

# Improving the design process: The implications of automated verification of client specific requirements using semantic web standards and rule checking techniques

By L.T. (Luuk) Moonen BSc

Master Thesis  
Construction Management and Engineering  
Eindhoven University of Technology

November 2016

## Supervisors:

Prof. Dr. Ir. B. (Bauke) de Vries – TU/e

Dr. Dipl.-Ing. J. (Jakob) Beetz – TU/e

PhD. Student C. (Chi) Zhang – TU/e

Ir. W.F. (Wilfred) van Woudenberg – BAM Advies & Engineering

C. (Carin) Barreveld – BAM Advies & Engineering



## Colophon

### *General*

Report : The implications of automated verification of client specific requirements using semantic web standards and rule checking techniques  
Date : 2 November 2016  
Date Presentation : 24 November 2016  
Place : Eindhoven, The Netherlands

### *Student*

Author : Luuk Thomas Moonen  
Student Number : 0716516  
E-mail : [l.t.moonen@student.tue.nl](mailto:l.t.moonen@student.tue.nl)  
: [ltmoonen@gmail.com](mailto:ltmoonen@gmail.com)  
University : Eindhoven University of Technology  
Master Track : Construction Management and Engineering  
Chair : Information Systems in the Built Environment

### *Graduation Company*

Company : BAM Advies & Engineering  
Division : BIM center

### *Supervisors*

#### *Chairman*

Prof. Dr. Ir. B. (Bauke) de Vries – TU/e

#### *First Supervisor*

Dr. Dipl.-Ing. J. (Jakob) Beetz – TU/e

#### *Second Supervisor*

PhD. Student C. (Chi) Zhang – TU/e

#### *Company Supervisors*

Ir. W.F. (Wilfred) van Woudenberg – Senior BIM advisor  
C. (Carin) Barreveld – Process manager Systems Engineering  
BAM Advies & Engineering

Construction Management and Engineering  
Eindhoven University of Technology  
De Wielen  
Postbus 513, 5600 MA Eindhoven  
Tel. +31(0)40 247 5051



BAM Advies & Engineering  
Koninklijke BAM Groep  
Runnenburg 12  
3981 AZ Bunnik  
Tel. +31(0)30 659 8933







## Preface

This thesis represents the end of my master Construction Management and Engineering at Eindhoven University of Technology. A master which I enjoyed immensely to further develop my skills in the field of process and project management in the construction sector. This research is carried out in collaboration with BAM Advies & Engineering. This collaboration has resulted in an interesting research that makes use of the knowledge of both the academic world and the current practice in the construction industry. This combination has contributed tremendously to my personal development and has driven me to continue to work and develop in this field of expertise. To execute this research and to gain the knowledge in this field of expertise I have received help and guidance from a couple of people. Without this help and guidance it wouldn't have been possible to execute the research. This section is dedicated to those that helped me during my graduation project.

Firstly I want to thank Jakob Beetz for his guidance in this process. From the first until the last meeting you have challenged me to think about the further possibilities and opportunities with the available techniques. This has resulted in interesting discussions about the applicability towards the current sector and future development in the construction industry. Thank you for the continuous support and helping me bring this research to the next level.

Secondly I want to thank Chi Zhang for the help you have given me. As I still remain a beginner in programming, I must thank you greatly for helping me develop the prototype of the tool. I learned very much from our discussions about the use of the checker and I highly appreciate the time you have invested in helping me.

Wilfred van Woudenberg & Carin Barreveld, thank you greatly for giving me the opportunity to conduct this research at BAM Advies & Engineering. You both have helped me greatly to improve my research and to ensure that the research was applicable to the current processes. The extensive advice you both gave me has enabled me to understand the current practice and what developments are happening. The meetings were always challenging my knowledge and after every meeting I knew how to continue and improve my work. I look forward continuing working together. Aside from my graduation supervisors I also want to thank the people from BAM whom I interviewed. I greatly appreciate the time you all have taken for the interviews which enabled my research.

Eline, Joost, Carin, my special thanks to you for the continuous support during the total process. The lack of clarity when you start formulating your research has been hard sometimes and your support has given me the strength to reach my goal. Last but not least, HDP. Thanks to all of you who made my life as a student unforgettable.

I hope that every reader will enjoy reading (and learning from) this thesis report.



Luuk Thomas Moonen



## Contents

Colophon.....	2
Preface .....	4
Contents .....	6
Summary.....	8
1. Introduction.....	9
1.1 Motivation.....	9
1.2 Problem definition.....	10
1.3 Research scope.....	10
1.4 Research Questions .....	11
1.5 Research Design .....	11
1.6 Expected results .....	12
2. Glossary.....	14
List of figures .....	14
List of tables .....	14
List of Listings.....	15
List of abbreviations .....	15
3. Literature review.....	16
3.1 Motivation.....	16
3.2 Design Process .....	16
3.2.1 Information exchange in the Design process.....	18
3.2.2 Systems engineering.....	19
3.2.3 Requirements.....	24
3.2.4 Verification .....	25
3.3 Rule Checking.....	27
3.3.1 Types of rule checkers.....	27
3.3.2 Rule checking process.....	29
3.3.3 Rule checking platforms .....	31
3.3.4 Difficulties in automated rule checking .....	31
4. Qualitative research.....	34
4.1 Motivation.....	34
4.2 Interview setup.....	34
Setup per subject.....	35
4.3 Interview outcome .....	36
4.3.1 Design Process.....	37
4.3.2 Verification .....	40
4.3.3 Automation of Verification.....	42
4.4 Conclusions on research questions.....	43
5. Model.....	45

5.1	Introduction.....	45
5.1.1	Research Problem .....	45
5.1.2	Importance .....	45
5.1.3	Previous Work .....	46
5.1.4	Primary hypothesis & objective.....	48
5.2	Method.....	48
5.2.1	Tasks for developing the requirements checker .....	50
5.3	Results.....	51
5.3.1	Rule interpretation.....	52
5.3.2	Building model preparation.....	61
5.3.3	Rule execution.....	63
5.3.4	Rule Reporting.....	67
5.4	Use case validation .....	69
5.5	Discussion .....	70
6.	Conclusion.....	73
7.	Recommendations .....	76
7.1	Company Recommendations.....	76
7.2	Future development & research.....	77
8.	References .....	79
	Annex A – Interview Questions.....	84
	Annex B – Verification Process .....	85
	Annex C – Requirements Hierarchy.....	86
	Annex D – Space Classification .....	87
	Annex E – Selected Requirements.....	88
	Annex F – SPIN Constraint Templates .....	90
	Annex G – BAM OTL.....	103
	Annex H – SPIN Inference Rules.....	108
	Annex I – SPIN Constraint template queries.....	113
	Annex J – Tool Flowchart .....	117
	Annex K – Script Tool .....	118
	Backend script.....	118
	Front end script.....	120
	Annex L – Application windows.....	133
	Annex M – Flowchart Template selection.....	136

## Summary

## 1. Introduction

The complexity of projects in the Architecture, Engineering and Construction (AEC) sector has increased over the years due to changes in responsibilities and more complex requirements from the client (Lenferink, Tillema, & Arts, 2013). A more integrated way of working is there for emerging in the industry to deal with this complexity. This must lead to a reduction in failure costs and higher quality designs (Abanda, Zhou, Tah, & Cheung, 2013). To achieve this higher quality, the performance towards requirements of a design must be monitored closely. To assure that a design is complying towards standards and requirements, the importance of information management within a building project is growing.

The use of Building Information Models (BIM) has made it possible to deal with this information management. The AEC sector has acknowledged the various benefits of BIM to create designs of higher quality with less errors (Chuck; Eastman, Teicholz, Sacks, & Liston, 2011). To structurally manage information of requirements, the AEC sector is implementing Systems Engineering (SE) in their design processes (Pels, Beek, & Otter, 2013). The information in requirements and in building models interact greatly in the verification process. Here the design is checked on the compliance with requirements. This process is a time consuming and error prone process which can be complex. Therefore the research of automated rule checking of 3D models has come forward to reduce these errors and achieve higher quality. The process of rule checking within the AEC sector is researched upon greatly. This research investigates the possibility of using rule checking techniques for automated verification of client specific requirements.

### 1.1 Motivation

The importance of managing the compliance of requirements has grown with the introduction of integrated contracts. These contracts have resulted in a shift of responsibility in a design process (Lenferink et al., 2013). This shift has resulted in a higher responsibility of the contractor during the whole construction project. This results in the need for increased visibility in the performance of a design. This performance is realized by firstly the design and after that the realization. Having a design that complies 100% with the requirements before a building is constructed, decreases the failure costs and improves the quality for a client and the end user. An opportunity here lies in to manage this performance with the use of building information models.

The management of the client specific requirements remains difficult as the client doesn't completely know what he wants in the start of a project. To define a working model before it is realized therefore is difficult as clients define their need iteratively by evaluating and updating their requirements for the design (Kim, Kim, Cha, & Fischer, 2015). A contractor therefore should be adaptable to this iterative character and manage the requirements closely. The management of requirements now remains a manual process which requires investigating the information about requirements and the design. This information is available within computer databases. The usability of this information depends on the data structure, the way the data is stored and how the users can interact with the data. Researching this area to see how the information should be documented for the usage in verification in the design process is there for necessary. The usage of this information for automated verification will improve efficiency and quality of the design process.

For checking data in BIM models, automated rule checkers are used. The current automated checkers of model data focus mainly on compliancy with building codes and well formedness of a building model. These checks have a static character and don't have an element of timing in the checks. The investigation of automated verification addresses this element of time in a design process. This can increase the value of a checker.

The current automated checkers also focus mainly on specific domains and elements and not on the total design process (Krijnen & van Berlo, 2016). The method for creating a rule checker that focusses on a general approach of capturing rules is there for investigated upon in this research. Lastly the method of creating rule checkers relies on expert knowledge on translating a requirement into a computer processable code (Zhang, Beetz, & Weise, 2015). This makes the openness of working with and creating the rule checkers difficult as high levels of expertise on programming, using data structures and the construction domain is needed. The person who will check a model will always be relying on the interpretation of the programmer and won't be

able to create the checks himself. Therefore a great opportunity lies in creating an automated rule checker that allows non-experts to create the rules themselves.

The usage of semantic web standard for this implementation makes this possible and also gives the possibility to create a knowledge system. An automated rule checker based upon semantic web standards makes reusability possible. This reusability creates the opportunity to create a knowledge system which can be expanded.

## 1.2 Problem definition

To ensure the functionality and quality of a building, the compliancy towards client specific requirements is of major importance. Non compliancy towards these requirements still results in extensive additional costs and extended duration of a project. The quality of a design is there for important to manage properly to reduce design errors. This required quality of a design is checked in the verification process. Managing this quality of a design remains problematic due to the complexity of the verification process, as this is a manual and error prone process.

The main problem in these errors originates from the complexity of managing the information in requirements and in the building model. This comes from the fact that there is a vast amount of requirements and standards which can apply to a building project. The applicability of these requirements varies among the different projects and stakeholders. Furthermore these requirements also are also varying greatly in how they are stated in the requirements databases. This results in complexity in managing the compliancy towards requirements in the design process.

Here lies an opportunity to improve the design process by using rule checking techniques to overcome this complexity. These techniques can be beneficial for the design process by defining this information once and translating this into reusable rules. When the knowledge for these checks is laid down in a smart and open system, the amount of errors can be reduced as the knowledge is captured. The usage of rule checking for client specific requirements relies on the applicability to the current design process, possibility of using a rule checker by the layman and aligning the information in requirements and objects. Therefore the design process must be elaborated on and the preconditions for automation must be defined clearly. Also the interpretation of the rules which are created must be defined clearly to realize automated verification.

## 1.3 Research scope

To prove the usability of rule checking for the purpose of automated verification of client specific requirements, defining the scope of this research is needed. This research will start evaluating the problem in a broad way to eventually derive a prototype which evaluates the concept which is defined along the steps in the research. To ensure that verification of client specific requirements can be automated, the design process must be evaluated to ensure that all factors and steps in the verification process are taken into account. The information structures and interactions will there for also needed to be evaluated. In this way pre conditions of automation of the existing design process can be defined. For creating a prototype for automated verification of client specific requirements, certain objects and requirements must be defined and chosen. This must be done to make this research specific and achievable.

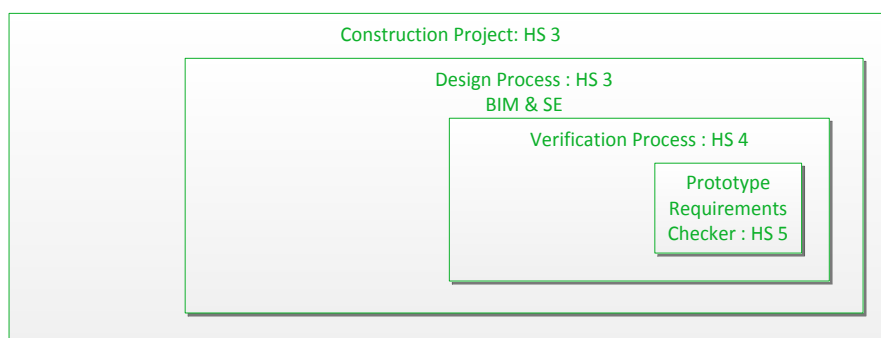


Figure 1: Research Scope

This research focuses on the requirements of internal walls. This object is chosen for evaluation as there is a vast amount of varying requirements which are applicable to walls and walls are occurring often in a building. The relation with spaces of these walls is essential to investigate the total working of a system. This gives a good opportunity to evaluate the possibilities of using rule checking for client specific requirements. This prototype is evaluated with a use case to define the usability and benefits and drawbacks of using rule checking techniques for client specific requirements.

## 1.4 Research Questions

From the problem definition and the scope the research questions can be drawn up. The main research question is:

**How can verification of client specific requirements be automated and improve the design process?**

This main research question is divided in seven sub questions. These sub questions elaborate on the two parts of the research. These parts are the design process and automated rule checking. The design process is evaluated with a view towards Systems Engineering, verification of requirements and the information exchange which is happening in the process. Automated rule checking is reviewed to evaluate the possibilities to create an automated rule checker for client specific requirements. These two subjects lead to the following sub questions;

### Design Process

- 1 What steps can be found in the design process and how do they align with the Systems engineering process?**
- 2 What type of requirements can be found in building projects and have the biggest impact in the case of non-conformity?**
- 3 What is the common practice in verification in the design process?**
- 4 What does automation of verification implicate for the design process?**

### Automated Rule Checking

- 5 How can a requirements checker be created for automated verification?**
- 6 How can a requirements checker be made beneficial for the design process?**
- 7 What are the conditions to make this requirements checker reusable?**

## 1.5 Research Design

This research consists of four parts which come forward from the research questions. The first four research questions are researched upon in two ways, firstly with the use of existing literature and secondly with an evaluation of the current practice with interviewing experts. Therefore firstly a theoretical research will be done. A literature review will be done on the two main themes; the design process and automated rule checking. In the literature review of the design process aside from a basic overview, a focus will be given on the use of Systems Engineering and the verification process and on requirements. After the theoretical research, a qualitative research upon the current practice will be done. Interviews will be held to evaluate the design process and assess where errors are occurring from to evaluate how the verification process can be automated. The conclusions of the interviews and the evaluation of the literature will be translated into the scope for the requirements checker.

To evaluate the possibilities for a requirements checker, a selection in requirements will need to be made to define the scope of this research. This will need to be done according to the various types of requirements. These types are identified with an analysis of requirements databases in existing projects. The applicability of this checker will focus on internal walls and adjacent spaces to evaluate the concept of rule checking for client specific requirements. The relation between walls and spaces has received little attention in research.



As requirements are most often space related this relation must be investigated properly to define a suitable model checker. In this way a clear evaluation can be made amongst the various requirement types instead of choosing one requirement type.

These steps which are described have been stated in the research model which can be found in Figure 2 on the next page.

## 1.6 Expected results

The expected results of this research come in threefold. Firstly a literature review upon the design process and automated rule checking will be given. Here a clear evaluation of the current processes will be given to outline the scope of the to be created tool and what the applicability issues and opportunities will be. This literature review can be found in chapter 3.

In the second part, interviews will be held to evaluate where problems with verification are occurring from in the design process. The evaluation of these interviews will be put in an interview report outside of this thesis report. The outcome will be summarized in chapter 4.

After the literature and the current practice are evaluated, research question one to four will be answered (section 4.4). Hereafter, the outline for automated rule checking for client specific requirements is given. In the fifth chapter this outline and scope will be used for the creation of the requirements checker. The creation of this checker will be done with the use of the semantic web standard to create a system which contains the knowledge which is required for the checks and is open and reusable. This reusability is realized by creating requirement templates and the usage of natural language concepts translated in an object type library. This system relies upon the definition of the elements which exist in a model and upon the knowledge which is built up in the requirement checks. This will eventually result in a prototype of an automated requirement checker for client specific requirements which is based upon the semantic web standard and open standards. In this prototype amongst a variety of requirement types certain examples for wall and space requirements are used to test the concept. This results in the evaluation of using automated rule checking for client specific requirements. This can then be evaluated if automated verification is beneficial for the design process. This development can be found in chapter 5.

