde keuze worden gemaakt over welke bron effectief zal worden toegekend.

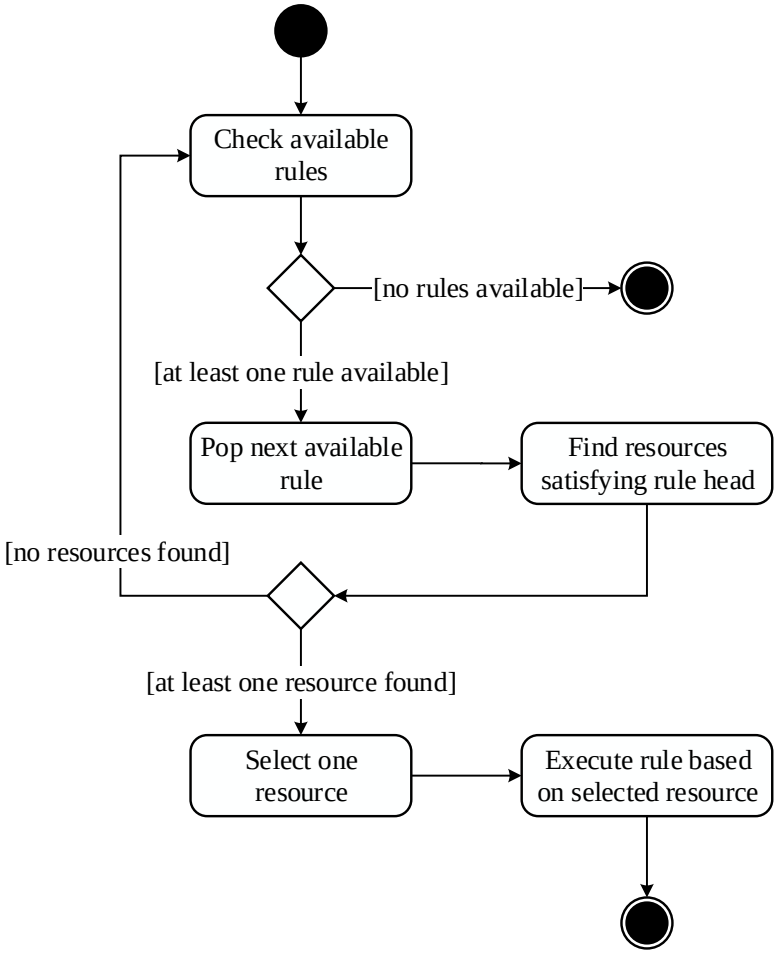
Implementatie allocatie (de `construct` methode)

In deze paragraaf wordt besproken hoe een bron effectief gealloceerd wordt. Per type bron (bv. apparatuur, personeel, etc) zijn hiervoor twee componenten vereist: een referentie naar de ontologie en een geordende set van regels (de allocatieregels). De allocatieregels van één type

bron zijn afkomstig van één RIF-document. De volgorde uit het document wordt behouden bij de vertaling naar SPARQL.

Om een bron te alloceren, wordt een procedure gebruikt die bestaat uit drie stappen (zie Figuur 4.6 voor een illustratie):

1. het vinden van beschikbare bronnen die voldoen aan het antecedent van één van de regels
2. het kiezen van één bron uit de verschillende beschikbare bronnen
3. het uitvoeren van de regel op de ontologie op basis van de gekozen bron



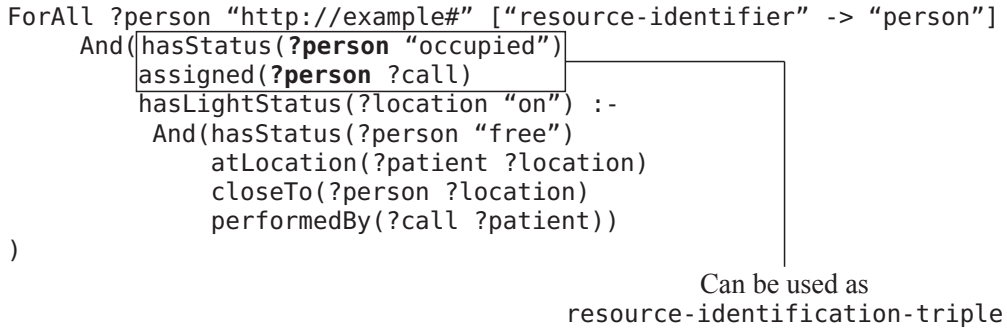
Figuur 4.6: Illustratie van het verloop van een allocatie.

Het vinden van beschikbare bronnen De eerste stap in een allocatie is het vinden van bronnen die voldoen aan het antecedent van één van de regels. Om te beginnen wordt de eerste

regel uitgevoerd op de ontologie. Het resultaat van deze uitvoering wordt opgenomen in een tijdelijk model. Indien dit tijdelijk model geen resultaten bevat, wil dit zeggen dat er op dat moment geen bronnen voldoen aan het antecedent van de eerste regel. Deze eerste stap wordt dan herhaald voor de volgende allocatieregel. Indien het tijdelijk model wél resultaten bevat, dan kan men op basis daarvan de verschillende bronnen identificeren die beschikbaar zijn.

Om het tijdelijk model te kunnen construeren, wordt de regel vertaald naar een query van het type CONSTRUCT. Deze query wordt vervolgens uitgevoerd op de ontologie, waarbij de resulterende graaf wordt toegevoegd aan het tijdelijk model.

Om bronnen in het tijdelijk model te kunnen identificeren wordt gebruik gemaakt van een speciale variabele: de `resource-identifier`. Dit is een variabele die gebonden dient te worden aan de identificatie van de bronnen. In de RIF-regel wordt (via metadata) aangegeven welke variabele de `resource-identifier` voorsteld. Op basis van deze variabele wordt in het consequent van de regel gezocht naar een `resource-identification-triple`. Dit is een triple waarin de `resource-identifier` gebruikt wordt. Op basis van dit triple kan bijgevolg achterhaald worden wat de identificatie van een bron is (zie Figuur 4.7 voor een voorbeeld).



Figuur 4.7: Voorbeeld van de extractie van het `resource-identification-triple` uit een RIF-regel op basis van voorziene metadata.

Bij het uitvoeren van de regel zal, per beschikbare bron, het `resource-identification-triple` worden toegevoegd aan het tijdelijk model. Door het tijdelijk model te doorzoeken naar triples die gelijkaardig zijn aan het `resource-identification-triple`, kan men achterhalen wat de waarde was van de `resource-identifier` variabele. Deze verschillende waarden worden verzameld en gebruikt in de volgende stap voor het kiezen van een bron.

Indien het tijdelijk model nog steeds leeg is nadat alle allocatieregels werden uitgevoerd, dan wil dit zeggen dat er op dat moment geen bron beschikbaar is. In dat geval stopt de procedure en wordt er geen bron teruggegeven aan de simulator.

Kiezen van een bron uit de beschikbare Afhankelijk van het aantal beschikbare bronnen (= het aantal gevonden waarden van de `resource-identifier`) moet een bijkomende selectie gebeuren. Er mag immers maar één bron toegekend worden per aanvraag van de simulator. Er werd gekozen om deze bron willekeurig te kiezen uit de verzameling van beschikbare bronnen. Dit voorkomt dat deze bijkomende selectie een invloed zou hebben op analyses zoals de werklast.

Uitvoeren van de regel Om de gekozen bron effectief te alloceren wordt de regel nogmaals uitgevoerd. Aangezien het resultaat van de regel dit keer wél rechtstreeks mag opgenomen worden, wordt er gebruik gemaakt van een SPARQL-query van het type INSERT. In deze query wordt de `resource-identifier` variabele gesubstitueerd door de identificatie van de gekozen bron. Hierdoor wordt slechts één bron beïnvloed bij de uitvoering van de query op de ontologie.

De pseudocode van een allocatie wordt getoond in Algoritme 1.

4.2.3 Deallocatie

In deze paragraaf wordt de deallocatie van een bron besproken en de pseudocode hiervan wordt gegeven in Algoritme 2. Net als bij de allocatie kan de deallocatie gebeuren op basis van meerdere regels. De regels worden behandeld volgens hun volgorde in het originele RIF-document. Alvorens de regels worden uitgevoerd dienen ze, net als bij een allocatie, worden aangepast. Een regel mag namelijk enkel worden uitgevoerd voor de specifieke bron die wordt vrijgegeven. Hiertoe wordt een substitutie uitgevoerd op de variabele die de identificatie van de bron voorstelt (de `resource-identifier` variabele).

Na de substitutie wordt de regel uitgevoerd op de ontologie. Indien er geen wijziging op de ontologie werd geregistreerd, wordt de volgende regel uitgevoerd op het model. Er wordt verwacht dat minstens één regel tot een wijziging van het model zal leiden.

4.3 Toevoegingen

De voorgaande paragrafen beschrijven de normale werking van de allocatie en deallocatie van een bron op basis van een ontologie. In dit deel worden enkele aanvullingen besproken die bouwen op deze procedures.

Algoritme 1: Allocatie van een bron.

Data: $ontoModel \leftarrow$ the ontology model

Data: $allocRules \leftarrow$ rules used for allocation

Result: $resourceId$ extracted from $ontoModel$ representing the identification of a resource which has been allocated

begin

- $tempModel \leftarrow$ an empty ontology model;
- **while** *tempModel is empty* **and** *allocRules has more rules* **do**
  - $rule \leftarrow$ get next rule in $allocRules$;
  - $constrQuery \leftarrow$ create CONSTRUCT-query based on $rule$;
  - $tempModel \leftarrow$ execute $constrQuery$ on $ontoModel$;
- **end**
- **if** *tempModel is not empty* **then**
  - $ids \leftarrow$ get resource-identifiers from $tempModel$;
  - $resourceId \leftarrow$ select random entity from ids;
  - $insertQuery \leftarrow$ create INSERT-query based on $rule$;
  - substitute in $insertQuery$ the variable representing the resource-identifier for $resourceId$;
  - execute $insertQuery$ on $ontoModel$;
- **else**
  - $resourceId \leftarrow$ `null`;
- **end**
- **return** $resourceId$

end

4.3.1 Functional properties

Een eerste aanvulling pakt het probleem aan van *functional properties* in OWL. Hiervan wordt er eerst een definitie gegeven:

> Een *object property* OP is *functional* indien er voor elk individu x uit het model maximaal één individu y kan gevonden worden, zodat x verbonden is met y via OP [27].

Dit wil zeggen dat een functional property nooit meer dan één keer gedefinieerd mag zijn voor hetzelfde subject. Is dit wel het geval, dan impliceert dit equivalente uitdrukkingen of een inconsistentie in de ontologie. Deze situatie kan voorkomen indien een regel een functional property verzekerd in het model. Het reeds bestaande triple met dit predicaat moet dan eerst worden verwijderd (enkel voor dat subject).

Algoritme 2: Deallocatie van een bron.

Data: $resourceId \leftarrow$ identification of the resource that has to be deallocated
Data: $ontoModel \leftarrow$ the ontology model
Data: $deallocRules \leftarrow$ rules used for deallocation
begin
  while *ontoModel did not change* **and** *deallocRules has more rules* **do**
    $rule \leftarrow$ get next rule in $deallocRules$;
    $insertQuery \leftarrow$ create INSERT-query based on $rule$;
    substitute in $insertQuery$ the variable representing the resource-identifier for $resourceId$;
    execute $insertQuery$ on $ontoModel$;
  end
end

Aangezien dit niet automatisch gebeurt bij het uitvoeren van SPARQL-queries, moeten deze queries worden aangepast.

Er werden nieuwe sjablonen voorzien die een RIF-regel omzetten naar een SPARQL-query met een DELETE-clausule. Met deze clausule kunnen de bestaande triples met *functional properties* eerst worden verwijderd van het model. Welke predicaten juist functioneel zijn, kan worden afgeleid van de ontologie. Apache Jena voorziet hiervoor de nodige methoden.

Bij het opstellen van een SPARQL-query met een DELETE-clausule worden bijgevolg extra stappen ondernomen. Het vertaalde antecedent en consequent worden respectievelijk gebruikt in een tijdelijke WHERE- en INSERT-clausule. De DELETE-clausule is initieel leeg. Vervolgens wordt voor elk predicaat uit het consequent gecontrolleerd of dit een functional property voorstelt in de ontologie. Is dit het geval, dan wordt een nieuw triple geconstrueerd. Dit triple is een kopie van het gecontroleerde triple, waarbij het object werd vervangen door een nieuwe variabele. Dit triple wordt vervolgens opgenomen in zowel de WHERE- als de DELETE-clausule van de query. Pas indien alle triples werden gecontrolleerd, kan de query gebruikt worden op de ontologie.