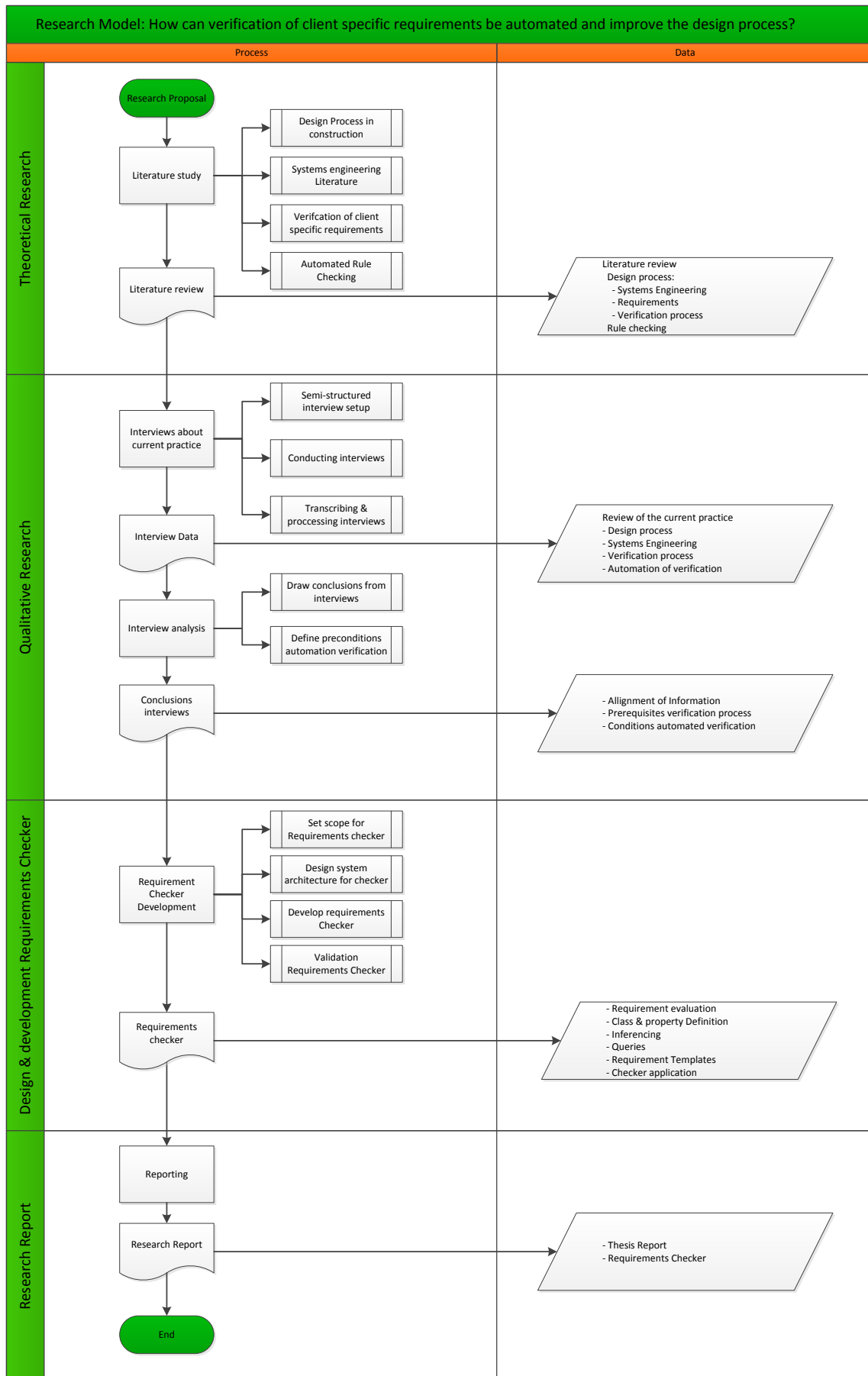


Figure 2: Research Model

## 2. Glossary

### List of figures

Figure 1: Research Scope.....	10
Figure 2: Research Model.....	13
Figure 3: Project Life Cycle of a construction project with project phases .....	16
Figure 4: Iterative Character during the design Phases.....	17
Figure 5: MacLeamy Curve .....	17
Figure 6: Interaction Requirements and Design solutions .....	18
Figure 7: Systems Engineering Process.....	21
Figure 8: Systems Engineering process extended.....	22
Figure 9: Interaction in design process.....	23
Figure 10: Hierarchy of client needs .....	24
Figure 11: Essence of the verification Process .....	26
Figure 12: Overview of model checking concepts .....	27
Figure 13: Process of rule checking.....	29
Figure 14: Role interviewees.....	36
Figure 15: Allocation of Object and Requirements.....	38
Figure 16: Deviation interpretation of requirements .....	39
Figure 17: Improving requirements analysis .....	40
Figure 18: Example of triple graph with URI.....	46
Figure 19: Architecture for Model Checking on the Semantic Web.....	49
Figure 20: Structure of using constraint templates for requirements checking .....	51
Figure 21: Interaction between information in requirements and objects .....	52
Figure 22: Requirements Type Classification .....	53
Figure 23: Example of iteration to create an equation with similar elements.....	54
Figure 24: Overview of Characteristics within project 3 .....	55
Figure 25: Space Hierarchy without subclasses .....	58
Figure 26: BAM Ontology in Topbraid Composer.....	62
Figure 27: Magic Property Example as displayed in Topbraid Composer.....	63
Figure 28: Overview Requirements Checker application .....	65
Figure 29: Overview of windows of the requirements checker.....	67
Figure 30: Visualization of checking report.....	68
Figure 31: Project 4 model.....	69

### List of tables

Table 1: Type of requirements.....	24
Table 2: Rule Checking Categories.....	28
Table 3: Examples rule classes.....	29
Table 4: Overview of possible difficulties in rule checking development.....	33
Table 5: Structure of a SPARQL Query .....	47
Table 6: Description analyzed projects.....	53
Table 7: Data Analysis Questions.....	54
Table 8: Data analysis come .....	56
Table 9: Constraint templates.....	59
Table 10: Selected requirements for prototype.....	60
Table 11: Property definition.....	62

## List of Listings

Listing 1: Rule Classes.....	28
Listing 2: Example Query.....	47
Listing 3: Example Inference.....	49
Listing 4: Constraint Template Space-object relation: value.....	60
Listing 5: Constraint template example.....	60
Listing 6: Inference rule for bam:FunctionalSpace.....	61
Listing 7: inference Rule for bam:InternalWall.....	61
Listing 8: Workflow for checking process.....	64
Listing 9: Filled in Constraint template for acoustic comfort of an internal wall .....	66
Listing 10: Filled in Constraint Template for Power Socket Contianment in a office space.....	66
Listing 11: Code for accessing constraint templates.....	66
Listing 12: Partial output of checking.....	67
Listing 13: Outcome of check for acoustic Comfort.....	69

## List of abbreviations

3D	: three dimensional
AEC	: Architecture, Engineering Construction
API	: Application Programming Interface
BCF	: BIM Collaboration Format
BIM	: Building Information Model(ling)
CAD	: Computer Aided Design
FBS	: Functional Breakdown Structure
GUID	: Global Unique IDentifier
IAI	: International Alliance for Interoperability
IFC	: Industry Foundation Class
ISO	: International Organization for Standardization
LOD	: Level of development/detail
MVD	: Model View Definition0
NEN	: Normalisatie en Normen
OBS	: Object Breakdown Structure
OTL	: Object Type Library
OWL	: Web Ontology Language
RBS	: Requirement Breakdown Structure
RDF	: Resource Description Framework
RDFS	: Resource Description Framework Schema
SBS	: System Breakdown Structure
SE	: Systems Engineering
SMART	: Specific, Measurable, Attainable, Realizable and Time bounded
SPARQL	: SPARQL Protocol And RDF Query Language
SPIN	: SPARQL Inference Notation
TTL	: Turtle
URI	: Unique Resource Identifier

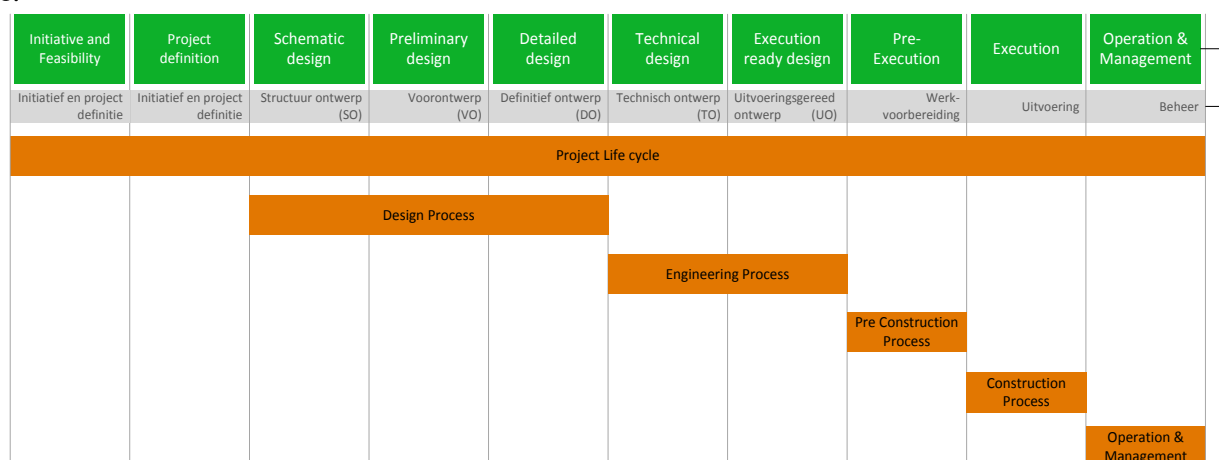
### 3. Literature review

#### 3.1 Motivation

In the literature review an overview is given on two specific topics. Firstly the research on the design process in the Architecture Engineering and Construction (AEC) sector and secondly on automated rule checking. This is needed to evaluate the possibility of using automated verification within the AEC sector for client specific requirements. The design process is evaluated to clearly define the process and environment where client specific requirements are used in. This can be defined as the phases where verification takes place and how data and information is used in building projects. Rule checking is evaluated to investigate the method and techniques which are used. This evaluation shows the possibilities for designing a rule checker which can be used for verification of client specific requirements within the construction industry.

#### 3.2 Design Process

To define the information development needed for verification, the development of a design during a project is evaluated. In a construction process in the AEC industry, the design process is defined in different phases of a project life cycle. There are various definitions of standard for design phases (BIMForum, 2015). This research focusses on the Dutch construction industry. In the Dutch construction industry the definition of phases is defined by the Dutch standardization institute in the Dutch standards (NEN) and the definition in “The New Rules” which are created by NLIngenieurs and BNA. The DNR-STB & NEN2574 define ten phases of a construction project (BNA, NLIngenieurs, & ONRI, 2009; Nederlands Normalisatie-instituut, 1993). The DNR-STB and NEN 2574 are defined for the use of traditional contracts where after a design is realized, a tender takes place. The timing of pricing and tenders vary greatly amongst the different contract forms (Chao-Duvis, Koning, & Ubink, 2013). To give a clear overview about the design process, this pricing phase is not evaluated in the overview. The overview of design phases have been combined and visualized in Figure 3.



**Figure 3: Project Life Cycle of a construction project with project phases based upon BNA et al., 2009; Eadie, Browne, Odeyinka, Mckeown, & McNiff, 2013; Nederlands Normalisatie-instituut, 1993**

The development of information in these phases can be seen as two different sources of information. Firstly there is information of requirements which is provided by the clients. Secondly there is the information which is created in the design which gives an answer to the requirements. The design is created according to the client's requirement and the environment the project is situated in. In this process, the interaction between the different sources of information can be found. During the development of a design, the earlier stages have a more conceptual and iterative character. The design phase is characterized by a top down approach where decisions are made about the key elements of a design in the earlier phases. This leads eventually to a more linear process where decisions are developed in a technical way in subsequent phases. This development of a design and the iterative character can be seen in Figure 4.

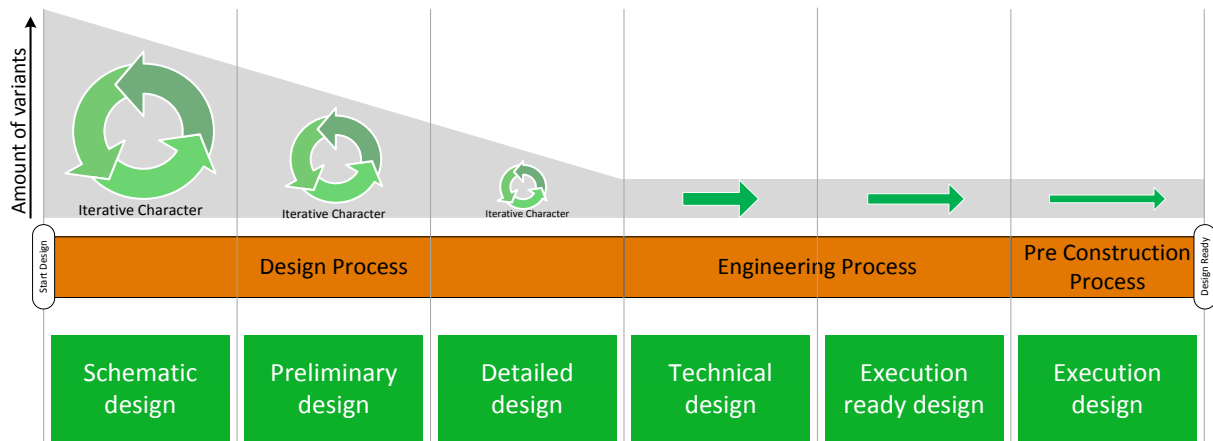


Figure 4: Iterative Character during the design Phases (own drawing)

In the earlier phases the biggest decisions are made to define what the core of a design will look like. Here the amount of variants is high, as a baseline for the design hasn't been defined yet. The amount of variants should decrease as well as the impact of decisions during the progress of a project. This is due the fact that the cost of changes will increase when changes are made in a later phase (Lu, Fung, Peng, Liang, & Rowlinson, 2014). This effect has been described in the MacLeamy Curve and can be seen in Figure 5.

The reason why the costs will rise for changes comes from the fact that the elaboration of a design will be of a higher level. The amount of rework and reconsideration brings much extra process costs in subsequent phases of the project. The amount of variants should there for be declining when progressing in the design process. The differentiation can there for be made between the design and the engineering process. In the design process, variants are assessed to choose the most suitable design solution. In the engineering process the design is clearer and meaning needs to be given to the rough elements on a higher level of detail. This implicates a change in character from more conceptual to high level of detail during the subsequent phases.

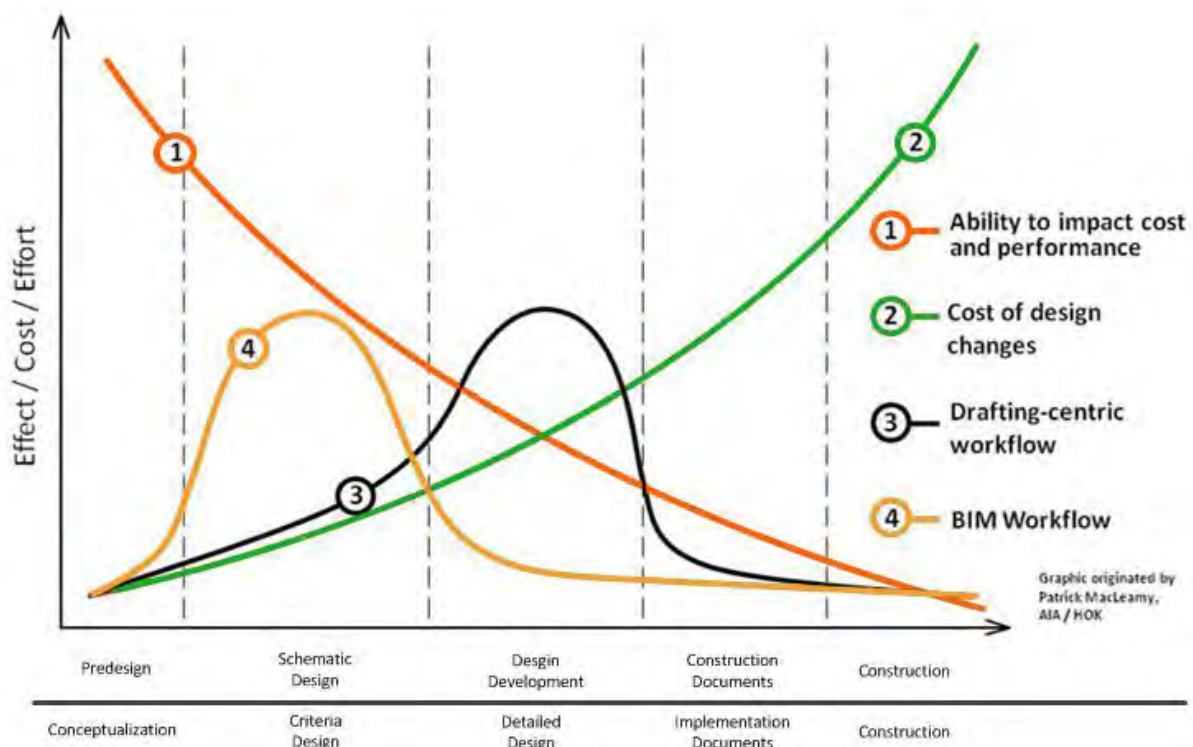


Figure 5: MacLeamy Curve (Lu, Fung, Liang, & Rowlinson, 2015)

This variation of level in detail and level of development go hand in hand with the elaboration during the design phases. Level of detail is the definition of how much detail is put into an element. The level of development is the degree of information and consideration which is put into a geometrical element in a 3D model (BIMForum, 2015). This is an important difference as the level of development has a factor of reliability in itself due to the consideration which is put into an element. The amount of variants should be reduced as the level of development will get higher as the definition of an element will be clearer. Therefore the higher level of development and the conceptual character will go hand in hand. The level of development varies among the various elements as there are dependencies and variance in importance of development (BIMForum, 2015; Solihin & Eastman, 2015).

### 3.2.1 Information exchange in the Design process

As there is an interaction between information in requirements and design solutions continuously during the various phases in a project, the interaction of data and information is crucial for the quality of a construction project (Chen & Luo, 2014). The documentation of the interaction between requirements and design solutions is gaining importance as the necessity for proving performance is growing with the introduction of integrated contracts (Kim et al., 2015; Pels et al., 2013). The interaction between requirements and design solutions is shown in Figure 6.

In this interaction it is important to make the right comparison as otherwise the information which is created will not be useful. The management of information and specifically the exchange of information in a construction project are important factors that can influence the quality of a construction project (Eastman, Teicholz, Sacks, & Liston, 2011).

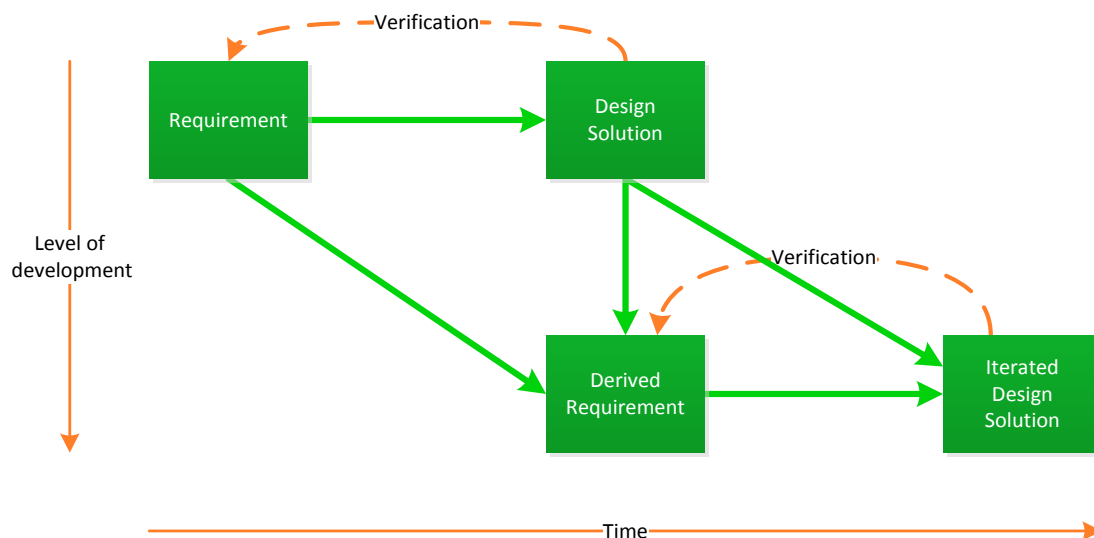


Figure 6: Interaction Requirements and Design solutions based upon (BAMinfra, 2008)

With the use of Building Information Modelling (BIM) the possibilities of managing the information in a construction project have grown. BIM is defined by Eastman as *"a modelling technology and associated set of processes to produce, communicate and analyze Building Models"*. These models are represented in a three dimensional way and contain objects which include graphical and computable data. This makes a BIM useful for further usage during the total lifecycle of a building.

In a BIM, information can be stored for a variety of purposes during a total process. The way the data is stored and the interoperability of such a model define how suitable a BIM will be. Managing the interoperability of information has been addressed as one of the major challenges with the use of BIM (Dimyadi & Amor, 2013; Eastman et al., 2011; Young Jr., Jones, & Bernstein, 2007). This is due to the various types of data used in the industry, unstandardized processes, varying classifications and great variety in stakeholders (National Institute of Building Sciences, 2011).



For the improvement of interoperability in the AEC industry, the Industry Foundation Classes (IFC) has been introduced by the BuildingSmart Alliance. The IFC standard is an international standard that provides the possibility of describing buildings throughout their lifecycle with neutral file exchange (BuildingSmart, 2013). This gives the opportunity to improve the collaboration between different domains and reducing the amount of errors coming from information exchange.

IFC is an object oriented data standard. The IFC standard is specified by the data schema. The architecture of the data schema is built up in four conceptual layers; the domain layer, the interoperability layer, the core layer and the resource layer (Liebich et al., 2013). The lowest layer is the resource layer where resource definitions are set; this layer doesn't have unique identifiers as they are defined at a higher level layer. In the core layer the entities which built up the building are defined. For example a wall or a door (IfcWall, IfcDoor) are defined here. In the interoperability layer more specialized objects and relationships can be defined. For example a relationship between a wall and a space can be defined here (IfcRelBoundary). In the Domain layer the specific concepts towards a domain are defined. Data regarding for example construction management are defined here. The scope of the example domain can cover data about values like the cost of an element. With the use of these layers, a total building is described in an open data schema.

The native software which is used by the designing companies in a design process can translate their models into IFC data to ensure interoperability. This data can then be used for various purposes within a building project. As the data can be stored in a standardized way, IFC makes it possible to use this data among varying projects in a same way. This opens up the possibility to use this data for automation of the technical processes within design phases.

### 3.2.2 Systems engineering

To evaluate the current design process, the use of systems engineering is looked upon as this is becoming a more standardized way of working in the construction industry (BNA et al., 2009). The use of systems engineering in the construction industry is introduced to manage the complexity of construction projects. Various definitions can be found in the literature about systems engineering. The international council on Systems engineering (INCOSE) gives the following definition which is adopted widely;

*"Systems engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance, training and support, test, manufacturing and disposal. Systems engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation. Systems engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the users needs"* (INCOSE, 2015).

A few keywords can be identified in these definitions which are crucial for proper implementation of systems engineering: System thinking, Interdisciplinary, Completeness and Quality. These keywords are essential for using systems engineering as they explain how the goals of SE are achieved.

#### 3.2.2.1 Characteristics of Systems Engineering

To understand the use of systems engineering within the AEC sector, the characteristics of systems engineering must be elaborated on. Identifying these steps will give an overview to see how the information in requirements interacts with a design.

##### System Thinking

A system is a whole consisting of interacting parts that work together for a stated purpose (INCOSE, 2007; ISO/IEC/IEEE 15288, 2015). A system is created by people to provide for a certain need within a defined environment (INCOSE, 2015). The parts of a system can be seen as objects, people, services or other entities. When using systems engineering in construction, most often the parts of a system are objects. The use of systems thinking is used for a better understanding of the total project or process and is the basis of systems



engineering. The total system consists of layers of subsystems. These sub systems are decomposed out of the total system and are used for dealing with complexity in a hierarchical matter (BAMinfra, 2008).

### Functional Thinking

The goal of the to be created product with the use of systems engineering is to fulfil a purpose. This purpose can be seen as a functionality of a system. A system is an answer to a certain group of functions. Therefore the need to think in functionality is important to create a proper system. Thinking in functions therefore also demands to think from large to small scale which aligns with the top down method of systems engineering (Ministry of Infrastructure and the Environment, 2005).

### Interdisciplinary

As complex systems have different aspects, experts on different knowledge areas are needed to create an integrated result which take every aspect into account (Emes, Smith, & Marjanovic-Halburd, 2012). When not all the aspects have been taken into account properly, a system will not work properly as the integration of different aspects of a system has influence on each other. As the construction sector works often segmented in different disciplines apart from each other, the total cohesion in a system might be lacking. The interdisciplinary relations are needed to evaluate to result in a better working system (Rijkswaterstaat, Bouwend Nederland, ProRail, & NLingineurs, 2009).

### Clients' needs central in the process

During the system development the need of the client is monitored continuously. The need of the client is the main directive to create a proper design. The needs are translated in requirements to verify the design. For creating a design which works best for the client, during the whole iterative process verification of the design is taking place (ProRail, 2015). In this way the need of a client stays in the focus of design.

### Transparency

To achieve a higher quality design it is needed to work transparent. This transparency makes it easier to retrieve the reasoning behind decisions which are made during the process. When these decisions aren't made transparent, interpretation can play a role to work out subsequent parts of a design. This is unadvisable as this can result in wrong assumptions which can result in design errors. An open and transparent process eventually results in less time loss and in higher quality (Werkgroep Leidraad Systems Engineering, 2007).

### Decomposition

For the top down working method of systems engineering, decomposition is needed to create an overview of a total system and to get more insight in the complex data (BAMinfra, 2008; ProRail, 2015). By using decomposition, the eventual tree structure of a system can be created and more insight on a higher level part can be given by subtracting lower level parts (Werkgroep Leidraad Systems Engineering, 2007).

### Interfaces

Interfaces in a project can be found where different systems or parts of the environment come together and influence each other through their connection (BAMinfra, 2008). In these interfaces the complexity of a project can become clear as the boundaries of different systems interact. This interaction can be found as physical forces, streams and information (ProRail, 2015). When these interactions are not investigated and monitored clearly, the interaction can affect the mutual influence on the total system. Building mistakes are often occurring due to this effect, this is why monitoring interfaces properly is a crucial part of Systems Engineering (Visser, 2011).

### Requirements

The focus on requirements during the whole lifecycle is essential for the implementation of systems engineering. From clear requirements a solution is derived which suits all the needs of the client. The exploration of these requirements is there for an essential part of the systems engineering process (Werkgroep Leidraad Systems Engineering, 2007). A further exploration on the role of requirements in Systems engineering is done in section 3.2.3.

### Verification & validation

The systems engineering handbook defines verification and validation as the following two questions; “Are we building the right thing (validation)?” & “Are we building it right? (verification)” (INCOSE, 2007). Verification and validation are essential parts of the systems engineering process which happen multiple times during the process to control the created elements of the system. Verification is needed to ensure the quality of the created product and validation is needed to ensure if the right product is created. When verification and validation is not done disciplined, goals in terms of time, cost and technical specification can be in jeopardy (Marchant, 2010). The verification process is discussed in depth in section 3.2.4.

### Life Cycle approach

Systems engineering approaches the development of a system with an approach of the total life cycle, from initiation until retirement. By evaluating the total life cycle of the product, a better understanding of the project can be achieved (ISO/IEC/IEEE 15288, 2015). In this way the needs stakeholders needs are understood better and the process stages are defined more clearly (INCOSE, 2007).

#### 3.2.2.2 Systems engineering process

The process of systems engineering is discussed greatly upon in the literature. The representation of the systems engineering process in Systems Engineering Fundamentals has been adopted greatly in the research upon Systems engineering. The core creation of the system has been illustrated in this representation and can be found in Figure 7. The main elements in this representation can be found in the interaction between requirements, functions and design elements. The relation between these three elements determine the functionality of the eventual system (US Department of Defense Systems Management College, 2001).

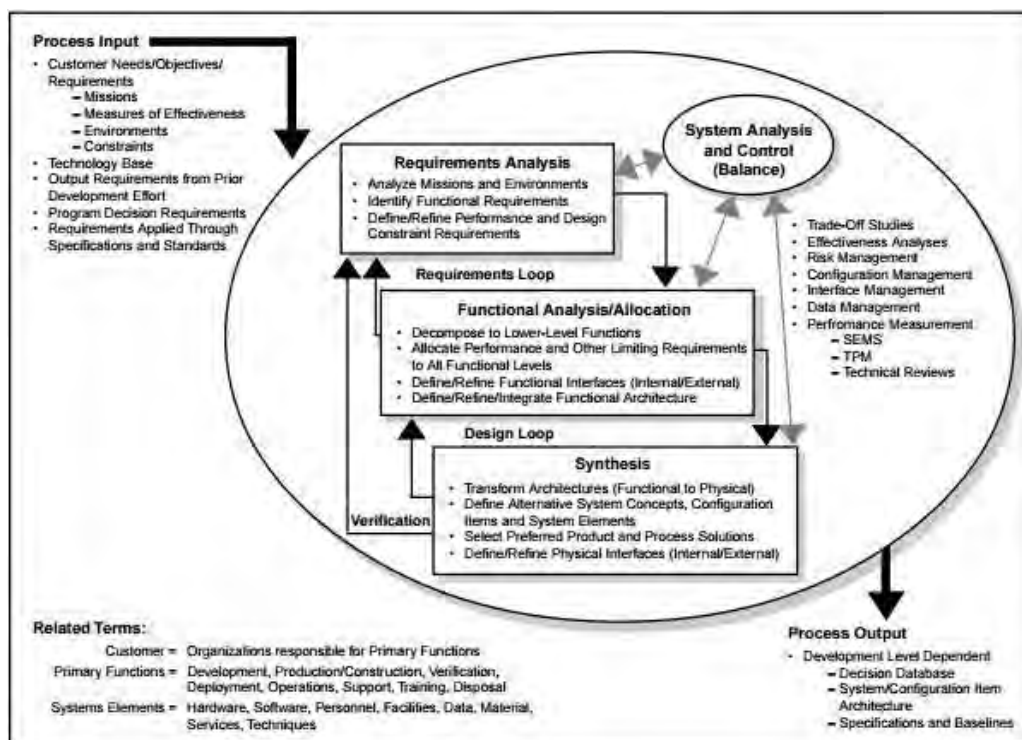


Figure 7: Systems Engineering Process (US Department of Defense Systems Management College, 2001)

To further define the process of systems engineering, the V-Model has been used greatly within the Systems Engineering literature. The V-model is used to illustrate the top down process in the design loop. Here the decomposition of the initial system is realized to give more insight in the total system (Scheithauer, Esep, & Forsberg, 2013). The V-model doesn't always show the total life cycle of a project with the use of systems engineering. Therefor a proper overview has been given by combining representations of the V-model with the total design. This total process can be seen in Figure 8 on the next page.

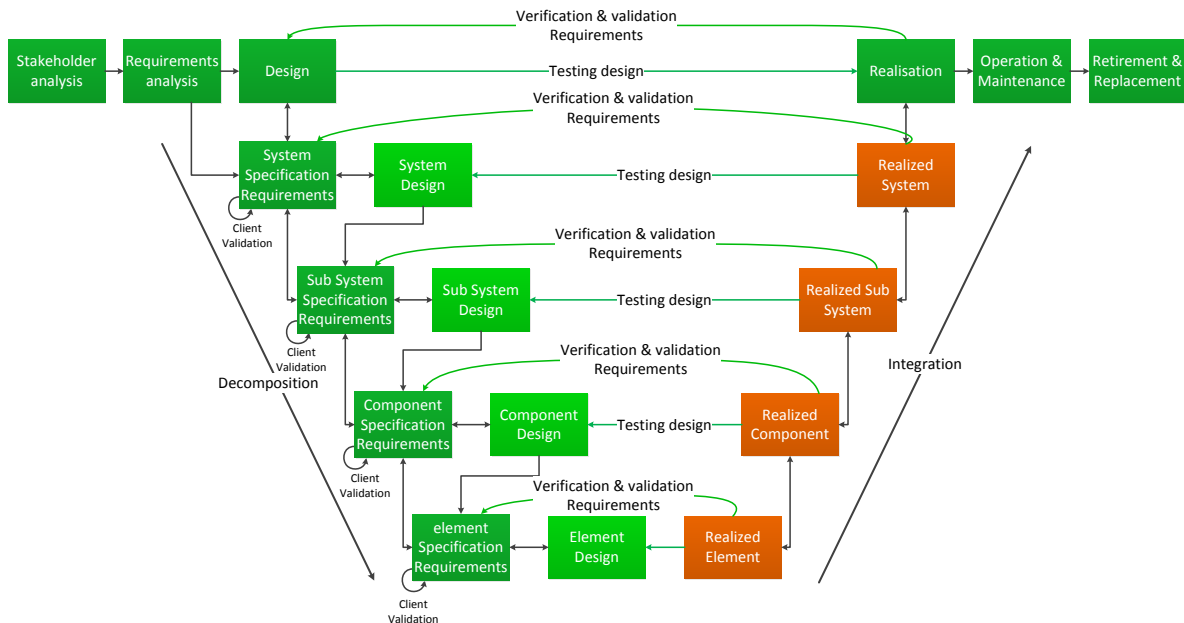


Figure 8: Systems Engineering process extended, based on (BAMinfra, 2008; INCOSE, 2015; Werkgroep Leidraad Systems Engineering, 2007)

In this process description the phases of the creation of a system can be seen throughout the whole lifecycle. These will now be elaborated upon.

### Stakeholder analysis

In the stakeholder analysis the key players in a project are identified. This is needed to identify their influence on the system and what their key necessities are in a system. To realize a proper requirements analysis the understanding of the stakeholders needs is key to identify what their goals are with a system (Glinz & Wieringa, 2007). This makes it possible to understand the functionality which stakeholders require for a proper working system. Weak related requirements to stakeholders are a main reason to project failure (Hull, Jackson, & Dick, 2006).

### Requirements analysis

The requirements analysis is defined as one of the most essential parts of the systems engineering process as the understanding of the requirements defines the design constraints (BAMinfra, 2008; Douglass, 2013; ProRail, 2015). Most often the requirements stated by the client are described ambiguously and multi interpretable (Marchant, 2010). This is due to the fact that requirements are often stated in natural text. This requires extensive analysis to understand the meaning and the goal of a requirement. In this process the understanding of the requirements results in a better understanding of the whole problem which is assessed. A proper validation with the client is needed to ensure that the interpretation is done right and to reduce discussion about the interpretation in further stages (Rijkswaterstaat, 2015).

### Design phases

In the design phases within a systems engineering approached project the system is realized. The system is realized with a top down approach. This means that throughout the design phase the system will be decomposed in to smaller elements to define the total system. In the interaction between requirements and design elements of the system it is essential to realize a system that is performing according to the required needs of a client (Schaap et al., 2008). This interaction is shown in Figure 7 and Figure 8. At every level of decomposition, an interaction between the requirements and the design is shown. Verification if the design complies with the requirements needs to be done at every level of decomposition to define if the design has the performance required by the needs of a client. If a verification isn't done at all levels, a continuation on a mistake can cause increase the impact of an error (Rijkswaterstaat, 2015). For this reason baselines are

defined after all phases within a project. This baseline needs to be verified according to the requirements to ensure the quality in a project and prevent continuing on mistakes.

The definition of a system and eventual building elements is an important interaction where information is related to each other. To ensure the right performance, the definition of the right comparison is essential for a proper working system. This interaction between requirements and the eventual performance is shown in Figure 9.