Pseudo-code van de vertaling van een RIF-regel met functional properties wordt gegeven door Algoritme 3. Een voorbeeld van een dergelijke vertaling wordt gegeven door Figuur 4.8.

```
ForAll ?person (
    ?person hasStatus "occupied" :-
        ?person hasStatus "free"
)
```

(a) De RIF-regel.

```
DELETE {
    ?person hasStatus ?status
}
INSERT {
    ?person hasStatus "occupied"
}
WHERE {
    ?person hasStatus "free" .
    ?person hasStatus ?status .
}
```

(b) De vertaalde regel in SPARQL.

Figuur 4.8: De vertaling van een RIF-regel naar SPARQL, waarbij rekening wordt gehouden met functional properties in OWL.

4.3.2 Performantie

De allocatie van een bron besproken in paragraaf 4.2.2 werkt in drie stappen: het uitvoeren van regels op een tijdelijk model, het selecteren van een bron en het uitvoeren van een regel op de ontologie. Aangezien performantie voor een discrete event simulator van belang is, werd er gezocht naar manieren om de impact van deze stappen te verkleinen. In de paragrafen die hierop volgen worden enkele alternatieven voorgesteld. Om te weten te komen welke de snelste is, werd een vergelijkende studie gemaakt. Deze testen, alsook de resultaten, worden besproken in hoofdstuk 5.

De `select` methode

De eerste methode vermijdt de constructie van een tijdelijk model bij de uitvoering van de CONSTRUCT-queries. Dit tijdelijk model wordt enkel gebruikt om te achterhalen welke bronnen voldoen aan het antecedent van een regel. Deze functionaliteit kan op een eenvoudigere manier bekomen worden door SPARQL SELECT-queries te gebruiken.

Het zoeken van beschikbare bronnen gebeurt in deze methode daarom op basis van SELECT-queries, waarvan het query pattern gelijk is aan het antecedent van de regel. Wetende welke variabele de `resource-identifier` voorsteld, kunnen de beschikbare bronnen uit het resultaat worden afgeleid. Hierna volgen de twee laatste stappen zoals ze eerder werden besproken in paragraaf 4.2.2. Figuur 4.9 toont de queries die gebruikt worden voor de allocatie via deze methode.

```
SELECT *
WHERE {
    ?person hasStatus "free" .
}
```

(a) Query gebruikt voor het identificeren van beschikbare bronnen.

```
DELETE {
    ?person hasStatus ?status .
}
INSERT {
    ?person hasStatus "occupied" .
}
WHERE {
    ?person hasStatus "free" .
    ?person hasStatus ?status .
}
```

(b) Query gebruikt voor de effectieve allocatie (`?person` wordt hier vervangen door de identificatie van een bron).

Figuur 4.9: De gebruikte queries bij de allocatie van een bron volgens de `select` methode.

De `random-select` methode

Deze methode voor het alloceren van een bron voegt de twee eerste stappen uit paragraaf 4.2.2 samen. Het kiezen van een bron uit de verschillende beschikbare wordt bijgevolg al uitgevoerd in de eerste query. Aangezien dit op een willekeurige manier dient te gebeuren, werd er gezocht naar methoden om in SPARQL een willekeurige node te selecteren uit de oplossingenverzameling van een query. Dit onderzoek wordt apart besproken in appendix A. De methode die het meest geschikt bleek, wordt hier gebruikt.

In deze methode wordt de oplossingenverzameling van een regel gesorteerd op een willekeurige waarde, waarna hieruit de eerste oplossing wordt gekozen. Op basis van dit resultaat kan de identificatie van de bron achterhaald worden. Deze waarde wordt op zijn beurt gesubstitueerd in een update-query die de effectieve allocatie uitvoert. De gebruikte queries worden getoond in Figuur 4.10.

De `random-update` methode

Deze laatste methode tracht alle drie de stappen uit te voeren met een enkele query. Om dit te bereiken wordt er beroep gedaan op *subqueries*[11] uit SPARQL. Deze functionaliteit laat toe dat een query pattern kan inwerken op het resultaat van een andere query, in plaats van op de bevraagde graaf.

[11] 'Subqueries are a way to embed SPARQL queries within other queries, normally to achieve results which cannot otherwise be achieved, such as limiting the number of results from some sub-expression within the query.' [33]

```
SELECT *
WHERE {
    ?person hasStatus "free" .
    BIND(RAND() AS ?randomVar) .
}
ORDER BY ?randomVar
LIMIT 1
```

(a) De query gebruikt voor het bepalen van de beschikbare bron.

```
DELETE {
    ?person hasStatus ?status .
}
INSERT {
    ?person hasStatus "occupied" .
}
WHERE {
    ?person hasStatus "free" .
    ?person hasStatus ?status .
}
```

(b) De query gebruikt voor de effectieve allocatie (`?person` wordt hier vervangen door de identificatie van een bron).

Figuur 4.10: De gebruikte queries bij de allocatie van een bron volgens de `random-select` methode.

De query uit de `random-select` methode wordt hier hergebruikt als subquery. Het resultaat van deze subquery wordt rechtstreeks gebruikt in de update-query voor de effectieve allocatie in het kennismodel. Door te luisteren naar de triples die worden toegevoegd aan het model, kan achterhaald worden welke bron effectief werd gealloceerd. Indien er geen triples werden toegevoerd, voldoen er op dat moment geen bronnen aan het antecedent van de regel. De methode wordt dan herhaald voor de volgende allocatieregel. Een voorbeeld van de query gebruikt in deze methode wordt gegeven in Codevoorbeeld 4.2.

4.3.3 Analyse tools

Een van de eisen van het te ontwerpen systeem was de aanwezigheid van tools om analyses uit te kunnen voeren op het gebruik van de regels. Er werd daarom een bijdrage geleverd aan het rapporteringsmechanisme in de simulator. Concreet worden de volgende gegeven bijgehouden tijdens een simulatie:

- Per allocatieregel:
  - het aantal keer de regel werd uitgevoerd
  - het gemiddeld aantal bronnen die beschikbaar waren (het aantal bronnen die op de moment van uitvoering voldeden aan het antecedent van de regel)
  - het minimum en het maximum aantal bronnen die beschikbaar waren

```
DELETE {
    ?person hasStatus ?status .
}
INSERT {
    ?person hasStatus "occupied" .
}
WHERE {
    {
        SELECT *
        WHERE {
            ?person hasStatus "free" .
            ?person hasStatus ?status .
            BIND(RAND() AS ?randVar) .
        }
        ORDER BY ?randVar
        LIMIT 1
    }
}
```

Codevoorbeeld 4.2: De query gebruikt om een willekeurige beschikbare bron te alloceren in een ontologie via de `random-update` methode.

- Per deallocatieregel:
  - het aantal keer de regel werd uitgevoerd
  - het aantal keer de regel leidde tot de deallocatie van een bron
  - het aantal keer de regel niet leidde tot de deallocatie van een bron

Deze gegevens zijn echter niet voor elke methode correct. Indien de allocatie gebeurt op basis van de `random-select` of `random-update` methode (zie *Performantie* onder paragraaf 4.3), dan gebeurt de selectie van één bron uit de beschikbare bronnen via SPARQL. Daardoor wordt er ofwel geen, ofwel één bron gevonden via deze methoden. Dit weerspiegelt zich in de rapporten waar er maximaal één bron zal worden getoond voor het aantal beschikbare bronnen.

Algoritme 3: Constructie van een SPARQL-query op basis van een RIF-regel, waarbij rekening wordt gehouden met functional properties uit OWL.

Data: $rule \leftarrow$ RIF-rule that will be executed
Data: $ontoModel \leftarrow$ the OWL-ontology on which the rule will be executed
Result: The translation of *rule* to a SPARQL-query with a DELETE-, INSERT- and WHERE-clause, taking into account functional properties of OWL

```
begin
    (antecedent, consequent) ← translation of rule to intermediate format;
    queryPattern ← antecedent;
    insertTriples ← consequent;
    deleteTriples ← empty collection of triples;
    foreach triple t in consequent do
        pred ← the predicate in t;
        if pred is a functional property in ontoModel then
            subj ← subject in t;
            var ← a new, unique variable;
            add triple < subj pred var > to deleteTriples;
            add triple < subj pred var > to queryPattern;
        end
    end
    query ← a SPARQL-query;
    set deleteTriples as the DELETE-clause of query;
    set insertTriples as the INSERT-clause of query;
    set queryPattern as the WHERE-clause of query;
    return query
end
```

Hoofdstuk 5

Evaluatie systeem

Het ontworpen systeem werd tijdens de ontwikkeling enkel getest met virtuele gegevens van beperkte omvang. Om te achterhalen of de implementatie gebruikt kan worden voor modellen van realistische grootte werden er testen uitgevoerd. Er werden testen voorzien voor twee verschillende domeinen: schaalbaarheid en praktische bruikbaarheid. Bij de schaalbaarheidstesten wordt onderzocht hoe het systeem reageert indien bepaalde simulatieparameters wijzigen (paragraaf 5.1). Daarnaast werd een realistische case uitgewerkt die moet aantonen hoe het systeem in de praktijk gebruikt wordt en welke praktische problemen er worden ondervonden (paragraaf 5.2).

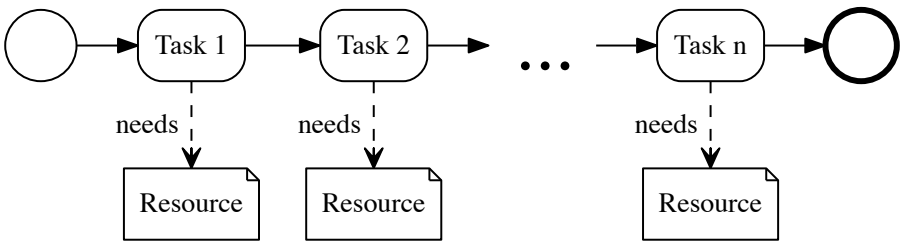
5.1 Schaalbaarheid

Om het systeem te testen op schaalbaarheid werden meerdere testen uitgevoerd. Elke test onderzoekt het effect van een specifieke simulatieparameter op de duur van een gemiddelde simulatie.

Allereerst wordt de algemene testopstelling en enkele begrippen besproken. Vervolgens worden de verschillende testen besproken in hun paragraaf. Tot slot wordt een algemene conclusie gegeven.

5.1.1 Algemene testopstelling

In alle testen wordt een lineair procesmodel gebruikt waarin alle taken elkaar opvolgen. Elke taak in het proces vereist één bron van hetzelfde type alvorens het kan worden uitgevoerd (zie Figuur 5.1 voor een voorbeeld) . Doordat de taken elkaar opvolgen zal eenzelfde procesinstantie op eenzelfde moment nooit meer dan één bron gebruiken. Het gelijktijdig gebruik van meerdere bronnen wordt dan enkel beïnvloed door het aantal simultane procesinstanties.



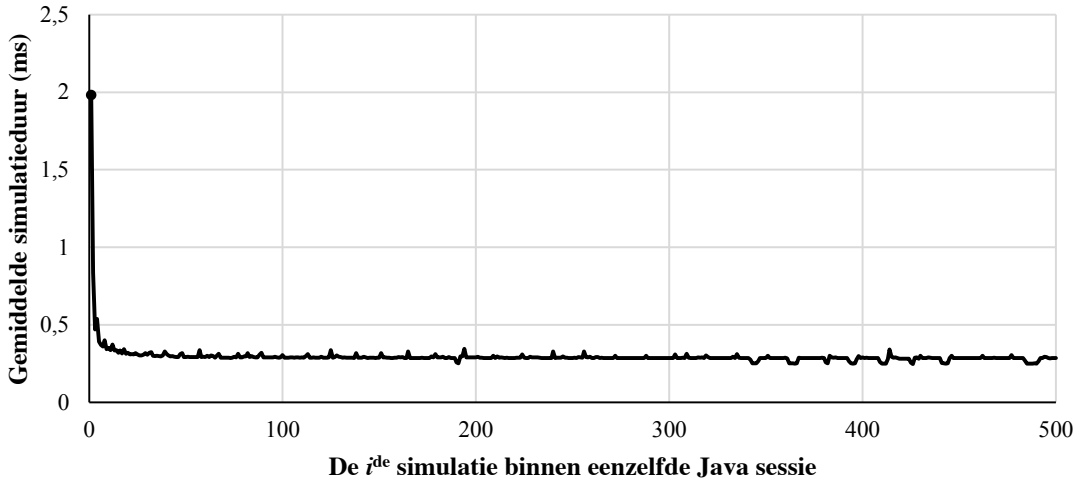
Figuur 5.1: Een voorbeeld van een procesmodel dat gebruikt werd in de testen.