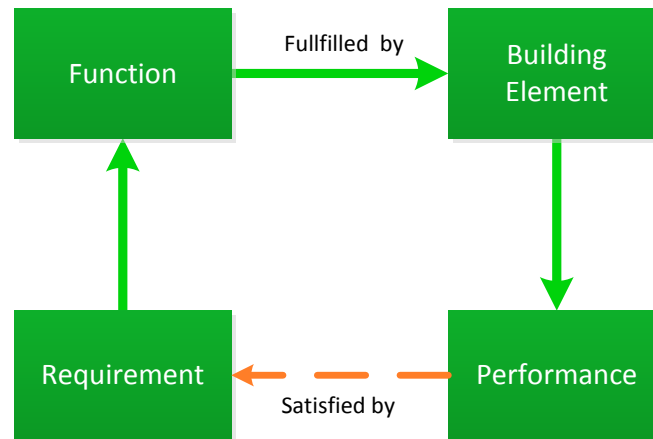


Figure 9: Interaction in design process, based upon Schaap et al., 2008

Every subsequent step in the design phase of the systems engineering process ensures the balanced development of the system and increases in level of detail (US Department of Defense Systems Management College, 2001). This balanced development and increased level of detail during the phases of the design can be seen in the V-model.

The decomposition of a system responds with this detailed development and increased detail. Decomposition in the construction sector is done in various ways depending of the type, the functionalities and the users of a building. For example an office has various functionalities (working, eating, holding meetings) but also has various users (employees, visitors, supporting personnel etc.). A system can therefore be broken down in various ways. Aside from a system breakdown structure, in the Systems Engineering Process also other breakdown structures are made. The following breakdown structures are used to decompose a project; Requirements breakdown structure, System breakdown structure, Work breakdown structure, Organizational breakdown structure and Functional breakdown structure (BAMinfra, 2008). These breakdown structures are related to each other on various levels and describe the total project. For example the steps which should be undertaken in the design process are described in the work breakdown structure. In these steps a link towards the system element are described which should be elaborated on. These objects are linked to a function, certain requirements and to responsible persons.

The design process happens iteratively as explained. When a system is described, variants are made to evaluate which design is the best solution for the requirements. This is preferably done with a trade-off matrix to evaluate all the aspects of a variant (BAMinfra, 2008). After a decision is made, the reasoning behind a choice should be documented and then requirements should be evaluated for a higher level of detail to derive the implications of iteration. This ensures that applicable requirements are taken into account in evaluating variants. The eventual steps which are taken in the design process are evolving from conceptual decisions to more detail during the subsequent phases. For example in the system design a hvac concept will be evaluated and a definition will be made about the functionality. In the subsystem design, this system will be iterated into a concept of distribution. In the component design this distribution concept will be drawn in a more specific way and in an element design the products will be chosen. Every step in this process should be verified according to the requirements to close the feedback loop as can be seen in Figure 7.

### Realization

After the design is defined, verified and validated by the customer, the construction can start. This implicates the realization of a system. This realization is done on an element level and will result in a bottom up realization of the total system. This realization needs to be tested through verification (Construction test, inspections and measurements) to ensure that a building functions according to the requirements (BAMinfra, 2008). Throughout the realization the connectivity of the various elements between the various breakdown structures ensure that the realization is done according to the realization plan and is verified according to all applicable requirements. The work breakdown structure is the most important breakdown structure in this phase as in this structure the execution of the realization is described.

### Operation and maintenance

Systems engineering can be useful in the operation and maintenance stage of a system. As the realization of the system is documented on, a description of the functionalities and the elements is available. This should make the execution of operation and maintenance more easy (BAMinfra, 2008).

#### 3.2.3 Requirements

The goal of defining a requirement is to translate the need of the stakeholders to define what the new system must be able to do (Hull et al., 2006). The definition of a requirement by the ISO standard of Systems engineering defines that a requirement is a statement that defines a need with associated constraints and conditions (ISO/IEC/IEEE 15288, 2015). A requirement originates from a certain goal, which can be translated into needs (Walraven & de Vries, 2009). These needs can then be translated into a requirement with a specific product performance. This hierarchy translates the origins of a requirement which always should be taken into account when considering requirements and the performance. This hierarchy is shown in Figure 10.

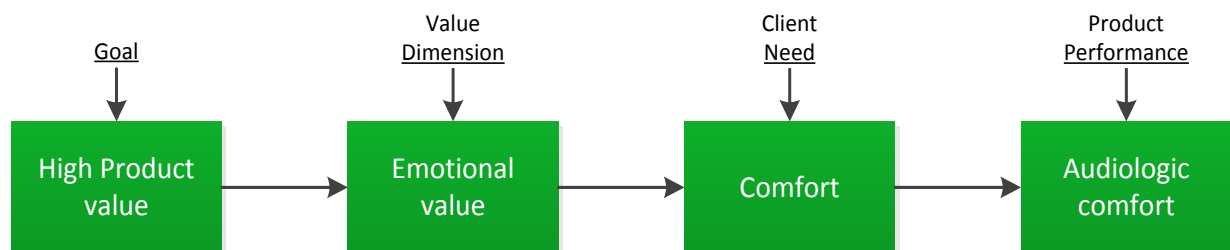


Figure 10: Hierarchy of client needs, based upon (Walraven & de Vries, 2009)

The conditions for a requirement to be used properly have been researched upon greatly. Sparrius defines that a requirement should be unambiguous, measurable traceable, verifiable and concise (Sparrius, 2014). Various types of requirements have been identified in the research on requirements engineering. Schneider & Berenbach identified three types of requirements; Physical, functional and non-functional requirements (Schneider & Berenbach, 2013). These types of requirements are illustrated in Table 1.

Table 1: Type of requirements based upon (Schneider & Berenbach, 2013)

Type of Requirement	Explanation	Example
Physical requirements	<i>Describes for example sizes, property</i>	<i>Windows must have a Rc of at least 3,0</i>
Functional requirement	<i>Describes a desired function</i>	<i>A window must have the possibility to be opened</i>
Non-Functional requirement	<i>Qualitative requirement</i>	<i>A room needs to be comfortable</i>

The ways requirements are portrayed in this table show the differences in interpretation. For example the physical requirement has a clear required value for a property which can be verified. The functional requirements must have certain ability. The last type of requirement is more difficult to see if it complies according to the requirement.



To verify these kinds of requirements in a model, the requirements need to be suitable for measuring the compliance of the model. This compliancy is defined by the performance of the design in comparison to the required performance. When requirements are not measurable and therefore not verifiable problems can emerge as interpretation can play a bigger role. To make these requirements verifiable, the requirements need to be SMART. SMART stands for: Specific, Measurable, Attainable, Realizable and Time bounded. This means that requirements need to be understandable and prevent ambiguity.

When a requirement is not quantifiable it is easily influenced by interpretation which can cause errors (Glinz, 2005). Therefore requirements in the construction sector can also be seen as numerical, relational and qualitative. Numerical requirements are easily reproduced and cause little problems as the numbers can be made clear. Furthermore this numerical kind of requirement can be translated into a mathematical equation which can be checked. This provides a possibility to automate this process. The second kind of requirement is relational and is a Boolean checking of the requirement (Schneider & Berenbach, 2013). This means that if the relation is there it is correct and if not, it is false. The last kind of requirement about quality can be disputable which makes it difficult to measure and therefore not quantifiable or possible to check without a lot of interpretation. In these kinds of requirements problems can occur. A risk in working with requirements can therefore be found in the interpretation of requirements. Communication with the client and verification of the performance is therefore an essential part of the whole project (Kiviniemi, 2005).

As the meaning of requirements can make a major impact on the design, a good understanding of these requirements is needed. Therefore it is an important part of the design process is the analysis of the requirements (BAMinfra, 2008). The validation of this interpretation with the client defines if the need of a client is satisfied.

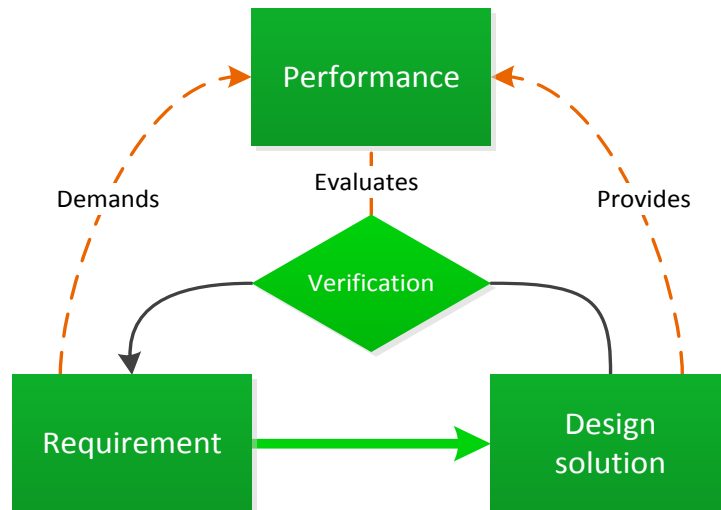
Management of requirements can often have little attention in a project while iterating the design. As requirements evolve due to iteration and decomposition, the design solution in the end result can shift away from the original goal (Kiviniemi, 2005). Kiviniemi has stated four reasons for the problems related to the management of requirements. These are the missing connection between requirements and designs, changes in personal during a project, not directly involved end-users and direct and indirect requirements. Kim et al., 2015 have defined two additional reasons why requirements management in construction can be difficult. This is firstly that the reasoning and the interpretation behind a requirement isn't documented properly and secondly that complexity in requirements arises from the many types of requirements, spaces and functions which are interrelated with each other (Kim et al., 2015). Malsane et al. have defined the following three characteristics which cause complexity; subjectivity, inconsistency in terminology and complexity in structuring and interrelationships (Malsane et al., 2015). This means that requirements are prone to the experience of the interpreter, often inconsistent in the terminology and are complex to structure and in the way they relate to other requirements and elements.

These problems relate mostly to the fact that the requirements are open to interpretation. As requirements are and text are often stated in text the measurability is low. The lack of documentation of the reasoning increases the complexity and the subjectivity. Another increasing factor is the fact that the direct link with the design is often missing. Creating a knowledge system that describes the relation between the requirements and the design should therefore be useful to overcome the difficulties existing from the missing links between the requirements and the applying elements. Also the understanding and the reasoning should be captured in this system to maintain the knowledge which is created during a project. A clear overview in the information stream from goal to product performance needs to be aligned as this process has many steps of iteration and interpretation. Due to the amount of steps made from need to product performance, this design process is remains difficult to manage and to keep close to the wishes of a client and the end-user.

### **3.2.4 Verification**

In ISO/IEC/IEEE 15288 the following definition of verification is stated; "Verification is a confirmation through the provision of objective evidence that specified requirements have been fulfilled" (ISO/IEC/IEEE 15288, 2015). This translates into the question for the design process; "is the design correct?". The verification process can be identified as a feedback loop to complete the design process. The essence of a

verification process is illustrated in Figure 11. In this representation the basics can be seen of the verification process. This process is only worth as much as the data which is put in the process. The definition of a verification process should there for be done properly as otherwise an evaluation will not have any value (Marchant, 2010). To give a verification value, validation of a requirement should be done. This ensures that the verification can be done correctly.



**Figure 11: Essence of the verification Process**

The validity of a requirement remains a difficult endeavor. To ensure this validity, Shishko & Aster have defined procedures to ensure that requirements are unambiguous, traceable, correct and well defined. When this definition is properly given before a design is made, the design will improve and the verification process will become easier (Shishko & Aster, 2007). To ensure the completeness of the verification process, Haskins have defined the steps to undertake to ensure proper verification. This consist of three steps; Preparing, Performing and manage the result of verification (INCOSE, 2015).

The first step consists of defining the strategy for verification in a project according to the costs and risks. In this step the definition of what should be verified (requirements, characteristics etc.) is firstly defined and with what procedures they are verified. After this the constraints are defined towards the execution. Lastly in the preparation of the verification the availability of information should be taken care of to ensure that the execution can be done easily. This all should be laid down in the verification plan. In this plan the definition of the verification, the success criteria, the used verification method, the needed data and the enablers is laid down (INCOSE, 2015). The second step is to execute the verification according to the plan and analyze the results. These results should be communicated upon to evaluate action which should be taken to deal with non-complying elements.

When looking into the origin of mistakes within the verification process, the following reasons can be found according to Marchant, 2010; Ambiguous, incorrect or incomplete requirements, incorrect allocation of requirements and missing elements. These reasons an all cause mistakes if the verification process isn't properly evaluated and validated. These reasons can result in a positive outcome of verification, but are actually failing due to the incompleteness or incorrectness. This appearance can be misleading and can lead to extensive rework and costs if discovered later on in the project.

The underestimation of this process can therefore be risky. This is also exactly the case within the AEC industry. The traditional way of working within the AEC industry relies heavily on the craftsmanship of constructors to deliver a building that is suitable for usage. Due to the growing complexity and the usage of integrated contracts, verification of requirements has become an important process (Bouwend Nederland, 2014). Underestimation of the importance of verification can result in costly mistakes. A big opportunity lies here to improve the design process to ensure that designs are complying with the requirements and improve the quality.

### 3.3 Rule Checking

As construction designs need to comply with various building codes, standards and other requirements, the evaluation of the performance of a design towards requirements is an important part of the design process. This process of checking compliance is growing in complexity as the amount of criteria and the amount of requirements from clients is growing (Nawari & Alsaffar, 2015). Checking the compliance of performance is mostly a hands on job which relies on the knowledge of the people executing this check and the processes to manage completeness (Eastman, Lee, Jeong, & Lee, 2009). This heavily relies on the knowledge and the interpretation of the executor. As interpretation of the requirements and codes can be an error prone process, the use of an automated process in checking can be very useful (Nawari & Alsaffar, 2015; Zhang, Teizer, Lee, Eastman, & Venugopal, 2013). When the interpretation of these requirements is laid down in a rule, the need for repeated interpretation by specialists is not needed anymore, this can reduce the chance on errors and reduce the time it takes to evaluate designs on every design rule (Solihin & Eastman, 2015). Capturing this knowledge into a rule enables the possibility to create a knowledge system to test building designs (Hjelseth & Nisbet, 2010). The value of improving the quality of checking a model and reducing the time it takes to inspect a model has been acknowledged by the AEC sector. The definition of a rule checker is described by Eastman as “assessing designs on the basis of the configuration of the objects, their relations or attributes” (Eastman et al., 2009). The outcome of this assessment is the evaluation if the object is configured properly according to the requirements set by building codes or clients. The implementation of a rule checker for client specific requirements can lead to improvement. The possible achievements can be found in the improved performance, reduced building mistakes, improved efficiency and reduced costs (Dimyadi & Amor, 2013).

Rule checking for building codes already exist from the 1980's and has been researched upon greatly (Dimyadi & Amor, 2013). Over 300 researches have been reviewed by Dimyadi et al. The researches have mostly focused on rule checkers on standards and regulations as these are easier to handle due to their more static character. The main focus has been on the prescriptive parts of building codes, leaving the performance based code receiving less attention. This is explained due to the more difficult character of these rules as a combination is needed between human knowledge, interpretation and translation into code.

#### 3.3.1 Types of rule checkers

The purpose of a rule checker is to assess designs by checking models to their configuration towards the requested performance (Eastman et al., 2009; Hjelseth & Nisbet, 2010). The answer to this check is always a yes or no (or unknown) on the given rules (Pauwels et al., 2011). These rules will have constraints were they are tested on. Hjelseth & Nisbet have defined that, there four different intentions for model checks. These are validating systems, guiding systems, adaptive systems and content based checking (Hjelseth & Nisbet, 2010).

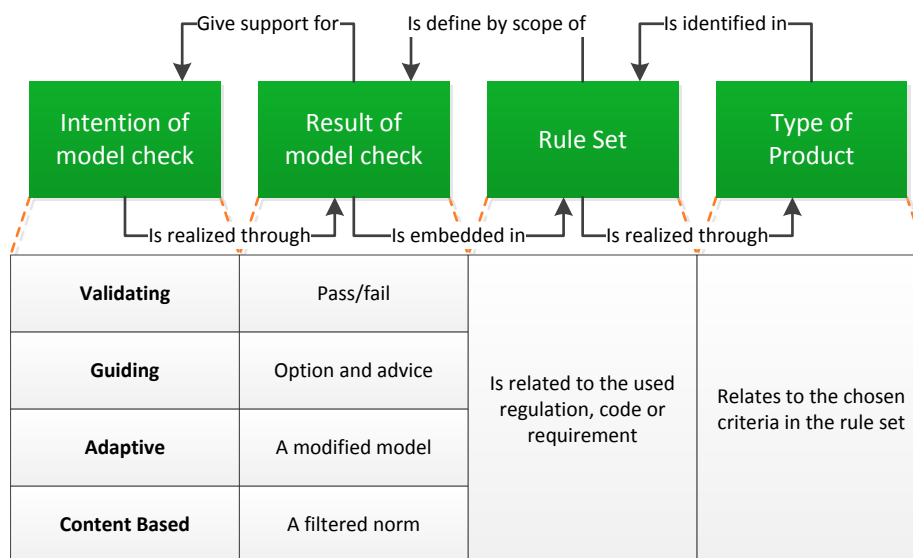


Figure 12: Overview of model checking concepts, based upon Hjelseth & Nisbet, 2010



The guiding and adaptive systems look into the possibility of using model checks for guiding to or generating a design solution. The validating systems and content based checks are more of an examining character. Content based checks are easily to create and check if certain content is apparent. Validating systems checks examine the validity of a design. The concept of model checks is concluded by Hjelseth & Nisbet in an overview where the intentions are stated according to the ontology behind it. This overview is given in Figure 12. In this figure, in green the logic in the ontology is shown, and underneath the explanation is given per model checking concept.

There are two basic types of validation model checks. Firstly there is the Geometry based checking and secondly there is compliancy checking (Hjelseth & Nisbet, 2010). Model checking is based upon topological relationships and Boolean algebra to define if the geometry of a design is complying with the given rules. For example the required floor space can be calculated with model checking. Also clash detection is a good example of model checking. Compliancy checking has the purpose to check if the design is according to the building codes and/ or requirements which apply in a building project. Solihin & Eastman have defined that the technique of rule checking can be used for seven categories. These are stated in Table 2.