Het aantal simultane procesinstanties is afhankelijk van de snelheid waartegen een nieuwe procesinstantie wordt opgestart. Deze waarde noemt men de *rescheduling*-parameter. Een kleinere waarde zorgt ervoor dat er sneller nieuwe procesinstanties worden aangemaakt. Indien dit sneller gebeurt dan de duur van een instantie, dan zullen er meerdere procesintanties simultaan voorkomen tijdens een simulatie. Doordat de taken in deze testen dezelfde (virtuele) duur hebben, zal de rescheduling-parameter worden uitgedrukt in *aantal taken*[12].

Elke test onderzoekt het effect van een specifieke parameter op de duur van een simulatie. In totaal worden er vier parameters getest: de grootte van het proces, de grootte van de ontologie, het aantal bronnen en het tekort aan bronnen in een simulatie. De testen van eenzelfde parameter werden uitgevoerd in eenzelfde Java-sessie die werd gestart met 500 simulaties voor elke allocatiemethode. Op deze manier was verzekerd dat de metingen zo min mogelijk beïnvloed werden door het vertragende effect van *class loading* in Java[6] (zie Figuur 5.2 voor een illustratie van deze *warmup*-fase). Per waarde van de te testen parameter werd de gemiddelde uitvoeringstijd bepaald van 500 simulaties (de eerste 50 simulaties werden niet in rekening gebracht, nogmaals om class loading te vermijden). Dit zijn de waarden die werden geplot in de verschillende grafieken uit dit hoofdstuk. De testen werden uitgevoerd voor alle allocatiemethoden besproken in hoofdstuk 4. Op basis hiervan wordt een vergelijkende studie gemaakt.

Ten slotte wordt vermeld dat in de testen onder deze paragraaf geen gebruik gemaakt wordt van een reasoner. Deze keuze werd gemaakt omdat het zeer moeilijk is om een willekeurige, representatieve ontologie op te stellen. De complexiteit hiervan hangt namelijk af van meerdere factoren zoals het aantal axioma's en de grootte vaan de klassenhiërarchie. Bijkomende testen zouden het effect van de complexiteit van een ontologie op de duur van een simulatie kunnen testen.

[12]Bijvoorbeeld: gegeven procesmodel met zes taken. Een rescheduling-waarde van twee zorgt ervoor dat tijdens de simulatie ongeveer drie procesinstanties gelijktijdig worden afgehandeld.



Figuur 5.2: Een illustratie van de opwarmingsfase van de *Java Virtual Machine* (JVM) voor de gemaakte implementatie.

Alle testen werden uitgevoerd op een server van het iMinds *iLab.t technical testing centre*[13]. Dit serverpark wordt door iMinds beheerd en ter beschikking gesteld aan onderzoekers. Het bestaat uit servers met speciale hardwareconfiguraties, onder meer bedoeld om computerintensieve taken uit te voeren. De finale testen werden alle uitgevoerd op dezelfde machine. De specificaties hiervan zijn terug te vinden in tabel 5.1.

Tabel 5.1: De specificaties van de machine gebruikt voor de testen uit paragraaf 5.1.

Besturingssysteem	Ubuntu 14.04.1 LTS
Aantal processoren	24
Model processoren	Intel® Xeon® Processor E5645
Snelheid processoren	2.4 GHz
Totaal werkgeheugen	6×4096 MB
Model werkgeheugen	Kingston® 9965426-059.A00LF
Type werkgeheugen	DDR3 SDRAM[14]
Snelheid werkgeheugen	1333 MHz
Model harde schijf	Hitachi® HTS723225A7A364

5.1.2 Grootte van het proces

In deze test wordt onderzocht in hoeverre de duur van een simulatie beïnvloed wordt door de grootte van het procesmodel. Zoals eerder vermeld wordt er een lineair procesmodel gebruikt.

[13]iLab.t technical testing centre, `http://ilabt.iminds.be` (laatst geraadpleegd op 05/06/15)

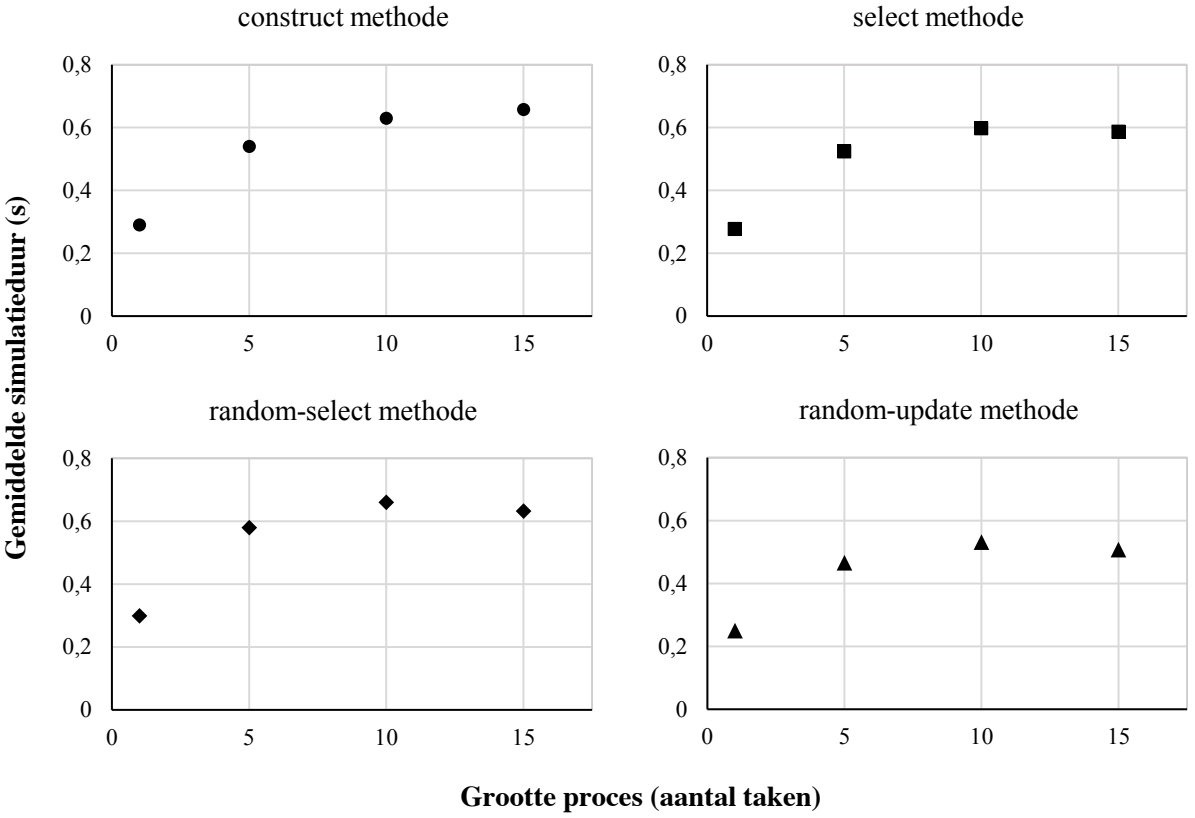
[14]DDR3 SDRAM, *Double Data Rate type Three Synchronous Dynamic Random-Access Memory*

De grootte geeft aan uit hoeveel taken het proces bestaat en varieert van één tot vijftien.

Gemiddeld wordt er om de drie taken een nieuwe procesinstantie opgestart. Hierdoor zullen er ongeveer drie processen parallel worden afgehandeld. Er worden tien bronnen voorzien in het proces. Verder bevat de ontologie naast de bronnen geen andere entiteiten.

Resultaten

De resultaten van deze test worden weergegeven in figuur 5.3. Uit deze grafiek blijkt de marginale impact op de duur van een simulatie kleiner wordt naarmate het aantal taken toeneemt. Hoewel de verschillende allocatiemethoden anders presteren, is dit effect gelijk voor alle methoden.



Figuur 5.3: De gemiddelde duur van een simulatie in functie van de grootte van een procesmodel.

Discussie

Deze resultaten zijn te verklaren met de stopvoorwaarden van de simulator. Deze zijn zoals eerder besproken (zie paragraaf 2.2.2) afhankelijk van de convergentie van de analyses van de simulator. Bij een groter procesmodel zijn er meer waarden die moeten convergeren. Deze convergentie gebeurt echter (ongeveer) gelijktijdig omdat er met een lineair procesmodel wordt gewerkt.

Indien er met complexere procesmodellen wordt gewerkt, dan zal dit een groter effect hebben op de convergentie van de simulator. Het gebruik van splitsingen en events kan er namelijk voor zorgen dat er meerdere paden bewandeld kunnen worden. De convergentie van de analyses kan daardoor langer uitvallen. Omdat het niet eenvoudig is om een willekeurig, complex procesmodel op te stellen werd dit in deze testen achterwege gelaten. Een bijkomende studie zou de impact van de complexiteit van een procesmodel kunnen testen voor deze simulator.

5.1.3 Het aantal bronnen

In deze test wordt onderzocht of het aantal bronnen een effect heeft op de performantie van het ontwikkelde systeem. Hiervoor worden meerdere testen uitgevoerd op eenzelfde procesmodel (bestaande uit vijf taken) met een rescheduling-factor gelijk aan drie (± twee gelijktijdige procesinstanties). Het aantal bronnen varieert van 5 tot 500. De ontologie bevat naast deze bronnen geen andere entiteiten.

Resultaten

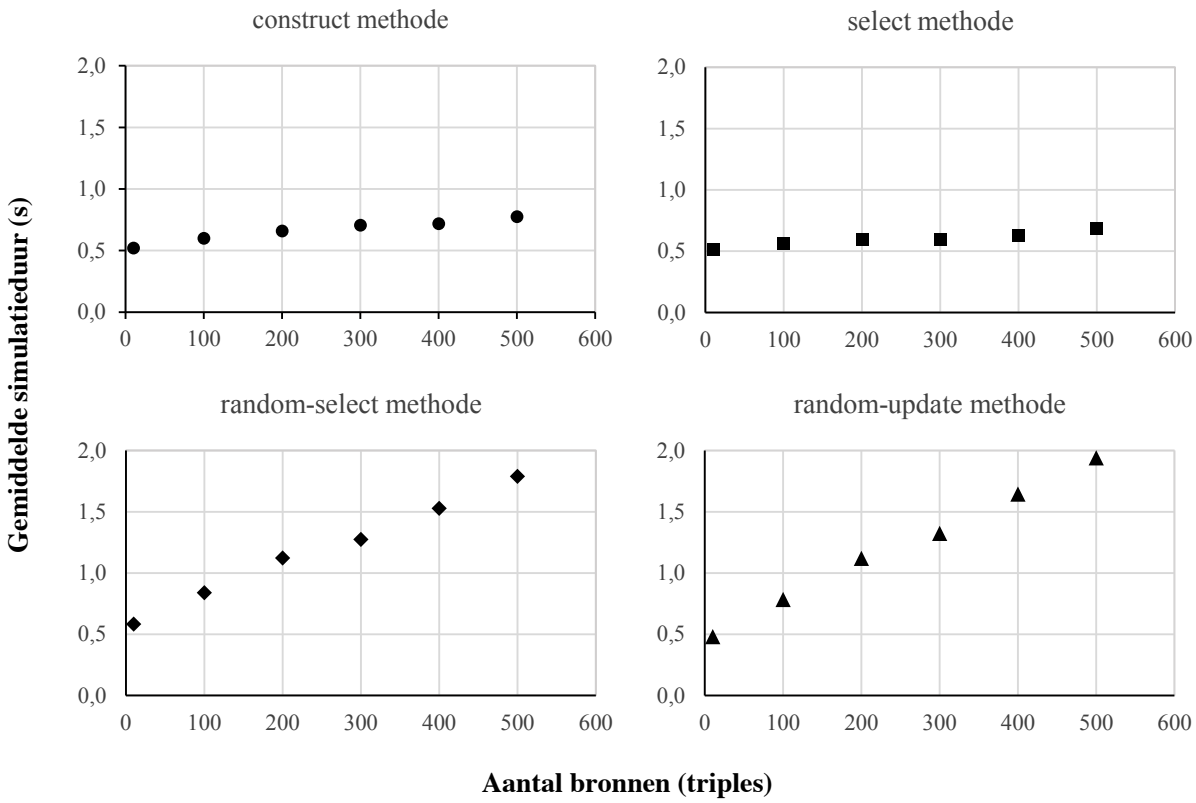
De resultaten van horende bij deze testopstelling worden weergegeven in figuur 5.4. Op deze grafiek is te zien dat het effect op de duur van een simulatie anders is voor de verschillende methoden. De methoden waarbij een willekeurige bron rechtstreeks wordt gekozen via SPARQL (`random-select` en `random-update` methoden) zijn beduidend trager wanneer het aantal bronnen toeneemt. Bij een klein aantal bronnen blijken deze methoden hier minder hinder van te ondervinden.

Discussie

Een *beste* methode blijkt hier niet te bestaan. Wel kan gezegd worden dat de `select` methode over het algemeen goed scoort, zowel bij een klein als bij een groot aantal bronnen. De `random-select` en `random-update` methoden vragen meer tijd omwille van de willekeurige selectie in SPARQL. Zij dienen alle bronnen te sorteren, alvorens één willekeurige kan gekozen worden. Dit zorgt voor bijkomende instructies die in de methoden `construct` en `select` worden vermeden.

5.1.4 Grootte van de ontologie

De vorige test onderzoekt het effect van het aantal bronnen in de ontologie. Hiermee werden de entiteiten bedoeld waarnaar verwezen wordt vanuit de regels. Een ontologie kan uit meer entiteiten bestaan, bijvoorbeeld indien er verschillende types van bronnen gebruikt worden. Daarom wordt in deze test onderzocht wat de invloed is van andere entiteiten in het model, naast de gemodelleerde bronnen.



Figuur 5.4: De gemiddelde duur van een simulatie in functie van het aantal bronnen in de ontologie.

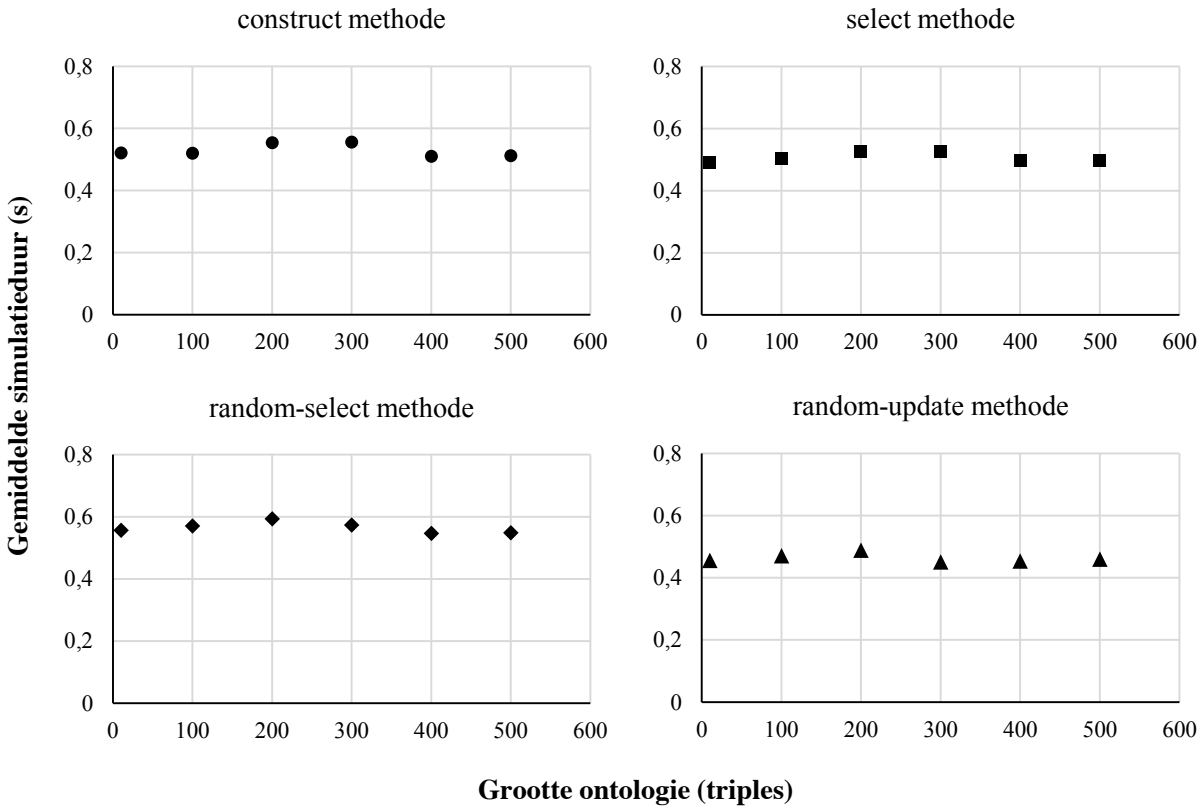
De opstelling is gelijkaardig aan de vorige test. Er wordt gewerkt met een vast procesmodel (vijf taken), een rescheduling-factor van drie. Daarnaast bevat de ontologie steeds vijfentwintig bronnen. Naast deze bronnen worden een verschillend aantal willekeurige triples toegevoegd aan het kennismodel.

Resultaten

De grafiek uit Figuur 5.5 toont de resultaten van deze test. Hieruit kan er afgeleid worden dat het aantal entiteiten in de ontologie een zeer klein effect heeft op de simulatietijd. De impact is gelijk voor de verschillende allocatiemethoden.

Discussie

De conclusie dat de grootte van de ontologie weinig effect heeft moet met enige voorzichtigheid gehanteerd worden. Doordat er in deze testen niet gewerkt wordt met een reasoner, is de ontologie niet meer dan een semantisch net. Doordat het zeer moeilijk is om een willekeurige en complexe ontologie op te stellen, werd dit in deze testen achterwege gelaten.



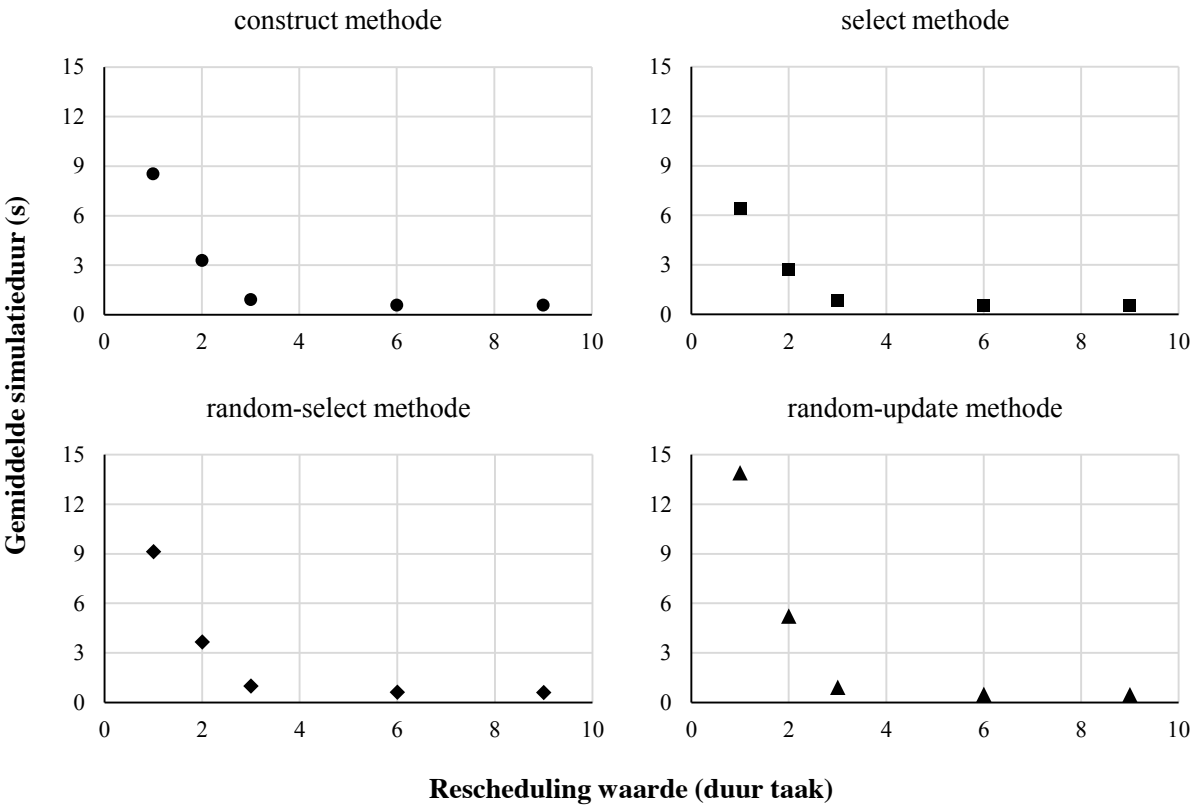
Figuur 5.5: De gemiddelde duur van een simulatie in functie van het aantal andere entiteiten in de ontologie (naast bronnen).

5.1.5 Tekort aan bronnen

In deze test wordt onderzocht in hoeverre een tekort aan bronnen de duur van een simulatie beïnvloedt. Dit tekort kan optreden indien een beperkt aantal bronnen dienst staat voor een groot aantal procesinstanties die gelijktijdig worden uitgevoerd. Om dit te testen worden verschillende waarden voor de rescheduling-parameter getest. Het procesmodel (negen taken) en het aantal bronnen (drie) blijft constant. Naast de bronnen bevat de ontologie geen bijkomende gegevens. De verschillende rescheduling-waarden die getest worden zijn: 1, 2, 3, 6 en 9. De waarde *één* en *drie* zullen respectievelijk voor ongeveer negen en drie parallelle processen zorgen. Aangezien er slechts drie bronnen aanwezig zijn in het proces, zullen sommige procesinstanties worden gepauzeerd totdat er terug bronnen beschikbaar zijn.

Resultaten

De resultaten van bovenstaande testopstelling zijn terug te vinden in de grafiek uit Figuur 5.6. Hierop is te zien dat een tekort aan bronnen in het proces een grote impact heeft op de duur van een simulatie. Dit is geldig voor alle allocatiemethoden, al komt de `select` methode er ook hier beter uit dan de andere methoden.



Figuur 5.6: De gemiddelde duur van een simulatie in functie van de rescheduling-factor die een effect heeft op het aantal parallelle procesinstanties.

Discussie

Indien een bron niet beschikbaar is, dan wordt de uit te voeren taak gepauzeerd. Bij de volgende ronde zal de simulator opnieuw controleren of de vereiste bronnen beschikbaar zijn. Bij deze controles worden telkens regels uitgevoerd op de ontologie, wat een complexe operatie is in vergelijking met de andere taken van de simulator. Daardoor neemt de duur van een simulatie sterk toe.

In principe mag dit geen impact hebben op de uitvoering van een simulatie. De simulator weet immers op voorhand hoe lang een taak zal duren. Hieruit kan afgeleid worden wanneer de bronnen van die taak worden vrijgegeven. De controle voor beschikbare bronnen moet enkel op die moment uitgevoerd worden. Wegens tijdsgebrek werd dit niet meer geoptimaliseerd in deze implementatie.

5.1.6 Conclusie

Uit de testen blijkt dat de grootte van het procesmodel, het aantal bronnen en een tekort aan bronnen een negatief effect hebben op de duur van een simulatie. Het grootste effect is af te

lezen bij een tekort aan bronnen in het proces. Een aanpassing aan de simulator zou dit negatief effect kunnen minimaliseren, door onder andere rekening te houden met de duur van een taak.

De verschillende parameters hebben hetzelfde effect op de simulatieduur voor de verschillende allocatiemethoden besproken in hoofdstuk 4. De grootte van het effect is echter verschillend. Op basis van de resultaten van de verschillende testen kan gezegd worden dat de `select` methode over het algemeen zorgt voor de laagste simulatietijden. Op basis van dit resultaat werd ervoor gekozen deze methode standaard te gebruiken in de implementatie.

5.2 Bruikbaarheid

In dit tweede deel van het hoofdstuk *Evaluatie systeem* wordt onderzocht of het ontwikkelde systeem praktisch gezien gebruikt kan worden voor de simulatie van realistische gegevens. Er werd daartoe een realistische case uitgewerkt waarbij alle stappen werden doorlopen die nodig zijn om uiteindelijk een simulatie uit te voeren. Het optimaliseren van dit proces (en bijhorende kennisgebaseerd systeem) behoort niet tot het opzet van deze test.

5.2.1 Realistische Case

Het proces uit de case handelt over de operatie van een patiënt in een ziekenhuis. Het begint bij de verplaatsing van de patiënt naar het operatiekwartier. Vervolgens wordt een eerste anesthesie toegediend. Vanaf dan vertrekt de patiënt naar een operatiekamer waar ook de effectieve operatie wordt uitgevoerd. Wanneer deze is afgelopen, gaat de patiënt naar recoveryruimte totdat hij/zij terug bij zijn positieven is. De patiënt wordt dan terug naar de oorspronkelijke afdeling gebracht. Op basis van documentatie afkomstig van interviews werd een BPMN-procesmodel opgesteld. Het hoofdproces hiervan wordt getoond in Figuur 5.7.