**Table 2: Rule Checking Categories (Solihin & Eastman, 2015)**

Category	Explanation of rule category
Syntactic build-up model checking	<i>Is the model according to the modelling agreements?</i>
Building code regulation checking	<i>Is the model complying with the building code?</i>
Client specific requirements checking	<i>Is the model complying with the requirements of a client?</i>
Constructability and contractor requirements checking	<i>Is the model constructible in the way it is designed?</i>
Construction safety checking	<i>Are safety regulations guaranteed during construction in the model?</i>
Warranty approval checking	<i>Has the model issues regarding warranty requirements?</i>
BIM data completeness checking	<i>Is the data according the requirements for facility management?</i>

This research will focus on the third category; client specific requirements. These are the requirements which are realized by the client to achieve certain qualities they need for a suitable environment according to the needs and functions of the building. This category for rule checking can be executed according the various intentions described by Hjelseth & Nisbet as the built-up of the requirements can vary in definition from validation checks to content based checks. The definition of the rules is defined by the structure of the requirements which are going to be evaluated in the model. Solihin & Eastman have defined a classification for the various types of rules (Solihin & Eastman, 2015). This classification has been set up according to the complexity of processing the rules. Eventually the data (IFC) which is going to be checked can be reduced to the attributes of the elements of a model. With this in mind four rules classes have been defined, these are stated in Listing 1.

**Listing 1: Rule Classes**

- Class 1: Rules that require a single or small number of explicit data
- Class 2: Rules that require simple derived attribute values
- Class 3: Rules that require extended data structure
- Class 4: Rules that require a “proof of solution”

**Class 1 rules**

These rules deal with explicit attribute data within a building model (Solihin & Eastman, 2015). The data which is checked in this rule class is accessible directly in the model and doesn't require extensive preparation. The access to the data can be done with basic queries to evaluate availability of data or relations.

**Class 2 Rules**

These rules deal with checks that are based upon single values or small data sets (Solihin & Eastman, 2015). New data does not have to be generated but multiple actions and arithmetic or trigonometric calculations could be needed to execute these rules. Implicit relationships within the requirements should be identified in this class of rules.

### Class 3 Rules

The rules in this class require the calculation of extended data (Solihin & Eastman, 2015). This means that data must be generated to execute the check. The complexity in this rule class is that the generation of this data can be done in various ways. The execution of these rules often relies on a geometry engine that can evaluate the model on its geometry and relations with the use of algorithms and calculations.

### Class 4 Rules

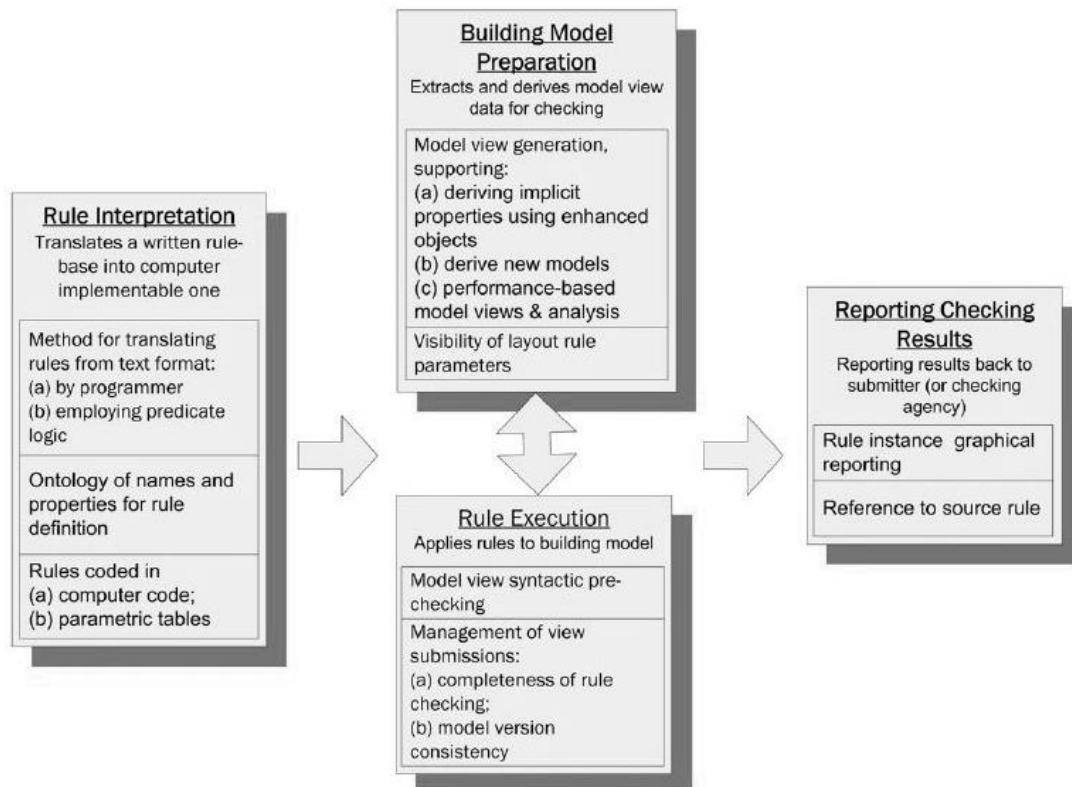
Normal rules evaluate if there is compliancy in the model according to requirements. These rules will give a Pass, fail or error. Class 4 describes the rules which require a proof of solution (Solihin & Eastman, 2015). This requires describing how a model passes the rules instead of just complying with the prescribed rules. This class focusses to capture the knowledge of a solution which is required. This gives the possibility to improve the solution. The complexity of these rules does not have to rise in comparison with the other classes but has a different focus. In the Table 3 some examples have been given of the different classes.

**Table 3: Examples rule classes**

Rule Class	Example
Class 1	<i>An internal wall should be 100mm thick minimum</i>
Class 2	<i>When a space has occupancy class 3 and is adjacent to a hallway, the wall in between should have a noise resistance level of 45dB.</i>
Class 3	<i>An office space should be 30 meters away maximum from an emergency exit or exit stairs</i>
Class 4	<i>Determine a zone where the placement of smoke detectors should be in on a flat sealing of corridors, when a smoke detector should be at least 300 mm away from a wall and the distance between smoke detectors is not more than 10 meters (Solihin &amp; Eastman, 2015)</i>

### 3.3.2 Rule checking process

The process of automated rule checking is defined by Eastman et al. into four steps; rule interpretation, building model preparation, rule execution and reporting checking results and is described in Figure 13 (Eastman et al., 2009);



**Figure 13: Process of rule checking (Eastman et al., 2009)**

Rule checking depends on two major parts to deliver validated results; The model which is going to be checked and the interpretation of the rules (Solihin & Eastman, 2015). As rules never apply to all the elements in a model the definition of which elements should be checked is important. The completeness of the availability of information is also essential for the execution of a rule. Here for the interpretation of a rule is important as this defines which elements the rule applies to. This usability of a rule can depend on three various ways of information availability (Eastman et al., 2009);

1. explicit information provision by a designer
2. Computer generated data
3. Information generated by complex simulation and/ or analysis

The definition of this availability of data for a rule is an important concern to evaluate what the strategy is for the creation of the rule checker, which data is needed and how the rule should be executed. The varying types of data influence the four steps of rule checking. These will now be elaborated on.

### **3.3.2.1 Rule interpretation**

The interpretation of a rule is defined as the most crucial step in the rule checking process as here the definition of the check is determined (C. Eastman et al., 2009; Hjelseth & Nisbet, 2010; Pauwels et al., 2011). Various ways are discussed upon the literature to define a rule in natural language into computer processable code. This translation greatly relies on the type of rule and the intention of the rule check. Eastman et al. has identified two different ways of implementing interpretation of a rule (Eastman et al., 2009). The first implementation relies on the interpretation of the programmer translating the rule into code. The second implementation relies on the logic in a rule in natural language which then is mapped into processable language. This leads to two ways of translating rules into conditions which can be processed; computer language encoded rules and parametric tables (Eastman et al., 2009).

The translation of requirements into rules comes with certain issues. Solihin & Eastman have defined two major issues which should be considered; Arity and combinatorial issues (Solihin & Eastman, 2015). Arity issues are occurring from the fact that requirements are hardly answered by only one answer. The ability of a model to answer towards a rule in various ways should always be considered properly. Combinatorial issues arise from the fact that multiple conditions need to apply on elements. The combination of multiple applicable rules needs to be evaluated properly if the combination of rules is complying.

### **3.3.2.2 Building Model preparation**

A building model has become object orientated with the introduction of BIM (Eastman et al., 2009). This means that the definition laid down in an object defines what the object is, whereas with 2D CAD drawing the visual correctness defined what an object was. This has increased the importance of correct modeling as for example a wall is only a wall when it is modelled with the use of wall objects. This correctness of modelling is needed as building models are becoming large and complex data sets with the use of BIM.

When evaluating buildings on rules, the evaluation of a total model will not be useful as a rule only applies to certain aspects of a building. The usage of the total dataset of a building model in a rule checking will make the check ineffective. The usage of Model View Definitions has therefore been initialized by BuildingSmart. A Model View Definition (MVD) is a subset of the data schema of a building model for exchanging data (Chipman, Liebich, & Weise, 2016). IFC is built up in a way that the information in a model can be used according to various level of development (BuildingSmart, 2016). A MVD defines which data is needed for an exchange. In this way the required data is made available for rule checking. The definition of what data is required relies on the requirements which are going to be checked. The definition of the properties which need to be evaluated must be defined properly before a MVD can be created as otherwise the rule checking will be incomplete (Eastman et al., 2009). The definition of the data which is needed depends on what kind of class of rule is going to be executed. Explicit data needs less effort to prepare a building model in comparison to a performance based rule which might require a simulation. A proper validation of the building model preparation is always needed to ensure reliability of the rule checking before it is utilized.

### **3.3.2.3 Rule Execution**

In the rule execution phase the rule checker will be executed. Before a rule can be executed the whole automated checking environment needs to be validated. This means that the interpretation of the rule will be evaluated to ensure that the interpretation is complete and correct (Eastman et al., 2009). Also the syntactic built up of checker must be evaluated to ensure that the checking will work as required. Aside from the interpretation also the model view needs to be validated. If this validation step is not done properly incompleteness can cause false presumed compliance. Aside from validation also the management of model versions consistency needs to be assessed. When iterations of the design hasn't been taken into account in the to be checked model a false presumed compliancy can be perceived.

### **3.3.2.4 Reporting checking results**

In the last phase of rule checking the results are evaluated (Eastman et al., 2009). A rule will have a fail, pass or error as result. Publishing the result should be easily understandable and can be made visible with the use of a viewer. Addressing the fails and errors is an important step to ensure complying designs. Also the documentation of objects which have passed is important to prove the performance of a design towards a client or regulatory organization. The link with the original requirement should always be made traceable to ensure the documentation and the reasoning is made clear and understandable. This makes the documentation which can be generated understandable for the reviewing party like a client or government.

### **3.3.3 Rule checking platforms**

In the AEC sector various code compliance software platforms are used. Various researches have indicated the following important platforms which make use of rule checking; Solibri Model Checker, Jotne EDModelChecker, FORNAX and SMARTcodes (Dimyadi & Amor, 2013; Eastman et al., 2009; Hjelseth & Nisbet, 2010; Kasim, Li, Rezgui, & Beach, 2013; Krijnen & van Berlo, 2016; Nawari & Alsaffar, 2015; P. Pauwels et al., 2011). These platforms have been evaluated greatly upon literature and are applicable for the AEC industry. They all use Industry Foundation Classes (IFC) data files but vary in their way they are built up. Eastman et al. have discussed the applicability of these four platforms regarding the process of rule checking.

### **3.3.4 Difficulties in automated rule checking**

Due to the vast amount of research which is done upon rule checking some clear difficulties can be found in creation and execution of a rule checker. These difficulties can be related to the steps in the rule checking process.

#### **3.3.4.1 Difficulties in rule interpretation**

In the creation of rule checking a lot of difficulties can be found in the rule interpretation. The main problem in rule interpretation can be found in the structure of a requirement. In section 3.2.3 the complexity in requirements has been defined. The main issues here can be found in the variety of requirements, the way they are built up and how they are related to other requirements and elements. The definition of a requirement requires proper interpretation as otherwise the wrong assumptions are made. This interpretation can vary among different actors due to their background and experience (Solihin & Eastman, 2015).

According to Solihin & Eastman, the complexity of translating rules can be found in the technical built up of a rule. The language structure, the embedded domain knowledge and the logic structure in a rule are three main factors that cause the complexity (Solihin & Eastman, 2015). This complexity can be translated to the consistency and the knowledge which is required to translate the requirement into a rule.

To translate a requirement into a rule, multiple approaches can be used. An important factor for creating a rule is the reusability and applicability of a rule. The flexibility of the language which is chosen can limit the possibilities of usage (Pauwels et al., 2011). This comes from the fact that computer coded language and parametric language are limited in their applicability to their own environment. The usage of logic theory based rule language can overcome this issue. This enables the possibility of using the rule checking environment on multiple environments. The usage of semantic web technologies have been suggested for this purpose by Pauwels et al. and Zhang & Beetz.

Another difficulty in the creation of a rule checker is information loss (Solihin & Eastman, 2015). The knowledge which is needed for interpretation must be maintained as this gives the value to a rule checker. The communication in the rule creation process is there for critical but remains a challenge in this process. Also after the rule is created the knowledge which is captured should be always retraceable to ensure that maintaining the usability of a rule can be done easily.

#### **3.3.4.2 Difficulties in model preparation**

To ensure that a rule can be checked a building model must be made usable. With this process, difficulties come forward which need to be addressed before a rule checker can be executed.

The first issue which needs to be dealt with in the preparation of a building model for rule checking is the Interoperability of the data of a building. The interoperability of data amongst the construction sector has been identified as one of the crucial issues (Beetz, Coebergh van den Braak, Botter, Zlatanova, & de Laat, 2015; Dimyadi & Amor, 2013; Young Jr. et al., 2007). The data needed for a rule checker needs to be accessible easily. This requires interoperability as the data which is required for the check can be located in various locations and platforms. If the interoperability isn't taken care for, the needed data for the model check can be incomplete, causing an incomplete checking.

The second issue can be found in the use of IFC. As described by Pauwels et al. two issues have been defined which need to be dealt with. Firstly there is the difficulty in partitioning the information in IFC to make it possible to deal with the magnitude of the IFC schema (Pauwels et al., 2011). The creation of model view definition remains difficult and is still researched upon greatly to define proper partitions. Secondly there is the issue of modelling properly with IFC. This comes from the fact that for modeling information there are multiple ways of describing the information within IFC. These two issues are mostly related to the description of IFC with the use of EXPRESS language which is difficult to enhance.

The third issue in preparing the building model for rule checking can be summarized in to consistency issues. The way a building model is built up relies on the way it is modelled. Efforts to standardize this modelling improve the consistency of models among the various projects. Despite the fact that standardized ways of modelling are occurring, the consistency in a model depends a lot on the quality of the modelling.

The validity of a building model is also an important aspect which should be addressed in the preparation for a rule checker. The validation of building models addresses the quality of the IFC model on their syntactic built-up and the quality of an export or import for interoperability (Zhang, Beetz, & Weise, 2015). The difficulty in defining a validated IFC instance lies in the fact that a model can be built up in various ways due to the openness of the IFC schema. For example a constructive quality if a wall is loadbearing can be described with a variety of ways in IFC. To evaluate IFC models on their validity, model checking for IFC validation is being reviewed upon by various researchers.

A fifth issue regards the usability of an IFC model. As stated before the right information should be available. This process defines the usability of an IFC model. To make sure that the right information is reviewed, model view definitions are used. Creating validated and reusable model views remains a difficult endeavor (Zhang, Beetz, & Vries, 2013). Reusable model views are needed to make the model checker usable in the various environments and domains. The difficulty here comes from two reasons. Firstly there can be various ways to represent the information. This can result in stating the data in a different way but meaning the same thing. Secondly this comes from the fact that to create useful model views, technical background of IFC is needed. This comes from the fact that implicit information in an IFC model isn't attainable without interpretation of IFC experts.

#### **3.3.4.3 Difficulties in execution and reporting**

Within the execution and reporting phase of the rule checking process there are less difficulties. The most important difficulty in the execution is the maintenance of a rule checker. As building codes can changes as well as requirements there must be a possibility to update a rule. To not disrupt the logic in a rule the built up of a rule must be done in a way the rule is updatable (Pauwels et al., 2011). The costs to keep a rule



checker updated can be high as the process of validation might be needed to be done again to ensure the usability (Dimyadi, Pauwels, Spearpoint, Clifton, & Amor, 2015; Zhong et al., 2012).

Aside from updating the checker along the usage amongst various projects, also the timing of when a checker is used can be difficult. This depends on the availability of data. To ensure consistent results the data which need to be checked need to be available. This data availability depends on the modeler. If a design is in an early phase, the data which is needed to check might not be available yet. The Level Of Development (LOD) is a major factor that affects the building model (Solihin & Eastman, 2015). The development of elements in a building model varies among the different phases. This needs to be taken into account when executing the check as when a doing a check which requires the development of the information to be at a LOD 300, checking data which isn't at this LOD yet will not be useful. Making agreements towards the level of development is there for essential to take into account during the various phases. This ensures that when verification of requirements will be done, the right level of development is achieved as otherwise a check will not be useful.