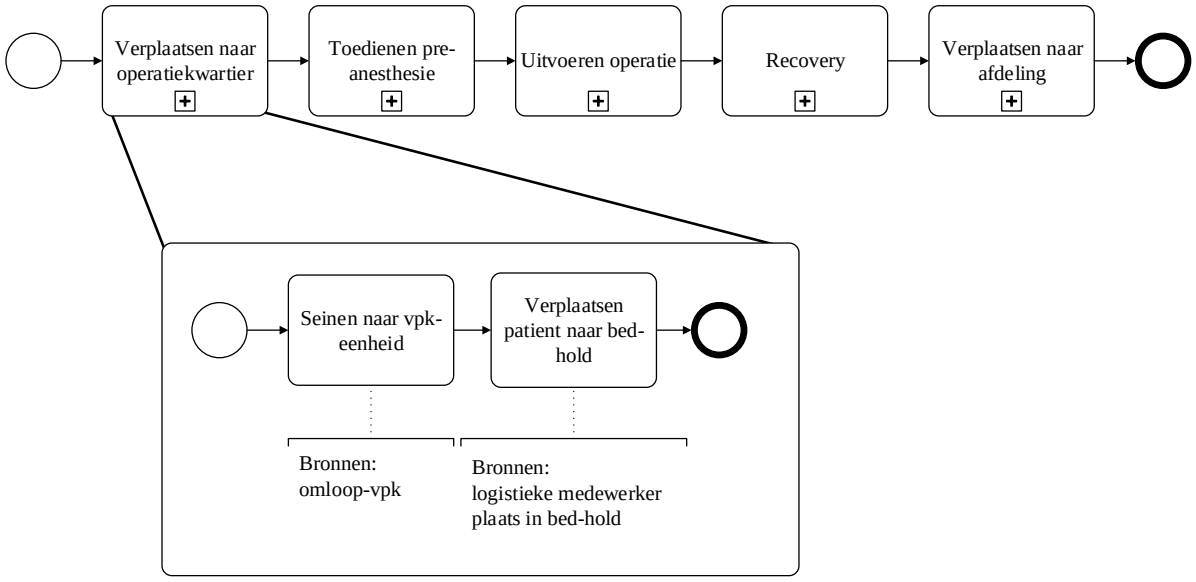
Procesmodel

Het procesmodel voor de test case werd uitgetekend met behulp van software van Amaron. Deze software werd ter beschikking gesteld aan de onderzoeksgroep bij de samenwerking met het HIPS-project.

Omwille van de grootte van het proces, werd dit opgedeeld in subprocessen. De koppeling van een kind proces naar zijn ouder proces werd verwezenlijkt via *call activities* uit BPMN.

Bronnen

Op basis van de voorziene documenten werden de aanwezige bronnen uit het proces geïdentificeerd. De ontologieën van het ACCIO-project[14] werden gebruik als basis voor het modelleren



Figuur 5.7: Een illustratie van het hoofdproces dat gebruikt werd in de testen uit paragraaf 5.2, waarbij één subproces wordt verduidelijkt.

van de bronnen. Deze ontologieën voorzien een algemene structuur voor het modelleren van een ziekenhuis gerelateerde domeinkennis. Voor de modellering werd gebruik gemaakt van de Protégé editor[15]. Enkele specificaties van de uiteindelijke ontologie worden getoond in Tabel 5.2.

Tabel 5.2: Specificaties van de ontologie gebruikt tijdens de simulatie van de realistische case.

Aantal axioma's	2241
Aantal klassen	295
Aantal object properties	157
Aantal data properties	41
Aantal individuals	116
DL expressiviteit	$\mathcal{SHOIQ}$(D)

Om de bronnen te kunnen alloceren tijdens een simulatie, werden er regels opgesteld. De beschreven regels voerden voornamelijk statuswijzigingen uit op de gemodelleerde bronnen. Dit liet toe kennis te maken met het modelleren van de regels, zonder dat dit te veel tijd in beslag nam.

Configuratiebestanden

Opdat dit proces gesimuleerd kan worden met het gebruikte simulatieframework, zijn enkele bijkomende documenten nodig. Allereerst moet elk type bron kenbaar worden gemaakt aan de

[15]Protégé, `http://protege.stanford.edu` (laatst geraadpleegd op 05/06/15)

simulator. Dit gebeurt aan de hand van een `*.resources.xml`-bestand. Hierin zijn de volgende gegevens vereist:

- de ontologie waarin dit type bron wordt beheerd
- het RIF-document waarin de regels voor allocatie worden beschreven
- het RIF-document waarin de regels voor deallocatie worden beschreven
- proces specifieke informatie zoals de kost voor het gebruik van de bron

Om aan te geven welke bronnen nodig zijn voor welke taken in het proces, wordt gebruik gemaakt van een `*.attributes.xml`-bestand. Dit dient te worden aangemaakt voor het hoofdproces en elk subproces. In het bestand worden per taak (onder andere) de volgende gegevens beschreven:

- de verschillende bronnen die vereist zijn voor de uitvoering van de taak
- de duur van de taak
- de kost voor de uitvoering van de taak

5.2.2 Simulatie

Het gemodelleerde proces met de bijhorende ontologie en regels werd gesimuleerd met het ontworpen systeem uit deze thesis. Aangezien de simulator enkel kan werken met procesmodellen zonder subprocessen, werd deze eerst aangepast. De aanpassing maakt het mogelijk dat simulaties uitgevoerd kunnen worden, waarbij de procesmodellen worden opgevraagd via een REST-serice[16].

Voor de simulatie van deze case werd gebruik gemaakt van een test-server van iMinds. Dit was nodig aangezien het werk van de reasoner op de modellen van ACCIO zeer computerintensief is. De opstelling van deze test-case vereiste ongeveer 4.5 GB werkgeheugen.

5.2.3 Bevindingen

De tool van Amaron bezit de belangrijkste functionaliteit voor het beschrijven van een procesmodel. Hoewel bepaalde functionaliteit nog in ontwikkeling is, verliep de modellering zelf redelijk vlot. De ontbrekende functionaliteit werd opgevangen door manuele aanpassingen in het procesmodel (bv. het aanpassen van de identificatie van een proces). Dit vereiste wel de nodige kennis van XML.

Voor de modellering van de bronnen werd beroep gedaan op Protégé. Dit is een ruim aanvaarde manier voor het modelleren van ontologieën. Deze tool zorgde dan ook voor weinig tot geen problemen.

[16] REST, REpresentational State Transfer

Het schrijven van de regels daarentegen lag moeilijker. Deze worden door het ontworpen systeem ingelezen in de XML-schrijfwijze van RIF. Software die regels in de presentation syntax kan omzetten naar de XML-schrijfwijze, werd niet gevonden. De modellering van de regels vereiste daardoor veel werk, was foutgevoelig en onoverzichtelijk.

Tijdens het modelleren van de bronnen viel op dat niet alle functionaliteit van ACCIO gebruikt zou kunnen worden. Zo is het niet mogelijk om locaties te gebruiken in het simulatieframework. Ook andere context-specifieke informatie kan moeilijk achterhaald worden (bv. de risicofactor van een patiënt, de relatie tot een patient, etc). Dit is zeer jammer aangezien de troef van een kennisgebaseerd systeem vaak ligt bij het context bewuste karakter. Uitbreidingen die meer context specifieke informatie kunnen voorzien zouden daarom de toepasbaarheid van het ontwikkelde systeem kunnen verruimen. Een dergelijke uitbreiding zal echter niet altijd mogelijk zijn. Zo vereist het werken met locaties een andere soort simulator. De situatie-specifieke informatie die wél al achterhaald kan worden (bv. het tijdstip van de allocatie, de patiënt, de taak in het BPMN-proces), wordt in ontwikkelde implementatie gesubstitueerd in de query.

De documenten die extra informatie geven over het proces worden ook ingelezen in XML. De configuratie hiervan is redelijk eenvoudig, al wordt dit snel een tijdrovende bezigheid indien de omvang van het proces en het aantal types bronnen toeneemt.

5.2.4 Conclusie

Uit de simulatie van de realistische case blijkt dat het mogelijk is om een procesmodel en ontologie van realistische grootte te simuleren met behulp van de ontwikkelde uitbreidingen op de simulator. Het opstellen van de regels en de nodige configuratiebestanden vraagt echter veel werk omwille van de schrijfwijze in XML. Voor de regels kan dit worden opgelost door ook ondersteuning te bieden voor andere notaties zoals de Presentation Syntax van RIF-BLD. Voor het opstellen van de configuratiebestanden kan een grafische gebruikersomgeving hulp bieden.

Een tweede conclusie is dat met het huidige ontwerp van de simulator niet veel situatiespecifieke gegevens achterhaald kunnen worden. Dit is een grote beperking voor een systeem dat deels bedoeld is voor de evaluatie van contextbewuste systemen. Naar de toepasbaarheid van het systeem toe is het daarom interessant om aanpassingen te voorzien die deze functionaliteit uitbreiden. Bronnen zouden dan geselecteerd kunnen worden op een meer intelligente en contextbewuste manier. Omwille van beperkingen van het gebruikte framework, zal een dergelijke aanpassing niet altijd mogelijk zijn (bv. bij het simuleren van locaties).

Hoofdstuk 6

Conclusies

6.1 Besluit

Kennisgebaseerde systemen kennen toepassingen in verschillende domeinen, waaronder procesondersteundende software. Daar kan een dergelijk systemen gebruikt worden voor het toekennen van bronnen aan taken in het proces. Hoewel deze systemen een hoge mate van intelligentie toelaten, weet men niet op voorhand wat hun effect zal zijn op de werking van het proces. Om dit te achterhalen moet het systeem getest worden. Deze testen uitvoeren met behulp van simulatie heeft verschillende voordelen (onder andere een lagere kost en een gecontroleerde omgeving), maar is moeilijk te realiseren voor een kennisgebaseerd systeem. De simulator moet namelijk het gedrag van het te testen systeem kunnen emuleren, alsook de omgeving waarin het gebruikt zal worden. Dit vereist een specifiek stuk software dat afhankelijk is van het te testen systeem. Daarbij is de ontwikkeling hiervan een tijdrovende en inefficiënte manier van werken. In deze thesis wordt daarom een alternatief voorgesteld waarin een willekeurig kennisgebaseerd systeem getest kan worden met behulp van processimulatie voor het toekennen van bronnen.

Allereerst werden verschillende eisen opgesteld waaraan het systeem diende te voldoen. Vervolgens werd een abstract model opgesteld dat als basis gebruikt werd voor een implementatie. Deze werd als uitbreiding ontwikkeld op de bestaande processimulator van onderzoeksgroep IBCN. Met deze uitbereiding is het mogelijk om bronnen toe te kennen in een proces op basis van een ontologie en bijhorende regels. Als laatste werd de implementatie onderworpen aan verschillende testen om hieruit de bruikbaarheid en schaalbaarheid van het systeem te achterhalen.

Wanneer wordt teruggeblikt op de implementatie, kan gezegd worden dat deze thesis in zijn opzet is geslaagd. Het ontworpen systeem laat namelijk toe dat een willekeurig kennisgebaseerd systeem[17] via processimulatie getest kan worden voor het beheer van bronnen. Dit vormt meteen

[17] bestaande uit een OWL-ontologie en RIF-regels

ook een antwoord op de onderzoeksvraag van deze thesis:

> Hoe kan processimulatie gebruikt worden om kennisgebaseerde systemen die instaan voor het beheer van bronnen, te testen en evalueren op een generieke manier?

Wel moet gezegd worden dat de implementatie leidt onder bepaalde beperkingen van de simulator. Specifieke functionaliteit, zoals het werken met locaties van patiënten, is bijvoorbeeld niet mogelijk. Voor kennisgebaseerde systemen die zeer contextgericht zijn kan deze implementatie daardoor tekort schieten. Ook aan de praktische bruikbaarheid van het systeem kan nog gewerkt worden. Uitbreidingen op deze implementatie kunnen een oplossing bieden voor deze problemen.

6.2 Mogelijke uitbereidingen

Het werk dat in deze studie werd verricht opent de deuren voor meerdere uitbreidingen die de toepasbaarheid van het ontwikkelde systeem kan verruimen. Enkele van deze uitbreidingen worden in de paragrafen hieronder kort toegelicht.

6.2.1 Gebruiksvriendelijkheid

Op basis van de testresultaten uit hoofdstuk 5 werd geconcludeerd dat het vrij complex is om een enkele simulatie op te stellen voor een realistische use case. Met de hulp van een ondersteunende GUI zou de nodige configuratie vereenvoudigd kunnen worden. Dit laat toe dat het systeem toegankelijker wordt voor gebruikters die geen kennis hebben van de interne werking van de simulator.

Ook het modelleren van de regels zou vereenvoudigd kunnen worden, wat momenteel via de XML-schrijfwijze dient te gebeuren. Door andere notaties te ondersteunen, kan de tijd verkleind worden die nodig is om een groot aantal regels te schrijven. Hiertoe kan bijvoorbeeld beroep gedaan worden op JavaCC[18].

6.2.2 Context bewuste simulatie

Zoals in de evaulatie van het systeem werd bevonden, is het gebruik van contextspecifieke regels nog redelijk beperkt. Er kunnen uitbreidingen op de simulatiesoftware worden voorzien die meer gegevens ter beschikking stellen aan het allocatiemechanisme van de simulator. Dit opent de deur naar de evaluatie van meer contextspecifieke systemen.

[18] JavaCC, The Java Parser Generator `https://javacc.java.net` (laatst geraadpleegd op 05/06/15)

6.2.3 Ondersteuning regeltalen

Momenteel is de implementatie beperkt tot de interpretatie van OWL-ontologieën en bijhorende regels beschreven in RIF. Hoewel RIF zorgt voor een meer generisch ontwerp, maken weinig implementaties van kennisgebeerde systemen hier daadwerkelijk gebruik van. Opdat deze toch getest kunnen worden moeten de gebruikte regels worden omgezet naar RIF. Uitbreidingen die dit proces automatiseren zullen bevorderend zijn voor de bruikbaarheid van het ontworpen systeem. Aangezien de ontwikkelde implementatie intern reeds een vertaling uitvoert, dienen deze bijkomende vertalingen met de nodige voorzichtigheid te worden ontwikkeld.

6.2.4 Automatische regeloptimalisatie

In deze studie worden bronnen in een proces toegekend op basis van een kennismodel en bijhorende regels. Vaak kunnen deze regels op meerdere manieren worden opgesteld, rekening houdende met verschillende factoren. Personeel kan bijvoorbeeld geselecteerd worden op basis van locatie, competenties, specifieke situaties en meer. Wat de effecten zijn van deze verschillende allocatiestrategieën is vaak moeilijk te voorspellen. Met behulp van het systeem uit deze thesis kan op een relatief eenvoudige wijze het effect deze verschillende regelsets op eenzelfde proces achterhaald worden. Naarmate er echter meer selectiemogelijkheden zijn, neemt het aantal mogelijke regelsets snel toe. Daarom kan het nuttig zijn om het systeem uit deze thesis te gebruiken als basis voor een automatische regeloptimalisator.

Een dergelijke optimalisator zou op basis van reeds bestaande regels, nieuwe regels construeren op een iteratieve manier. Hiertoe dient men eerst te achterhalen uit welke bouwstenen deze regels opgebouwd zijn. Op basis van deze bevindingen kunnen er automatisch regels geconstrueerd worden die men vervolgens kan testen met het systeem uit deze thesis. Door vooraf op een formele manier de KPI's van het proces vast te leggen, kunnen de verschillende regelsets op een eenduidige manier beoordeeld worden.

Aangezien het testen van werkelijk *alle* mogelijke regels niet realistisch is, dienen hiervoor gepaste algoritmen te worden gebruikt, bijvoorbeeld met *simulated annealing*. Het opstellen van de regelsets kan gebeuren met behulp van genetische algoritmen. Omdat geweten is dat regels opgebouwd zijn uit kleinere bouwstenen (onder andere predicaten en termen) zijn de mutatie en kruising van meerdere regels zeker mogelijk.

Bijlage A

Willekeurige selectie met behulp van SPARQL

In paragraaf 4.2.2 wordt besproken op welke manier bronnen gealloceerd kunnen worden met behulp van een kennisgebaseerd systeem. Er kwam hierbij een probleem naar boven indien meerdere bronnen voldoen aan het antecedent van eenzelfde regel. Per aanvraag van de simulator mag er slechts één bron gealloceerd worden in het kennisgbaseerd systeem. Om dit op te lossen werd er voorgesteld één bron te selecteren uit de verschillende beschikbare bronnen (de bronnen die voldoen aan het antecedent van de regel). In deze appendix wordt onderzocht hoe dit rechtstreeks kan gebeuren met de uitvoering een enkele SPARQL-query.

Concreet wordt er gezocht naar queries die een willekeurige deelverzameling van nodes selecteren uit een RDF-graaf op basis van een gegeven query pattern. De twee factoren die bepalen of een oplossing *goed* is, zijn de snelheid waartegen de query kan worden uitgevoerd en de willekeurigheid van de bekomen oplossing.

In paragraaf A.1 worden eerst de verschillende geteste queries toegelicht, waarna de uitgevoerde testen en resultaten worden besproken in paragraaf A.2. Op basis van deze resultaten wordt een conclusie getrokken in paragraaf A.3.

A.1 Queries

Om queries op te stellen die voldoen aan de intentie van dit onderzoek, werd gezocht naar beschrijvingen in de literatuur. Deze werden niet direct gevonden, waardoor beroep werd gedaan op de SPARQL-community voor het vinden van gepaste queries. De verschillende voorstellen worden in overeenkomstige paragrafen besproken.

A.1.1 StandardQuery

Om te beginnen wordt de code gegeven van een gewone SPARQL-query. Deze werd mee onderworpen aan de testen opdat een vergelijking kon worden gemaakt met de uitvoering van willekeurige SPARQL-queries (voornamelijk op vlak van uitvoeringssnelheid). De code hiervan is terug te vinden in Codevoorbeeld A.1.

```
SELECT ?s
WHERE { ?s ?p ?o }
LIMIT x
```

Codevoorbeeld A.1: Voorbeeld van de geteste, standaard SPARQL-query.

A.1.2 Sorteren op basis van RAND()

De twee paragrafen onder deze titel bespreken elk een type query dat de resultaten sorteert met behulp van de RAND-functie[19] uit SPARQL. Deze functie genereert een willekeurig getal tussen nul (inclusief) en één (exclusief).

BindRandVar

In deze query wordt een willekeurige waarde (afkomstig van de RAND-functie) gebonden aan een variabele, waarna op deze variabele wordt gesorteerd (Codevoorbeeld A.2). Deze methodiek vereist dat de RAND-functie wordt herberekend per oplossing van de query. Of dit effectief gebeurt blijkt afhankelijk te zijn van de gebruikte SPARQL-implementatie. In Apache Jena gebeurt dit op de vereiste manier (dit in tegenstelling tot bijvoorbeeld Sesame[20], waar de RAND-functie één keer wordt berekend per uitvoering van een query).

OrderByRand

Een gelijkaardige query ten opzichte van de vorige methode ordent rechtstreeks op basis van de RAND-functie (Codevoorbeeld A.3).

[19] 'Returns a pseudo-random number between 0 (inclusive) and 1.0e0 (exclusive). Different numbers can be produced every time this function is invoked. Numbers should be produced with approximately equal probability.' [33]

[20] Sesame, `https://rdf4j.org/` (laatst geraadpleegd op 14/05/15)

```
SELECT ?s
WHERE {
    ?s ?p ?o .
    BIND(RAND() AS ?randVar)
}
ORDER BY ?randVar
LIMIT x
```

Codevoorbeeld A.2: Voorbeeld van de BindRandVar-methode.

```
SELECT ?s
WHERE { ?s ?p ?o }
ORDER BY RAND()
LIMIT x
```

Codevoorbeeld A.3: Voorbeeld van de OrderByRand-methode.

A.1.3 Permuteren van de oplossingenverzameling

Een tweede manier om een willekeurige oplossingenverzameling te bekomen is door gebruik te maken van de OFFSET-clausule uit SPARQL. Deze structuur laat toe dat er wordt verschoven in een geordende oplossingenverzameling.

RandOffset

In deze query wordt een random offset berekend op basis van de RAND- en COUNT-functie[21] van SPARQL. De berekening wordt rechtstreeks gebruikt in de OFFSET-clausule.

Het blijkt echter niet mogelijk om een dergelijke query uit te voeren met Apache Jena. Het framework laat niet toe dat variabelen en / of functies worden gebruikt in de OFFSET-clausule. De query wordt getoond in Codevoorbeeld A.4.

ExternalRandomOffset

Door bij de constructie van de query een willekeurig getal te plaatsen in de OFFSET-clausule kan alsnog deze methodiek gebruikt worden. De willekeurige *offset* wordt bijgevolg bepaald vóór de uitvoering van de query en gesubstitueerd op de correcte plaats (zie Codevoorbeeld A.5 voor een voorbeeld).

[21] 'Count is a SPARQL set function which counts the number of times a given expression has a bound, and non-error value within the aggregate group.' [33]

```
SELECT ?s
WHERE { ?s ?p ?o }
ORDER BY ?s
OFFSET (
    RAND() * COUNT(distinct ?s)
)
LIMIT x
```

Codevoorbeeld A.4: Eerste poging van de RandOffset-methode.

Uit de testen blijkt het zeer moeilijk te zijn om op basis van deze manier oplossingen te bekomen. Er zullen geen resultaten zijn indien de verschuiving groter is dan de grootte van de oplossingenverzameling. De resultaten vormen geen circulaire lijst, waardoor de grootte van de verschuiving beperkt moet blijven. Indien het aantal oplossingen niet op voorhand is geweten, kan deze methode bijgevolg niet gebruikt worden. Indien de grootte van de oplossingenverzameling wél op voorhand geweten is, is dit wel een mogelijkheid. Dit wordt in de resultaten apart besproken onder de naam *KnownExternalRandomOffset*.

```
SELECT ?s
WHERE { ?s ?p ?o }
ORDER BY ?s
OFFSET <random inserted number>
LIMIT x
```

Codevoorbeeld A.5: Voorbeeld van de ExternalRandomOffset-methode.

A.1.4 Bemonsteren van de oplossingenverzameling

De laatste queries die besproken worden maken gebruik van de SAMPLE-functie[22] uit SPARQL. Deze functie selecteert op een arbitraire wijze één bron uit de oplossingenverzameling.

SampleVar

De eerste query maakt gebruik van de SAMPLE-functie op de meest triviale manier. Er wordt simpelweg één variabele bemonsterd met behulp van de functie (Codevoorbeeld A.6).

Uit de testen bleek dat de SAMPLE-query verschillende nodes selecteert uit de graaf. De selectie gebeurt echter niet volledig willekeurig, waardoor bepaalde nodes vaker worden geselecteerd dan andere.

[22] 'Sample is a set function which returns an arbitrary value from the multiset passed to it.' [33]

```
SELECT SAMPLE(?ss AS ?s)
WHERE { ?ss ?p ?o }
GROUP BY ?ss
```

Codevoorbeeld A.6: Voorbeeld van de SampleVar-methode.

SampleSubquery

Om de selectie terug willekeurig te laten gebeuren, worden methoden voorgesteld die de resultaten van een *subquery*[23] bemonsteren. De subqueries die bemonsterd worden zijn deze besproken in paragraaf A.1.2. De complete queries worden gegeven in Codevoorbeeld A.7 en A.8.

```
SELECT (SAMPLE(?ss AS ?s)
WHERE {
    {
        SELECT ?ss
        WHERE { ?ss ?p ?o }
        ORDER BY RAND()
    }
}
```

Codevoorbeeld A.7: Eerste voorbeeld van de SampleSubquery-methode.

```
SELECT (SAMPLE(?ss AS ?s)
WHERE {
    {
        SELECT ?ss
        WHERE {
            ?ss ?p ?o
            BIND(RAND() AS ?randVar)
        }
        ORDER BY ?randVar
    }
}
```

Codevoorbeeld A.8: Tweede voorbeeld van de SampleSubquery-methode.

[23] 'Subqueries are a way to embed SPARQL queries within other queries, normally to achieve results which cannot otherwise be achieved, such as limiting the number of results from some sub-expression within the query.' [33]

A.2 Testen

Alle hierboven beschreven queries werden getest op willekeurigheid van de oplossing (met een beperkte grootte van de graaf)(paragraaf A.2.1) en de snelheid van de uitvoering ervan (paragraaf A.2.2). Beide testen werden twee maal uitgevoerd: een keer waarbij slechts één bron wordt opgevraagd en een keer waarbij tien bronnen worden opgevraagd (dit laatste geldt enkele voor queries met een LIMIT-clausule).

A.2.1 Willekeurigheidstesten

Het doel van deze test is het bepalen van de queries die een zo willekeurig mogelijke oplossingenverzameling opleveren. Dit houdt in dat elke node uit de bevraagde graaf met een even grote waarschijnlijkheid wordt opgenomen in het resultaat van de query. Om dit te testen werden alle queries 100.000 keer uitgevoerd op eenzelfde RDF-graaf met 100 nodes. Op basis van de resultaten van de verschillende uitvoeringen kon achterhaald worden hoe vaak een bepaalde node werd geselecteerd.