**Table 4: Overview of possible difficulties in rule checking development**

<b>Difficulty</b>	<b>Description</b>
<b>Rule Preparation</b>	
Structure of requirements	<i>Complexity due to the variety of requirements, the way they are built up and how they are related to other requirements and elements</i>
Technical built up of rules	<i>Complexity in technical built up due to the language structure, the embedded domain knowledge and the logic structure in a rule</i>
Flexibility of language for built up rule	<i>The flexibility of the language which is chosen can limit the possibilities of usage due to the limitation of the applicable environments</i>
Knowledge loss	<i>Translating the rule requires interpretation, during the phases of rule development this gained knowledge can get lost</i>
<b>Building Model Preparation</b>	
Interoperability of data	<i>Easy accessible data requires interoperability which can cause difficulties</i>
Issues with IFC	<i>Difficulties with partitioning IFC &amp; multiple modelling options in IFC due to various ways of describing information</i>
Consistency issues	<i>Consistency problems due to lacking standards</i>
Model validation	<i>Difficulty in validation lies in the multiple options in describing data</i>
Usability MVD	<i>Creating valid MVD's requires technical IFC background</i>
<b>Execution and reporting</b>	
Updateability of checker	<i>Updateable rules are needed. Updating rules is complex and costly</i>
Timing of checking	<i>The timing of a check relies on availability of data, the level of development required is there for important to be defined before a checker is used</i>

## 4. Qualitative research

### 4.1 Motivation

The goal of this research is to investigate the design process and the possibilities of automation of the verification process. In order to do this, a clear understanding is needed on the design process and the verification process. Aside from literature an overview of the current practice in the design process is needed to evaluate where possibilities in automation lie.

To understand which steps are taken in the design and verification process interviews are held to give an accurate representation of the current situation. This way the problems which are now occurring in the design process can be found and prerequisites for automation can be described. For the design of the interviews, sub questions have been used to address the needed information.

### 4.2 Interview setup

The first four research questions are part of the qualitative research. These questions are researched upon with the use of a literature review and Interviews. The interviews are used to address the current situation and problems within a utility building contractor in the design process. As can be seen in the research description, this research focuses on different elements of the design process. The research focusses on the usage of BIM and SE and where they are coming together in the design process. The following research questions are addressed;

- 1 **What steps can be found in the design process and how do they align with the Systems engineering process?**
- 2 **What type of requirements can be found in building projects and have the biggest impact in the case of non-conformity?**
- 3 **What is the common practice in verification in the design process?**
- 4 **What does automation of verification implicate for the design process?**

The goal for the interviews per research question will now be discussed.

#### **1. What type of requirements can be found in building projects and have the biggest impact in the case of non-conformity?**

In this question there are two parts which need to be looked upon, namely the design process and Systems Engineering within the construction sector. The mistakes which are most often occurring during the design process are discussed upon in the interviews in depth to see what kinds of issues are occurring. Also the needed improvements are discussed to see if the need to improve is equal among the different roles amongst the interviewees.

When looked into the design process, the implementation of Systems Engineering is also an important part of the interviews. The implications of implementing Systems Engineering are reviewed to see where complications rise from. This is needed to look upon as BIM and SE come together in verification in the implementation. These complications are needed to evaluate to draw up preconditions for automation.

#### **Goals:**

- Identify the problems which are now occurring in the design process
- Define where these problems arise from
- Investigate the problems which arise with the implementation of Systems Engineering in the design process of a construction project
- Evaluate how BIM and Systems Engineering are interlaced in the design process

## **2. What type of requirements can be found in building projects and have the biggest impact in the case of non-conformity?**

Goals from this research question for the interviews are to discover what kind of requirements can be categorized and how a requirement is built up. Also the problems with verification of requirements are described here to investigate how the process of working with requirements could be improved.

### **Goals:**

- Discover the different kinds of requirements
- Investigate how requirements are built up and what problems can arise with usage
- Define which type of requirements give the most problems

## **3. What is the common practice in verification in the design process?**

To implement automation of verification, the total process of verification needs to be evaluated. The essence of a good verification will there for be discussed in the interview to identify the key factors of a good verification process within the design process. The interlacing with the design process is investigated in the questions regarding this part. The verifications which are most difficult and are the timeliest will also need to be investigated. After this the origin of mistakes made in verification is also going to be questioned upon.

### **Goals:**

- Evaluate total verification process to identify key elements
- Evaluate interlacing of verification in the design process
- Define what kind of errors arise and where they originate from in the verification process

## **4. What does automation of verification implicate for the design process?**

When the verification process is described, the implications for the design process need to be defined to set a scope for the prototype for the requirements checker in the next part of the research. In this part of the interviews, these preconditions are discussed together with the benefits of automating the verification process. Also the implications for the design process are discussed to evaluate if the automation is possible.

### **Goals:**

- Define the benefits of automation of verification for the design process
- Describe the precondition for automation of verification
- Investigate the improvements needed in the design process to facilitate automation

### **Setup per subject**

In the description of the goal of the interviews, the four research questions are evaluated on what the goals are behind the research question. These goals can be translated back eventually to four subthemes. These subthemes are the design process, Systems Engineering, verification and automation of verification. With the use of the goals of the sub-questions the interview questions have been created. The interview questions can be seen in Annex A. The goal per subject will now be discussed.

### **Design process**

The purpose of this part is to get insight in the current practice to see where the biggest problems come forward from. In this way a clear insight can be given in what way mistakes occur. This is needed to see if these problems are related to the verification process and to see where the opportunities lie for creating a tool.



### Systems Engineering

To see where systems engineering and BIM come together, a clear overview of the current practice is needed. This gives also a good insight to see how SE is already interlaced in the design process and where the implementation is limited. As working with systems engineering gives the possibility of structuring information properly the current limitations should be evaluated as verification is where SE and BIM come together.

### Verification

To ensure improved quality in designing, the current practice in verification should be clear. This is reviewed upon in the interview questions by looking into the preparation of a verification, where mistakes are made in and what the causes of these mistakes are.

### Automation of verification

To evaluate if the goal if this research is possible, questions regarding the pre conditions of automated verification are asked. In this way the benefits of automation can be confirmed, the pre conditions can be identified and a clear overview can be given about what requirements are suitable for automation.

## 4.3 Interview outcome

In total twelve interviews have been held among different backgrounds. A variety of people with different functions have been interviewed. The goal of this research is to see how the design process can be improved by looking into the process of verification and the information flows accompanying this process. The different stakeholders in this process have been interviewed. In Figure 14, the composition of roles is shown. The departments of Systems Engineering & BIM are the main knowledge carriers in this process. Together with design and technical developers the total scope has been addressed. Aside from the utility building sector where this research focusses on, also an interview has been done with a systems engineer from the Infrastructure branch to look into their approach.

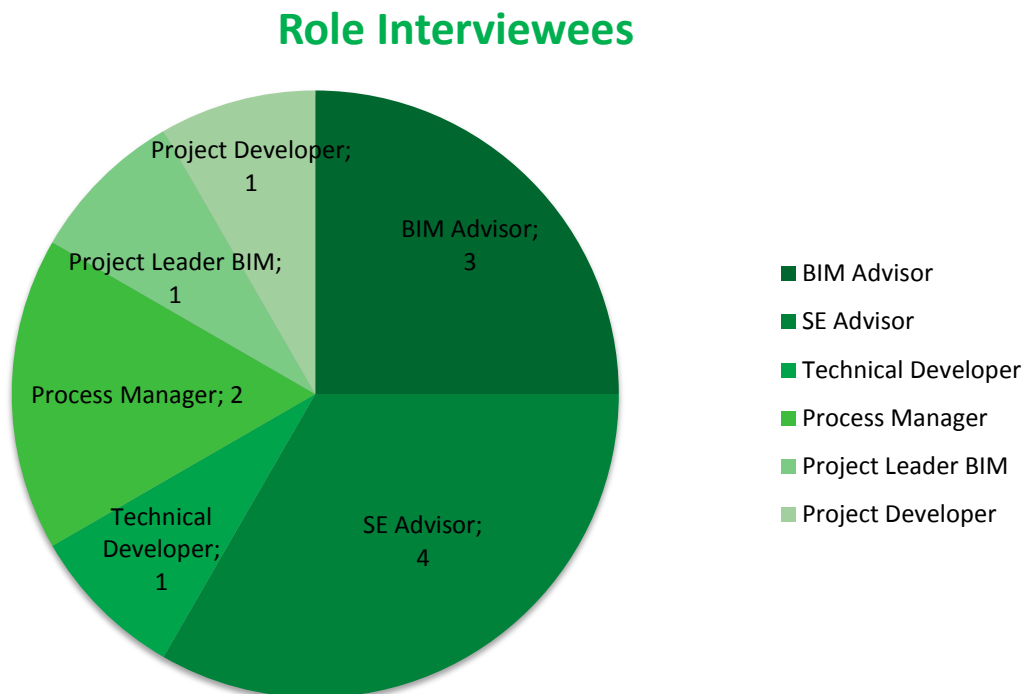


Figure 14: Role interviewees

The interviews were held in a semi structured way. This means that a guideline of questions were is followed but there are possibilities to deviate from this questions to get more in-depth information and examples. The interviews have been recorded and transcribed as the variety of the response needs to be analyzed to be

made useful for concluding on the initial research questions. The responses of the interviewees have been analyzed and per interview question. A conclusion has been made per answer of a respondent. These conclusions have been coded and per question the comparison between the different respondents have been made. These conclusions per question have been evaluated in the interview report. These conclusions per question are combined into a total conclusion per subject. These subjects are the design process, Verification and automation of verification.

### **4.3.1 Design Process**

#### **4.3.1.1 Information alignment**

The main problem in the design process can be found in the information streams. As in integrated contracts, proving performance is of major importance, the information which provides this proof is essential to be valid and consistent. There are a few information streams which can be identified. The first stream is the customer requirements information. This stream is where the translation is made from customer requirements to design prerequisites. The information where the solution is created is where the answer is given to the customer requirements. The understanding of the requirements is crucial to achieve the product the customer wants and how it should work. When a proper understanding is achieved, the parts where requirements are applying to, need to be made clear. This is a way of structuring information and where a link is created between requirements and parts of a design. The definition on which parts of the design requirements apply, defines where the answer is given and the two different information processes come together.

When the requirements are clear, a good allocation is needed to link the need to a part of the design. The problem which can be found here is that before allocation can be done, a system design should be made. This is a difficult process, but should be done with precision as this allocation links the requirements and the objects. When the allocation is not done correctly, problems can arise from missing allocations to parts of the design. This implicates that before a design is made actually a system already is set up from the requirements, in this system the relations between elements is already made from the information which is generated in the requirements specification. Without this clear definition of the system, mistakes can be easily made during the design process and a structure for proving the performance is difficult to make.

The information streamlining is crucial as the different streams are depending on each other. To talk about the same thing in the customer requirements, the design and in the verification of the design, is essential to facilitate proving performance and to give insight in what the quality of the design is. If the information in requirements and information in a 3d model is not aligned, talking about the same things will be difficult. The variation in projects and clients influences the information structure of requirements greatly. Standardization is difficult in these requirements which results in a vast variation in structure. This is the main reason why allocation is difficult.

The information stream of the 3D model is improving more and more as the agreements which have been made in the construction sector result in clearer data structure with consistent information. This is done by making agreements on naming and modelling for BIM.

There are a few conditions before the total process can be proven to be working correctly. The first condition is that the interpretation of requirements is done correctly. The allocation is the next condition to provide a process that works properly and to provide a system that talks about the same things. To prevent comparing apples to oranges, is the most important thing to prevent problems during the verification process, a good interpretation and allocation are the first conditions to prevent this. Aside from allocating objects to requirements, also other information needs to be allocated to objects and requirements. The main reason why this allocation must be done properly is to give a connection meaning as a wrong allocation will create a comparison which isn't valid. The process of allocation which is portrayed in Figure 15 will now be discussed per subject.

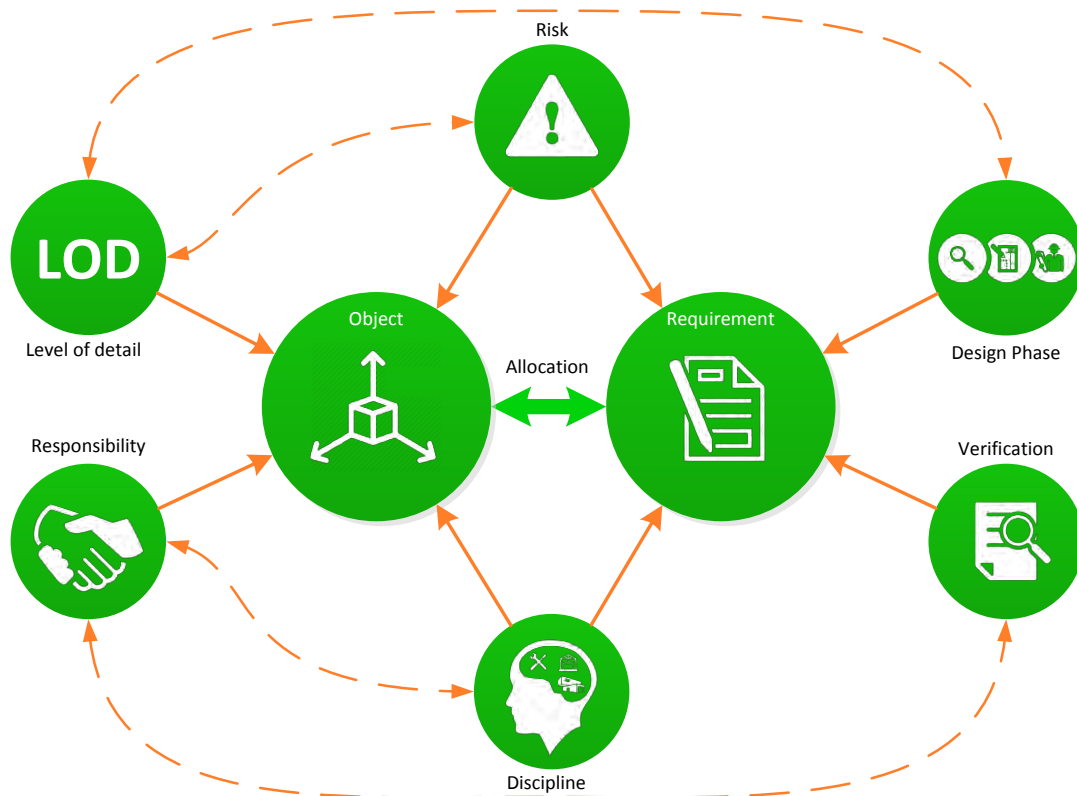


Figure 15: Allocation of Object and Requirements (own Drawing)

### Allocation of objects and requirements

The allocation of objects which are going to be created and requirements should always be done before a design is created. The definition of an object can be seen in two ways, the objects which are developed in creating a system from a requirement and the objects which are created in a design. These two different objects should always be mapped to each other on an object level and allocated to a requirement. The complexity in this part can be found in the fact that multiple objects can be allocated to one requirement but also vice versa that multiple requirements can be allocated to one object. This plurality in allocation can cause incompleteness in correct allocation due to a possible difficulty in the clarity of the total system.

### Acting discipline

The right disciplines should always be involved in making decisions on objects. This is the basis of integrated design and the basis of linking requirements and objects. The definition of which disciplines are active in requirements and objects should be done integrally as a first step. This gives a clear overview which disciplines are involved in working with different requirements. This results in a clear structure where can be seen which people are needed to be involved in designing objects. The responsibility is in this way also connected to make it visible which people are the acting in creating the design.

### Level of risk

Aside from defining who are responsible for creating the object coming from the requirements, the level of risk of requirements and to be created objects have a major impact on an integrated design process. Risk is defined as the possibility of the occurrence of a negative scenario. The risks in requirements can be found in the probability of not complying with requirements and the probability of extra costs in the design process. The analysis of risks is there for essential in the design process. When the impact of risks is assessed, this should be linked to the affected requirements and objects. The implications of iterations should always be investigated with a look into the allocation of object and requirements. Sometimes it can be useful to iterate a design into a much higher level of detail to simulate the effects of the decisions. Therefore the link between risk and level of detail exists.

### Level of Detail

According to a design phase the level of detail of an object should be determined. This influences per phase what the minimal level of detail in an object there should be. When an object has a higher risk level, the level of detail can be needed to be higher to ensure that iterating prevents unpleasant surprises. An overall equal level of detail is needed per design phase to do meaningful comparisons in verification. When the level of detail of different objects is not equal among different disciplines, the design process can be disturbed. This is due to fact that the implications on requirements of different elements aren't comparable and measurable.

### Responsibility

The person which is responsible for creating an object should be linked to an object as this creates better insight in designing an object. It also ensures that every element is allocated in responsibility. This responsibility is for creating the object but also for doing the verification. The relation between responsibility and disciplines should be considered to ensure that the right disciplines are involved in creating an element.

### Applicable Design Phase

A requirement should always be evaluated in which design phase a solution should be created. This depends on the level of detail which is asked in the requirement. It is sometimes needed to deduct a parent requirement when a requirement already needs to be discussed in an earlier phase on. This can be done to manage a risk level of requirement.

### Verification

Before a design is created the proper verification for a requirement should be defined. As the objects are already allocated to requirements, the verification is the element which verifies if the proposed object is complying with the definition of the requirement. The way the verification should be done, should be defined before a design is created. When this is thought of beforehand a clear method can be defined which should be measurable. The person responsible for this verification is always also the person which creates the object.

#### 4.3.1.2 Interpretation of requirements

A client doesn't always perfectly know how the building should look like and how it should work. This insight is created during the design process. As the requirements are often written in natural language, the interpretation of a designer can vary greatly from what a customer wants. The validation from a client of the interpretation of a designer towards the interpretation is important to achieve what a client wants. In Figure 16, the deviation in interpretation between designer and customer can be seen. The reason why this deviation is occurring is the openness and the ambiguity in the statement of the requirements. This deviation in interpretation needs to be minimized to prevent design mistakes. A dialogue with a customer can strengthen this understanding and help with validating the interpretation of the designer.



Figure 16: Deviation interpretation of requirements

The reason why an extended requirements analysis isn't done can be found in two reasons. Firstly the construction sector is used to start designing straight away and amend its design due to the iterative character of the design process. Due to this effect, not enough time is taken to fully understand the need of the client. The second reason lies in the fact that the investment costs of extending the requirements analysis are won back after a tender is won. This brings forward a dilemma as not every tender is won, which makes it impossible to win back every investment in extended requirements analysis. This dilemma can be overcome by making the understanding of a project a core element to win a tender. To use this to as a strategy to win will need more investigation on how to exploit this opportunity.