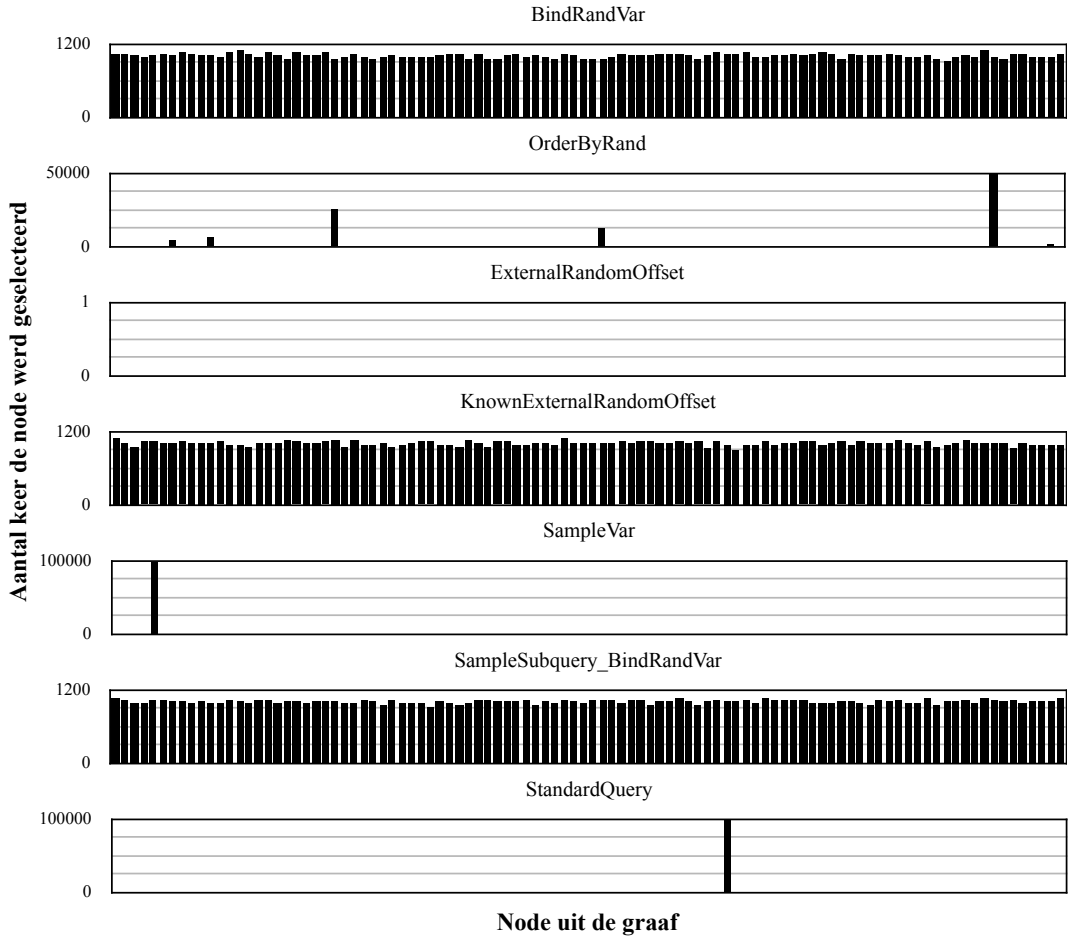
Bij de testen waarbij tien bronnen worden opgevraagd, wordt een andere telling gebruikt. Om te kunnen achterhalen of de volgorde van de oplossingenverzameling willekeurig is, wordt een verschillende score gegeven aan de resultaten uit de oplossingenverzameling. De eerste oplossing krijgt namelijk een hogere score (tien) dan de laatste oplossing in de verzameling (1).

De resultaten van de selectie van één bron zijn terug te vinden in Figuur A.1. De resultaten van de selectie van tien bronnen staan in Figuur A.2. In deze grafieken is te zien hoe vaak (y-as) een bepaalde bron (x-as) werd geselecteerd[24] over alle uitvoeringen heen van een specifieke methode. De resultaten laten zien dat slechts drie methoden geschikt zijn om willekeurige nodes te selecteren uit een graaf: `BindRandVar`, `KnownExternalRandomOffset` en `SampleSubQuery`. De `offset`-methode kan echter enkel gebruikt worden indien maar één bron wordt opgevraagd. In het andere geval hebben de eerste tien bronnen namelijk een kleinere kans om geselecteerd te worden.

A.2.2 Snelheidstesten

In deze test wordt gekeken naar de uitvoeringssnelheid van de queries. Aangezien het niet eenvoudig is om in Java een representatieve *benchmark* uit te voeren, werd beroep gedaan op een framework beschreven in [6]. Ook deze testen werden dubbel uitgevoerd met elk een verschillende waarde in de LIMIT-clausule (één en tien). De resultaten worden weergegeven in de figuren A.3 en A.4.

[24] In de testen met tien bronnen wordt de het resultaat van de aangepaste telling getoond.



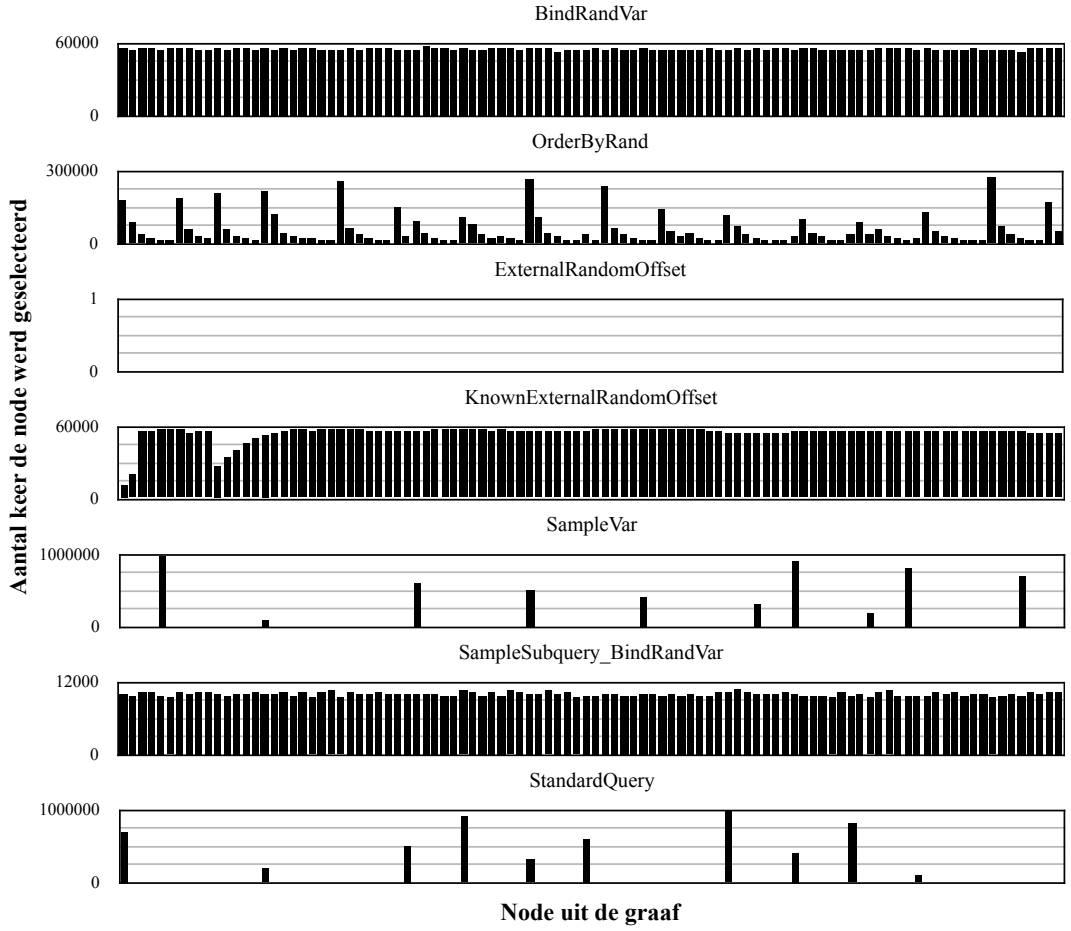
Figuur A.1: Het aantal keer een bepaalde bron werd geselecteerd op basis van de besproken methoden. Per uitvoering van een query werd er één bron geselecteerd uit een graaf van 100 nodes.

Op de grafieken is te zien dat een willekeurige selectie via SPARQL een grote *overhead* geeft ten opzichte van een gewone query (zie `StandardQuery` op de grafieken). Enerzijds komt dit door het gebruik van een subquery. Deze zorgt ervoor dat er meerdere query's moeten worden uitgevoerd. Andere methoden eisen dat alle mogelijke oplossingen worden gesorteerd. Dit eist de nodige instructies.

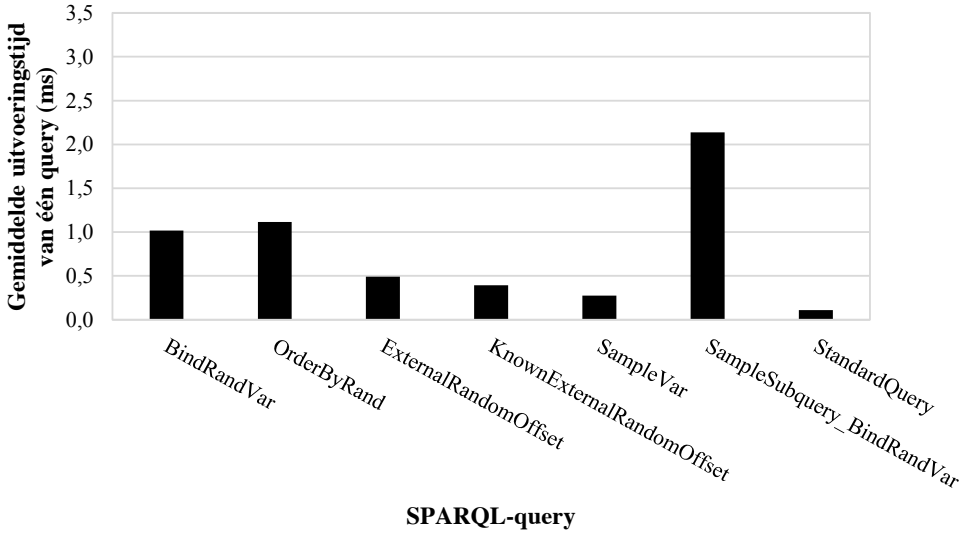
A.3 Conclusie

In deze bijlage werd getest op welke manier willekeurige nodes uit een RDF-graaf geselecteerd kunnen worden door middel van SPARQL. Op basis van de resultaten van beide testen kan geconcludeerd worden dat `KnownExternalRandomOffset` de beste methode blijkt te zijn. Het nadeel van deze methode is dat meestal niet op voorhand geweten is hoe groot de oplossingen-verzameling zal zijn (wat hier vereist is). Ook in deze studie is dit het geval, waardoor er wordt gegrepen naar de tweede beste methode: `BindRandVar`. Deze methode werkt beduidend sneller

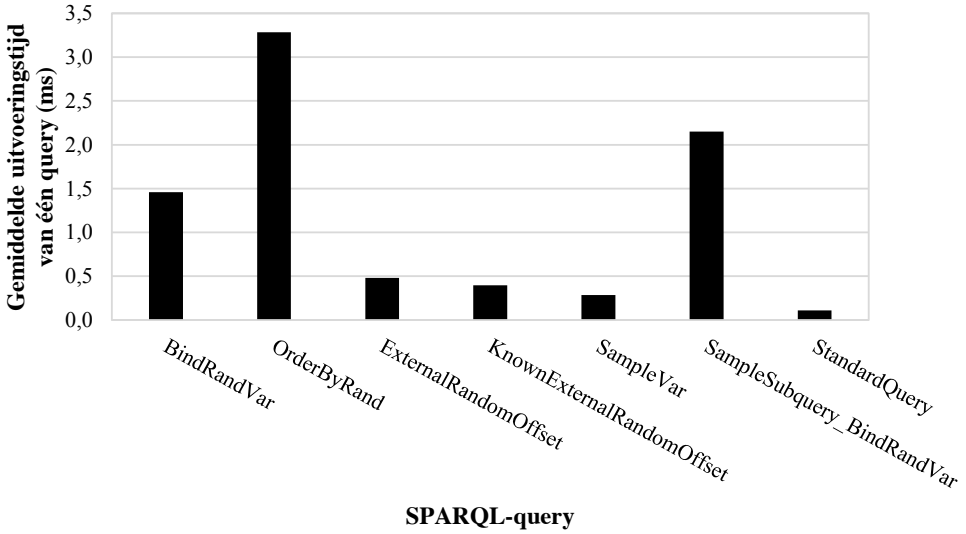
dan het laatste alternatief: `SampleSubQuery`. Ook moet opgemerkt worden dat een willekeurige selectie via SPARQL de uitvoeringstijd van een query sterk beïnvloed.