Another aspect of interpretation of requirements lies in the plurality of applicability of requirements. A lot of requirements are being filled in by multiple elements. This can be seen in two ways. Firstly a requirement can be filled in by a combination of different objects. For example a fire compartment is built up out of different walls which together make the sum if a compartment is complying towards the requirements from a standard. Secondly an object can have multiple requirements being applicable. This combination of requirements or objects being applicable makes it complex to create a complying design solution.

The impact of requirements was assessed in the interviews. Interpretation and the plurality in applicability of requirements are the essential factors which affect the complexity of a requirement. This can also be found in which requirements are identified as the most complex requirements. The requirements which need a simulation to prove their performance are designated as the most complex requirements. In the interviews, comfort related requirements are confirmed as the most influential requirements which often cause mistakes.

In conclusion of the interpretation of the requirements an overview has been given of what benefits an improved requirements analysis can have in Figure 17.

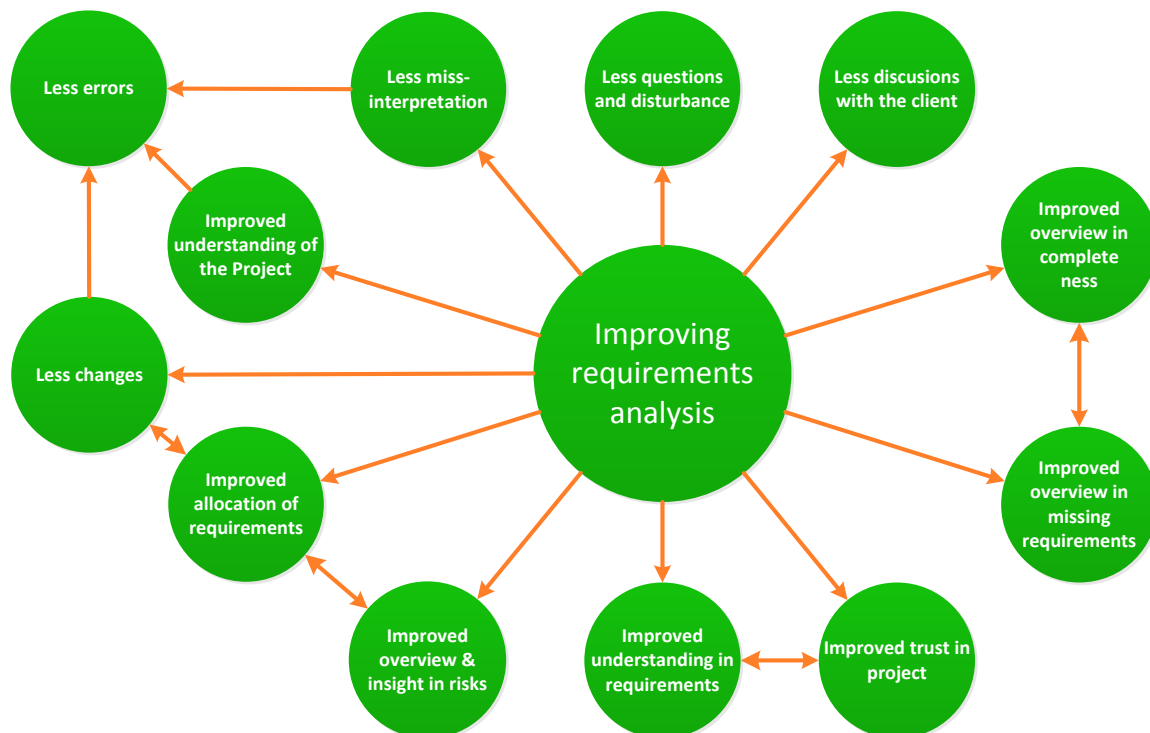


Figure 17: Improving requirements analysis

#### 4.3.2 Verification

The quality of the verification process depends on the process which takes place before a requirement is done. Also the verification is only worth as much as the definition of the steps behind the verification. This means that before a verification can be done properly, some preconditions are present. These pre conditions

are also present for the design process. For the total verification process pre conditions have been defined with the use of the interview response;

- The interpretation of the requirement must be validated by the client
- The allocation to the proper elements must have been done in a way that all the elements which a requirement is applying to are allocated. In this way all requirements are applying to all the elements in the design
- The process of defining if a verification complies or not must be done in such a way that an unambiguous answer can be given.
- The definition of the level of detail of a design allocated in the time according to the level of risk must be done properly. This prevents discovering mistakes too late as higher risk requirements need to be monitored closely.
- The level of detail of different disciplines needs to be defined clearly to validate the verification as the different levels of detail from different disciplines can inflict problems due to occurring changes. These changes can occur as a deeper level of detail can inflict changes on different disciplines.

#### **4.3.2.1 Verification Process**

The key elements of verification are related closely to the definitions laid down in the design process. The key element of verification is to construct the right comparison. This means that to do a proper verification, the comparison of a requirement and object must be correct and defined clearly. The actual process therefor consists of three steps:

1. Preparation: The definition of the elements used in the verification must be clear. This is done mostly in the design phase where the allocation is defined as discussed in 4.3.1.1.
2. Defining the verification procedure: The process of selecting the right procedure and defining the rules of a verification.
3. Verification during the design process: After the verification procedure is defined and the elements are designed, the verification can be executed and documented

This total process is captured in annex B.

#### **4.3.2.2 Requirements classification**

During the interviews multiple definitions have been given about the classification of requirements. The first definition which has been addressed often is the level of ambiguity or interpretation in requirements. Here a clear difference between the following three types of requirements is defined;

1. Value (numerical) requirements
2. Relational requirements
3. Textual requirements

In these three types the level of ambiguity can rise easily when the definitions are not laid down clearly. Textual requirements often are difficult to measure and are highly interpretable.

Aside from the classification in measurability also classifications in the interviews are brought forward related to the various disciplines in construction. For example technical requirements are different in comparison to architectural requirements. These requirements are related mostly to different qualities of a building, for example technical requirements are often related to comfort requirements and architectural requirements are more related to aesthetic quality. Aside from the requirements which can be classified in certain domains, also the requirements related to norms and standards can be found. Norms and standards are often related to safety, security and usability. A variation in sources and domains where requirements are related to is defined by varying needs of a client. This can be seen in for example that aesthetics are more architectural requirements, technical requirements are more related to comfort and norms and standards are applying more towards safety and usability.



### **4.3.3 Automation of Verification**

#### **4.3.3.1 Possibilities for using rule checking**

As rule checking is already used for validation of models, the usability of rule checking for automated verification is also questioned upon in the interviews. The interviewees responded on these questions that the way a model checker works should be possible to use for checking a large part of the requirements. Only non-quantifiable, more qualitative requirements like look & feel requirements will remain impossible to automate. An important conclusion although is that a model checker will only be useful if the process of verification is validated and the information is correctly. If this isn't the case a rule checker still can be built but the meaning of the check will not be valid. The most important pre condition for using rule checking is there for that the Information is consistent on both sides (requirement & model) and that the requirement is translatable to a non-ambiguous rule.

#### **4.3.3.2 Pre conditions for automation of verification**

The preconditions for automation of verification have interviewed upon. Before the verification can be automated the following conditions must be addressed in the preparation phase of verification;

- Requirements are measurable/ translated in an equation
- Interpretation of requirement is validated with the client
- Allocation of requirements and objects is done properly
- The level of detail needed in the associated design phase is defined for the verification
- For every verification the right information must be defined

This means that the total process which happens before verification is done is clear and consistent. The definitions of a requirement and the verification should be clear to ensure that the proper data is used for the verification. To prevent doing rework every project, this process needs to be standardized to implement in every project. Before verification should be done, a data quality should be done to check if verification can actually be done. The completeness of data is needed to ensure that the verification is complete.

In the execution phase of verification the following conditions must be addressed;

- The definition of a complying solution must be given in an unambiguous equation where a clear not complying or complying is given
- Definition of requirement must be reusable for different projects
- The information and the value of what a requirement demands must be available in the BIM model

The procedure of verification should have a clear equation with needed values made clear. These values should be available and allocated to the right objects.

The eventual way of constructing the automation is also defined in the interviews. This can be defined as followed:

1. Data quality check
2. Querying the data
3. Equation if a requirement is met
4. Create documentation for the outcome

This data quality is needed to ensure that the preparation conditions are checked and to be sure if the right information is available for the verification. If this isn't the case, the information is modelled wrongly or not available. After this the right information is needed to be brought forward as not the whole model should be queried. This ensures that the right information is questioned upon and reduces the needed computing power for the checker. This requires that model information needs to be consistent throughout different projects as otherwise the query will not be complete.

## 4.4 Conclusions on research questions

The conclusions which are made for research questions 1 to 4 are given with the use of the literature and the interviews. These will now be given.

### 1. What steps can be found in the design process and how do they align with the Systems engineering process?

The steps of the design phase and the systems engineering process have been identified. The level of detail and level of development of a design increases along the steps in a design. From a conceptual way of defining the need of a client in a building, the conceptual design is improved amongst the phases of a project to a more performance based design. These steps of increased level of detail happen simultaneously with the decrease of variants. The conceptual decisions made integrally develop the definition of a design. These exact steps are also proposed in a systems engineering approached process of a design. The decisions made in a systems engineering approached process follow these steps of iteration in a same way.

To improve the design process the development of a design should be monitored closely and the integral decisions should be laid down properly as a baseline. The level of detail and development amongst these steps should be equally amongst the different disciplines to assure integrality in the decisions for variants. For making design decisions having the correct information is essential.

The main problems in the design process have been identified in the interviews. Firstly the understanding of integrated contracts is not at a level where it should be, this is due mostly to the transition which is happening in responsibility. This transition originates from the introduction of integral contracts which require developing a design with a complete building team much earlier on. Not only an architect is there for acting in an early phase, but a complete team with all disciplines is defining the building. In this field also a change is coming in the technical area of a building as this discipline is required to work more conceptually and parametric in comparison to traditional ways of developing a design.

Aside from the main way of working together also the information streamlining in a design process causes issues. The lack in standard ways of delivering information makes it difficult to assure that the right information is available. This is primarily the case in requirements data. A standardized process is not apparent in defining the requirements. This causes a lot of human interpretation to work with the requirements which can cause errors easily due to misinterpretation.

An eventual improvement of implementation of systems engineering can be found in a more standardized way of working and on the other hand improvement in experience in working with Systems Engineering. Key in this improvement is the understanding of a complete project team of integrated contracts and the goals. An important is found in understanding the set of requirements and the need of the client in an early phase. This is crucial as the decisions made in the early stages of the project are the most vital decisions as the impact of changes later on in a project have major impact in costs and time.

### 2. Which requirements can be found most often in projects and have the biggest impact in the case of non-conformity?

Requirements are researched upon how they are built up to identify what types of requirements exist. Requirements can be defined in the structural built up of a requirement and in the classification towards needs.

Firstly the structural built up of a requirement can be defined as quantifiable, relational and qualitative. This defines the built up of a requirement and where the requirement is applying to. This can be a value, a relation or a qualitative description. This closely relates to the measurability and the level of ambiguity of a requirement. These levels define the complexity in working with a requirement. This structural built up of a requirement defines the first hierarchy in the structure in defining requirements

Secondly requirements relate to a certain need and/or discipline. For example comfort requirements relate closely to technical installations. A relation between a need and a requirement is an important part to create



structure in a requirement. This translation and iteration of a need into a certain requirement brings forward the second hierarchy of defining a requirement.

The iteration process from a need towards a requirement is essential for the clarity in the requirements. This process starts with a requirements analysis to understand the meaning of a requirement, on which elements it applies to, which stakeholders are involved and what level of detail in which phase is required. The requirements analysis doesn't receive enough focus yet. An important cause why this requirements analysis isn't executed extensively lies in the changes in responsibility but also in the way tenders are executed in the construction sector. The interpretation of the requirements coming from the analysis requires client validation. This validation with the client is essential to ensure that the right thing is designed. This is not a common process yet in the construction sector.

Non conformity regarding requirements relate to structures in built up of requirement. The way a requirement is built up can have a big impact on the design process. Misinterpretation of a requirement is often occurring in the design process due to the way a requirement is built up structurally.

The requirements which have the biggest impact are in the end the requirements which have the highest priority towards the needs of the client. This closely relates to the risk of a requirement. Aside from this also the timing of identifying non conformity is essential to measure the impact. A requirement which isn't complying investigated in an early phase has a smaller impact as a requirement which isn't complying in a more developed phase. Progressing in iteration on a mistake increases this impact. Creating a solution that complies after every phase is there for essential to manage the risks.

The process to manage requirements will rely on the reusability of the defined structures. This will enable to standardize the way of working and can improve the usability of requirements to measure the performance of a design.

### **3. What is the common practice in verification in the design process?**

Verification in the design process, currently takes place after a phase in the project is finished. This is a hands-on process where the quality of the execution relies on the interpretation of the person who executes the verification.

A process of verification has been defined in the interviews. Here three elements have been identified. This is the definition and allocation of the requirements, the preparation of the verification and the execution of the verification. The equation which is used in the execution of the verification defines how useful the verification will be. This verification process relies on the completeness and the quality of the requirements. The performance in a requirement defines in the end the usability of a verification.

### **4. What does automation of verification implicate for the design process?**

Eventually automating the verification of client specific requirements relies mostly on the preparation and the definition of requirements. When the system which is going to be checked is inconsistent or incomplete automation will remain useless. The process improvements in the field of requirements management is there for essential for implementing automated verification in the design process. The most important parts of these process improvements are defining a measurable equation where a clear and measurable performance is defined, the completeness defined by the correct allocation and lastly the equality in level of development amongst the different disciplines.

Aside from the process of defining the system of requirements and allocated objects the data which is going to be checked needs to be consistent and correct. Therefore the data quality of the design solution in a BIM must be complete and correct. When the checker is correctly built up but the data which is going to be checked is inconsistent, the results will not be valid.

## 5. Model

### 5.1 Introduction

Changes in responsibilities are occurring more and more in the AEC industry. Due to integrated contracts, contractors need to prove their performance more structurally. Together with the need of preventing building mistakes, implementing methods to monitor and manage quality in complex building projects has become an urgent matter. The use of systems engineering and building information modelling (BIM) are important developments to improve the building sector. Verification of a model according to the requirements is an important part where systems engineering and BIM can come together to see if the required quality is achieved. The verification process is mostly a hands-on job where interpretation of the requirements can be of high influence. This manual process is time consuming and error prone and must improve to achieve the required higher quality in the construction sector.

The translation of requirements and evaluating if they comply can be seen as rule checking. The process of rule checking within the AEC sector is researched upon greatly. These researches focus mainly on building code compliancy and the validity of an Industry Foundation Classes (IFC) file. Using rule checking for client specific requirements has received less attention. The possibility of using rule checking in the verification process can result in an unambiguous automated verification process which is less error prone. This automation can eventually lead to the possibility of using requirements as variables for automated design solutions.

#### 5.1.1 Research Problem

The main problem investigated in this research can be found in the data management in the current verification process in the design phases. The verification process is still a manual check of data which is error prone as the quality of the interpretation of requirements defines the quality of verification. Capturing the knowledge required for verification in a knowledge system isn't happening in the AEC industry yet. This results in the fact that in every unique project, a verification method must be defined for every requirement. This costs major amounts of time in preparation and in execution of verification. Capturing the required knowledge for verification in a model checker therefor gives a big opportunity to reduce errors and saving time. Model checkers already exist in various forms. Developing a checker requires extended expertise in requirements engineering, modelling data and programming. This makes the usage, maintenance and development of a model checker only worth investing for repetitive and frequent checks. The main problem with model checkers therefor lies in the fact that they are not open for reuse and that they are not usable for non-programming experts. To make a checker usable for automated verification a model checker must be;

- Flexible in use
- Easy to expand
- Understandable for the laymen
- Time saving in comparison to the current verification process

To investigate the problem of opening up the complex data of a BIM model to the laymen will therefore be important to define a rule checker that can be used by a non-programming expert. This requires aside from developing a rule checker with an interface in the domain language also proper alignment with the current design and verification process.

#### 5.1.2 Importance

Investigating the data management in verification in the design process is of great importance as this ensures the quality of a building. This process can ensure that mistakes are discovered early on and prevent costly mistakes in execution. As object and requirements come together here, the interrelation of this data is researched upon. This can improve the usability of the data in a building model in the design process.

Aside from the data management, the automation possibilities are of great importance to research upon as this can improve the design process. With the use of automation, repetitive administrative work can be minimized and the focus can be brought upon the essential work in a building project. The investigation on

automation of verification is a useful part where quality improvement & assurance, data management and automation of administrative work come together.

### 5.1.3 Previous Work

Various model checking applications have been developed. These checkers are often closed down to use for the laymen. Aside from that also these checkers aren't open to develop as they are provided by software suppliers as a black box.

As creating a rule requires high expertise on interpretation and on programming, the various researches have confirmed that the definition of a rule and the processing with the use of programming should be separated from each other (Eastman et al., 2009; Zhang & Beetz, 2015). To enable this structure, using semantic web technologies have been suggested by various researches to develop a model checking environment (Beetz, 2009; Pauwels et al., 2011; Zhang & Beetz, 2015). This enables the possibility of using more than just the IFC data as the semantic web makes it possible to create linked data sets. Aside from additional open data, using the semantic web in a rule checker, also the possibility comes forward to query IFC data. To ensure understandability of using the semantic web standard for model checking a brief introduction will now be given.

The semantic web is a proposition where the vast amount of data stored in various data sets is made available as a web of data on the world wide web (W3C, 2011). The current web is a web of documents. The information in this web is stored in the documents. The knowledge is therefore encapsulated in the document and not in the web.

Connecting the data in these documents remains a human interpretation (Allemang & Hendler, 2011). A lot of data is currently not available to be connected to each other as the data is stored in the various applications. In documents links can be made between certain elements in the real world but on a data level this connection is not there. By opening up and connecting this data as web of linked data, a smart web of data can be created. This web of data can be used to investigate relations between the various elements which are described. For example a connection in data about a geographical location, information about what is allowed to be built at that location and data about the population income can give an interesting and easy to create analysis on what kind of development is possible. This basic link in data is an example for numerous linked data connections which can help developing real world solutions.