Figuur A.2: Het aantal keer (aangepaste telling) een bepaalde bron werd geselecteerd op basis van de besproken methoden. Per uitvoering van een query werden er tien bronnen geselecteerd uit een graaf van 100 nodes.



Figuur A.3: De gemiddelde snelheid waartegen de queries uit de verschillende methoden worden uitgevoerd, gemeten in milliseconden. In deze testen werd één bron gezocht.



Figuur A.4: De gemiddelde snelheid waartegen de queries uit de verschillende methoden worden uitgevoerd, gemeten in milliseconden. In deze testen werden tien bronnen gezocht.

Bijlage B

Configuratie

In deze bijlage wordt getoond hoe de simulator opgestart kan worden. Er wordt daarbij specifiek aangegeven welke configuratie vereist is om bronnen te kunnen alloceren via een ontologie gebaseerd systeem.

B.1 Configuratiebestanden

Er zijn drie configuratiebestanden nodig om een procesmodel te kunnen simuleren. De bestanden voor een enkele simulatie worden verwacht op dezelfde plaats in het bestandssysteem te staan. Daarbij dienen ze, naast de extensie, eenzelfde bestandsnaam te hebben.

- Het **procesmodel** wordt aangeleverd in BPMN 2.0 XML-notatie (extensie `.bpmn20.xml`).
- In een bestand met de extensie `.resources.xml` worden de verschillende types bronnen beschreven die aanwezig zijn in het proces. Per type bron wordt het pad opgegeven naar de bijhorende ontologie en regels. Een voorbeeld bestand wordt gegeven in Codevoorbeeld B.2.
- Om aan te gegeven welke bronnen vereist zijn voor een bepaalde taak, wordt er gebruik gemaakt van een laatste configuratiebestand (extensie `.attributes.xml`). Ook andere informatie over de verschillende elementen uit het proces wordt hier beschreven (zie paragraaf 4.1.2). Een voorbeeld configuratiebestand wordt gegeven in Codevoorbeeld B.3.

B.2 Simulatie

Om een simulatie te starten wordt gebruik gemaakt van `Maven`[25]. Hiertoe dient men een *terminal*-venster te openen op de locatie van de broncode. De commando's uit codevoorbeeld B.1 geven vervolgens aan hoe een simulatie gestart kan worden.

[25] Apache Maven, `https://maven.apache.org` (laatst geraadpleegd op 05/06/15)

Bij de start van de simulatie worden alle configuratiebestanden gezocht en verwerkt. Daarna zal de effectieve simulatie starten, waarvan het verloop gevolgd kan worden via de uitvoer op het *terminal*-venster. Na de simulatie kan het rapport[26] ervan teruggevonden worden in de `reports` map. Deze rapporten kunnen bekeken en geanalyseerd worden met een aparte GUI uit het framework (zie Codevoorbeeld B.1 voor een voorbeeld).

```
Navigeer naar de broncode van het hoofdproject
   $ cd "pad/naar/broncode/"
Compileer het hoofdproject
   $ mvn clean package install
Start een simulatie
   $ cd "pad/naar/broncode/HIPS-simulator-and-gui/"
   $ mvn exec:java -Dexec.mainClass=be.iminds.hips.Main
Bekijk & analyseer de rapporten
   $ mvn exec:java -Dexec.mainClass=be.iminds.hips.gui.LoadWindow
```

Codevoorbeeld B.1: De commando's die nodig zijn om een simulatie te starten vanuit een terminal-venster (uitgetest op een UNIX-systeem).

[26] Het rapport van een langdurige simulatie wordt soms opgedeeld in meerdere bestanden.

```
<?xml version="1.0"?>
<resources>
    <!-- Define the resources -->
    <resource>
        <id>NURSE</id>
        <name>nurse</name>

        <type>ontology</type>
        <reasoner>true</reasoner>
        <method>select</method>   <!-- this is the default -->
        <ontology>path/to/ontology.owl</ontology>
        <allocation-rule>path/to/allocation.rif</allocation-rule>
        <deallocation-rule>path/to/deallocation.rif</deallocation-rule>
    </resource>
    <resource>
        <id>DOCTOR</id>
        ...
    </resource>

    ...

    <!-- Map resources to groups -->
    <groups>
        <group>
            <id>MEDICAL-STAFF</id>
            <elements>
                <id>NURSE</id>
                <id>DOCTOR</id>
            </elements>
        </group>

        ...

    </groups>
</resources>
```

Codevoorbeeld B.2: Een voorbeeld van het configuratiebestand dat de bronnen beschrijft.

```
<?xml version="1.0"?>
<attributes>
    <element id="start">
        <!-- Specify how fast new process instances are created -->
        <rescheduleDistribution type="poisson">
            <mean>100</mean>
        </rescheduleDistribution>
    </element>

    <element id="operation">
        <!-- Task duration -->
        <distribution type="poisson">
            <mean>10</mean>
        </distribution>

        <!-- Required resources -->
        <resources>
            <resource>
                <groupId>MEDICAL-STAFF</groupId>
                <amount>3</amount>
            </resource>
        </resources>
    </element>

    ...

</attributes>
```

Codevoorbeeld B.3: Een voorbeeld van het configuratiebestand dat de attributen beschrijft van de verschillende BPMN-elementen uit het procesmodel.

Bibliografie

[1] What the heck is an ontology? URL http://www.catalysoft.com/articles/WhatIsAnOntology.html.

[2] Jena inference support. URL http://jena.apache.org/documentation/inference/index.html.

[3] Rif faq, februari 2013. URL http://www.w3.org/2005/rules/wiki/RIF_FAQ.

[4] F. De Backere, F. Ongenae, F. Van den Abeele, J. Nelis, P. Bonte, E. Clement, M. Philpott, J. Hoebeke, S. Verstichel, A. Ackaert, and F. De Turck. Towards a social and context-aware multi-sensor fall detection and risk assessment platform. *Computers in Biology and Medicine*, 2014. ISSN 0010-4825. doi: 10.1016/j.compbiomed.2014.12.002.

[5] Howard Beck, Kelly Morgan, Yunchul Jung, Sabine Grunwald, Ho young Kwon, and Jin Wu. Ontology-based simulation in agricultural systems modeling. *Agricultural Systems*, 103 (7):463 – 477, 2010. ISSN 0308-521X. doi: 10.1016/j.agsy.2010.04.004.

[6] Brent Boyer. Robust java benchmarking. URL http://www.ibm.com/developerworks/java/library/j-benchmark1.

[7] Gregory Casier, Jan Van Ooteghem, Koen Casier, and Sofie Verbrugge. Application of a discrete event simulator for healthcare processes. In *BMSD (accepted paper)*, Milan, Italy, July 6-8 2015.

[8] Femke De Backere et al. The ocareclouds project: toward organizing care through trusted cloud services. *Informatics for Health and Social Care*, pages 1–19, oktober 2014. doi: 10.3109/17538157.2014.965306. PMID: 25325572.

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN 0201633612.

[10] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5–6):907 – 928, 1995. ISSN 1071-5819. doi: 10.1006/ijhc.1995.1081.

[11] Marco Marano, Philipp Obermeier, and Axel Polleres. Processing rif and owl2rl within dlvhex. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Web Reasoning and Rule Systems*, volume 6333 of *Lecture Notes in Computer Science*, pages 244–250. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-15917-6. doi: 10.1007/978-3-642-15918-3_24.

[12] Adrian Marte. Rif4j - a reasoning engine for rif-bld. Master's thesis, Semantic Technologies Institute (STI), Leopold–Franzens–Universität, Innsbruck, maart 2011.

[13] Norm Matloff. Introduction to discrete-event simulation and the simpy language, februari 2008. URL `http://heather.cs.ucdavis.edu/~matloff/156/PLN/DESimIntro.pdf`.

[14] Femke Ongenae, Dries Myny, Tom Dhaene, Tom Defloor, Dirk Van Goubergen, Piet Verhoeve, Johan Decruyenaere, and Filip De Turck. An ontology-based nurse call management system (oncs) with probabilistic priority assessment. *BMC Health Services Research*, 11(1): 26, 2011. doi: 10.1186/1472-6963-11-26.

[15] Femke Ongenae, Maxim Claeys, Wannes Kerckhove, Thomas Dupont, Piet Verhoeve, and Filip De Turck. A self-learning nurse call system. *Computers in Biology and Medicine*, 44 (0):110 – 123, 2014. ISSN 0010-4825. doi: 10.1016/j.compbiomed.2013.10.014.

[16] Axel Polleres. How (well) do datalog, sparql and rif interplay? In Pablo Barceló and Reinhard Pichler, editors, *Datalog in Academia and Industry*, volume 7494 of *Lecture Notes in Computer Science*, pages 27–30. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-32924-1. doi: 10.1007/978-3-642-32925-8_4.

[17] Laurens Van Poucke and Pieter Demyttenaere. Techno-economische evaluatie van ehealth-diensten met behulp van ontologiegebaseerde operationele procesoptimalisatie. Master's thesis, UGent, 2014.

[18] Francisco Ruiz, Felix Garcia, Luis Calahorra, Cesar Llorente, Luis Goncalves, Christel Daniel, and Bernd Blobel. Business process modeling in healthcare. *Stud Health Technol Inform*, 179:75–87, 2012. ISSN 0926-9630 (Print); 0926-9630 (Linking).

[19] Oumy Seye, Catherine Faron-Zucker, Olivier Corby, and Corentin Follenfant. Bridging the gap between rif and sparql: Implementation of a rif dialect with a sparql rule engine. In *Artificial Intelligence meets the Web of Data workshop, ECAI*, volume 20, pages 19 – 24, montpellier, France, augustus 2012. URL `https://hal.inria.fr/hal-00724291`.

[20] Nova Spivack. Making sense of the semantic web. The Next Web, april 2008. URL `http://vimeo.com/1062481?pg=embed&sec=1062481`.

[21] C. Turnitsa, J.J. Padilla, and A. Tolk. Ontology for modeling and simulation. In *Simulation Conference (WSC), Proceedings of the 2010 Winter*, pages 643–651, Dec 2010. doi: 10.1109/WSC.2010.5679124.

[22] Stijn Verstichel, Femke Ongenae, Leanneke Loeve, Frederik Vermeulen, Pieter Dings, Bart Dhoedt, Tom Dhaene, and Filip De Turck. Efficient data integration in the railway domain through an ontology-based methodology. *Transportation Research Part C: Emerging Technologies*, 19(4):617 – 643, 2011. ISSN 0968-090X. doi: 10.1016/j.trc.2010.10.003.

[23] *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C, mei 2004. URL `http://www.w3.org/Submission/SWRL/`.

[24] *OWL 2 Web Ontology Language - Primer*. W3C, oktober 2009. URL `http://www.w3.org/TR/owl2-primer/`.

[25] *OWL 2 Web Ontology Language - Document Overview*. W3C, second edition edition, december 2012. URL `http://www.w3.org/TR/owl2-overview/`.

[26] *OWL 2 Web Ontology Language - Quick Reference Guide*. W3C, second edition edition, december 2012. URL `http://www.w3.org/TR/2012/REC-owl2-quick-reference-20121211/`.

[27] *OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax*. W3C, second edition edition, december 2012. URL `http://www.w3.org/TR/owl2-syntax/`.

[28] *RIF Core Dialect*. W3C, second edition edition, februari 2013. URL `http://www.w3.org/TR/rif-core/`.

[29] *RIF Framework for Logic Dialects*. W3C, second edition edition, februari 2013. URL `http://www.w3.org/TR/2013/REC-rif-fld-20130205/`.

[30] *RIF Overview*. W3C, second edition edition, februari 2013. URL `http://www.w3.org/TR/rif-overview/`.

[31] *RIF - Primer*. W3C, second edition edition, februari 2013. URL `http://www.w3.org/TR/rif-primer/`.

[32] *SPARQL 1.1 Overview*. W3C, maart 2013. URL `http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/`.

[33] *SPARQL 1.1 Query Language*. W3C, maart 2013. URL `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

[34] *RDF 1.1 Concepts and Abstract Syntax*. W3C, februari 2014. URL http://www.w3.org/TR/rdf11-concepts/.

[35] *RDF 1.1 Primer*. W3C, juni 2014. URL http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/.