To describe this data, all the elements are described as resources which can be accessed through the web. Describing these resources in the semantic web is done with the use of the Resource Description Framework (RDF). Describing data with the use of resources in RDF enables the connection between various datasets. In RDF, data is described as a triple statement where the various unique resources are used. These resources are defined as Unique Resource Identifiers (URI) which can be called upon with an unique http address (Curry et al., 2013). With these URI's, a triple statement can be made about relational data. For example the statement "Amsterdam is the capital of the Netherlands" can be broken down with the use of URI's. This can lead to the following triple stated in Figure 18.

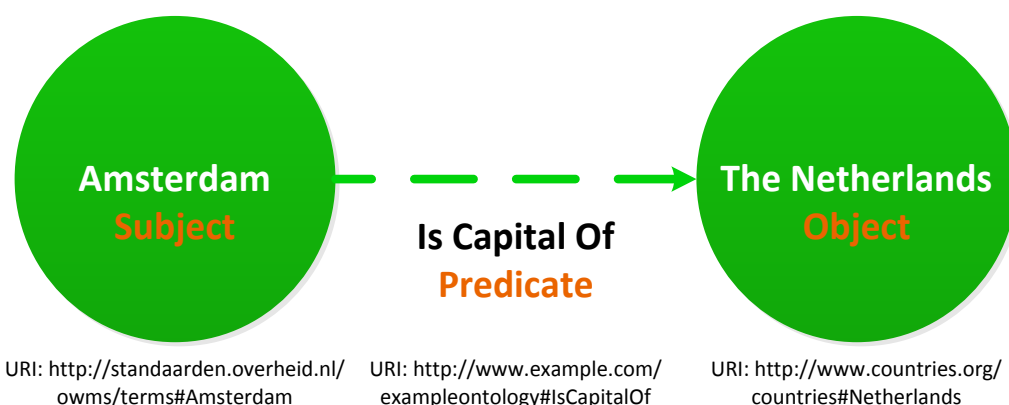


Figure 18: Example of triple graph with URI

The basic building block for RDF can be seen here. This building block consists of a subject, predicate and an object (Allemang & Hendler, 2011). The subject is where the statement is about. The predicate is the identifier which defines the property and the object is the value which is related to the subject via the predicate. In this way all kinds of relations between URI's can be made. The development of these links creates a vast network of data.

To extend the usability of RDF, the RDF Schema (RDFS) is developed. RDFS is an extension of RDF with a vocabulary to define classes, hierarchy and properties (Bruijn et al., 2008). In this way a hierarchy is created in the relations of the triples. A schema can then be built up and used for the triples. For example with the use of RDFS the relation to a subclass can be created to give more meaning to the unique resources.

To further extend the expressiveness of RDFS, the Web Ontology Language (OWL) has been developed (Bruijn et al., 2008). With the use of OWL more detailed relations and constraints between classes, entities and properties can be described (Allemang & Hendler, 2011).

The data set for the semantic web must be accessible to be useful. Data in the semantic web is retrieved with http. Querying is done with the use of the query language SPARQL (Allemang & Hendler, 2011). SPARQL stands for SPARQL Protocol and RDF Query Language. With the use of SPARQL, the various relations in the triples can be accessed. Aside from simple querying also inferencing can be done with the use of SPARQL. This makes it possible to enable links between the various graphs and databases (Feigenbaum, 2009). The structure of a SPARQL query is shown in Table 5.

**Table 5: Structure of a SPARQL Query (Feigenbaum, 2009)**

Structure part	Goal	Example
Prefix declarations	<i>Abbreviating URI's</i>	# <i>prefix declarations</i> PREFIX foo: <http://example.com/resources/> ...
Dataset definition	<i>Definition to be queried RDF Graph</i>	# <i>dataset definition</i> FROM ...
Result clause	<i>What information needs to be returned</i>	# <i>result clause</i> SELECT ...
Query pattern	<i>What and how to query</i>	# <i>query pattern</i> WHERE { ...
Query Modifiers	<i>Rearrangement for the query result</i>	# <i>query modifiers</i> ORDER BY ...

With the use of a SPARQL query various results can be retrieved from a dataset. The result clause query patterns and modifiers facilitate the variety of results. For example from a dataset the various instances can be retrieved with the use of SPARQL. An example query with SPARQL can be seen in Listing 2.

**Listing 2: Example Query**

```

1# SELECT ?City ?country
2# WHERE {
3#     ?city example:isCapitalOf ?Country .
4#     ?city a geoinfo:capital .
5#     ?Country a geoinfo:Country .
6#         LIMIT { 10 .
7#     } .
8# }
```

By this query, city and country variables are retrieved. The relationship which is required, is that all cities must be selected which are a capital of a country. This query will therefore result in a list of capitals and corresponding countries. To retrieve this result, the variables `?city` and `?country` are defined in this query. A `?city` is defined as a `geo:capital` and a `?country` is defined by `geo:country`. The prefix defines where the data can be found. With the use of these prefixes, the query becomes more readable and easier to build up. Additions to the query are made with modifiers. In the example a limit is added. In this way, the result becomes more suitable for the associated goal. Also filters can be added to the pattern to retrieve specific results.

Aside from retrieving data, also rules and constraints can be tested on a dataset. This can be done with three basic rules of the SPARQL Inference Notation (SPIN)<sup>1</sup> (Knublauch, 2011). There are three class description properties in SPIN. The `spin:constraint` which defines a condition for a class, a `spin:rule` which is used to specify inference rules for SPARQL and the `spin:constructor` which can be used to initialize new instances with default values. In this way a rule can be created for a data set. This opens up the possibility of defining if the data in a RDF graph is according to the requirements. Exactly this possibility makes the use of the semantic web for rule checking useful. The usage of SPIN will therefore be important for evaluating the implications of automated verification.

#### 5.1.4 Primary hypothesis & objective

From the problem in streamlining information in the verification process, a prototype for automated verification is developed which is available to the laymen. The hypothesis of this research states that automated verification will improve the design process by reducing the time taken for verification and reduces the amount of errors which can be made. The objective of developing the tool is to investigate which information and elements are crucial of influence to ensure automated verification. This development is created to overcome the problems with existing rule checkers. This delivers a prototype which is based upon the semantic web standard. This gives the possibility to add various forms of information and use this information for various purposes. For the design process an evaluation will be coming forward on what steps in the process need to be undertaken to ensure that the data will be useful.

## 5.2 Method

The method used in this thesis for the creation of the automated rule checker is firstly the process for rule checking described by Eastman et al., 2009 and the method for building the model checker by Zhang & Beetz, 2015. Eastman has described a standardized process on how to define an automated rule checker. This process consists of four steps; Rule Interpretation, Building Model preparation, Rule Execution and Reporting checking results (Eastman et al., 2009). The purpose of the rule checking defines how the checker is built up. The various types of rule checkers all follow this creation process in a way to develop and assess the checker. These steps are followed in the development for the model checker for automated verification.

The model checker in this thesis has the purpose to automatically check client specific requirements. As described in the introduction and in the pre conditions of automated verification, the model checker must be able to be used by non-programmers and must be extendable. This is done by using the proposed method of Zhang & Beetz. They have proposed an architecture for a model checker which uses semantic web standards for checking the validity of a IFC model according to the Statsbygg BIM standard. The architecture for the model checker is stated in Figure 19. The semantic web standard enables the possibility to create a web of linked data (Curry et al., 2013). The semantic web relies upon relational open data (Radulovic et al., 2015). The relation between various data creates meaning and opens up the possibility of receiving answers on the requirements. These answers come forward from querying the relations between the data. For the usage of the semantic web standards, the non-relational data needs to be converted into relational RDF (Resource Description Framework) data.

---

<sup>1</sup> <http://spinrdf.org/spin.html>

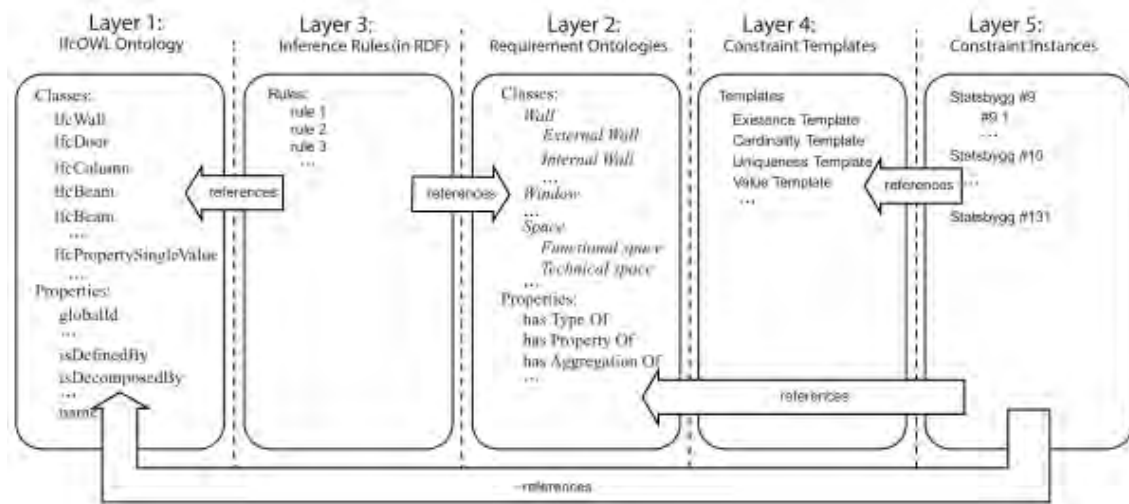


Figure 19: Architecture for Model Checking on the Semantic Web (Zhang & Beetz, 2015)

A requirements checker which can be used by a non-expert is expected to be valuable as every construction expert will be able to use it. To make a requirement checker available to a non-programmer, a check must be available in the construction domain language. This is done by creating an inference layer between the complex IFC data and the domain language. The IFC data is interpreted by the IfcOWL Ontology in Layer 1 of the model. The layer between the ontology and the domain language (layer 3) will function as a library from the IFC standard to the domain language classes and properties (layer 2). Here relations are made between natural language concepts and the related parts of the IFC schema by creating inference rules. Through the relations in this layer, the IFC schema is made usable for non-experts. For example a connection between the natural language concept “Internal wall” and the IfcOWL ontology can be made with the usage of the relations in the translated IFC schema. This is done by creating a link between a Wall in the IFC schema and its property “IsExternal”. This is stated in Listing 3.

#### Listing 3: Example Inference

```

''Internal wall'' = IfcWall or IfcWallStandardCase
                    Pset_WallCommon IsExternal = FALSE

```

In this way a check can be built up with normal language with the right referencing towards IFC without needing the extensive knowledge about IFC. This translation is reusable for various purposes as only the relation is laid down in this layer between the concept internal wall and the parts in the IfcOWL Ontology. This concept can then be used in the various requirements which are applicable on internal walls. The inferences are further explained in section 5.3.2.3.

Requirements are always built up as a constraint with a relation in its statement (Bhatt, Hois, & Kutz, 2012). This makes the usage of the semantic web standard for this kind of checking suitable. The various types of requirements are translated in to constraint templates with the use of SPARQL Inference Notation (SPIN) and stated in layer 4 (Zhang & Beetz, 2015). The relations which are required in a requirement can be queried with these constraint templates. An element where a relation is not existing or which isn't according to the required values is returned as these are the elements which aren't complying. Similar relationships can be found in the various requirements. This enables the possibility of reusing the constraint templates for various requirement checks. For example a constraint for existence of an object in a space can vary greatly as there are numerous objects which can be contained in a space. As long as a relation exists the relation can be checked.

The architecture which is now proposed, defines that there is a system with an undefined amount of requirements which can be checked with the varying classes and properties. To enable this open architecture for the model checker to be used in the verification process, the working of an interface is researched upon to be sure that the process of verification as executed in the design process is followed. This is elaborated



upon in section 5.3.3.1. An important part of this development is to make sure that the client requirements which define the constraint instances in layer 5 are used in the interface development as this improves the usability in the total verification process. This will contribute to using the right constraint templates for a certain requirement and accompanying need. This will eventually give the possibility to evaluate the implications of automated verification.

### **5.2.1 Tasks for developing the requirements checker**

In the rule interpretation process, the definition of a requirement is translated to a computer processable rule. The various types of requirements have been identified in the literature study (section 3.2.3) and the interviews (section 4.3.2.2). Aside from this, an analysis is done on five existing projects to define a list of standard requirements. This list is developed according to the needs of a client in a hierarchical way as proposed by Walraven & de Vries, 2009. This gives a clear structure from value, to need, to requirement, to performance. In this way also the interaction in verification according to a systems engineering approach can be followed as stated in 3.2.2.

These requirement types are evaluated on the applicability for automation. This definition of the requirements will lead to the development of a set of requirement types where the right constraint templates can be defined for. The definition of these requirement types will need to be made SMART to create a set of standardized rules which are measurable according to the performance. A clear definition of the elements and needs is therefore essential. From this set of requirements a selection is made for the prototype.

Aside from defining the requirements, also the elements and properties which are present in these requirements need to be defined. These elements need to be represented as natural language which is reused amongst the various projects. As the prototype will focus on walls and spaces, a definition of these two elements need to be given. For the space definition an investigation on classification of spaces is done to define a summarized classification which applies to the Dutch building Code which can be used openly. The properties and elements which are investigated upon in the prototype need to be evaluated if they are defined in IFC and if how they are defined. This is needed to evaluate which requirements can be translated and which can't without external referencing.

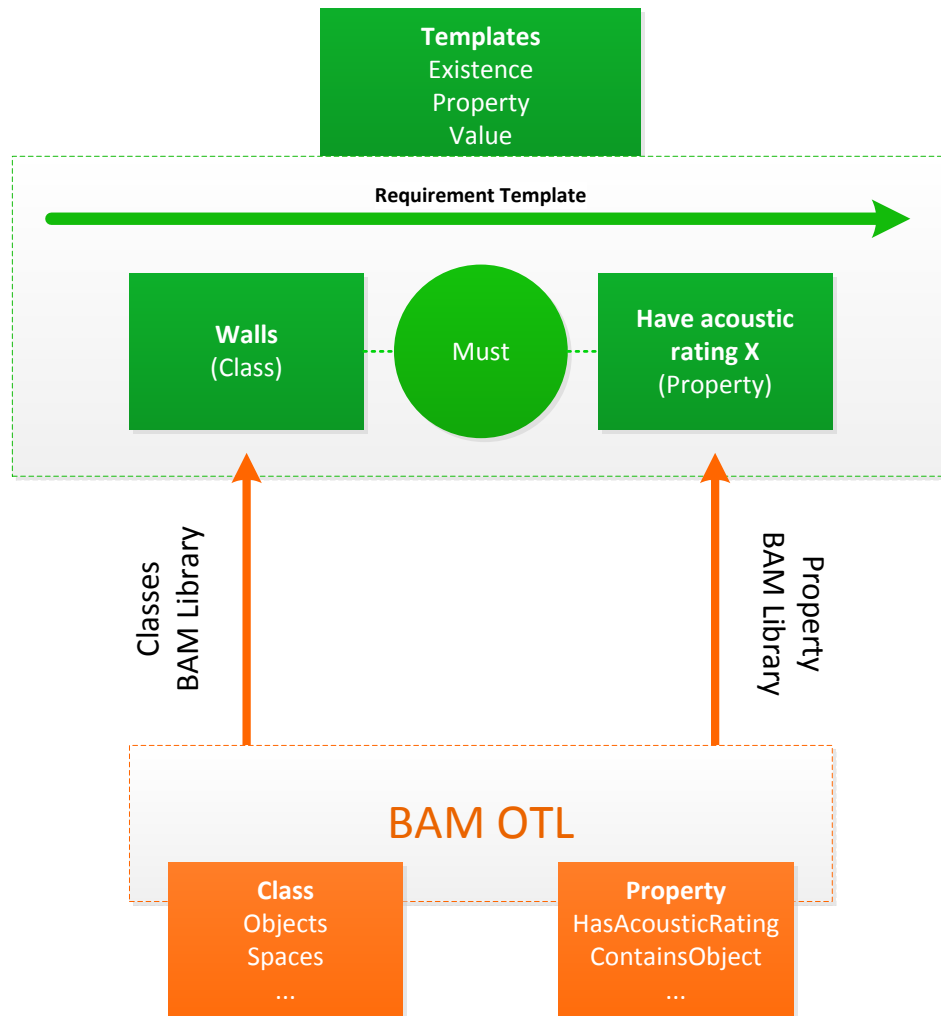
To prepare the building model for automated rule checking, the IFC building model is converted into a RDF data set with the use of the IFC-to-RDF convertor of Pauwels & Terkaj. This enables the possibility to query the IFC data. A further explanation is given in section 5.3.2.2.

As the IFC standard has been chosen to be used for the data input, the right elements in the rule checks need to be matched to the IFC standard. To develop the connection between the natural language concepts and the IFC standard, a connection is made between the IfcOWL ontology and the natural language concepts by using inference rules with SPIN. This will enable that the elements in an IFC file can be used with natural language concepts in the BAM object type library.

When a connection is made from the concepts in the IFC schema, the quality of the building model determines how useful a checker is. A data quality check is therefore essential before a requirements check is executed. Ensuring that the to be evaluated elements are modelled correctly must therefore be done before a check is executed. This is therefore an important part of the execution of a rule checker. This is further explained in the flowchart of the interface. In the prototype, an IFC file is used which is checked on the quality and corrected accordingly. Evaluating the crucial effect of building model quality will also be evaluated along the development of the checker.

In the execution of rule checking the right data needs to be brought forward and evaluated. Here the connection between the requirements and the data in the IFC model is made. By defining a set of requirement types according to the needs, the various constraint templates can be developed where the objects and properties can be used as variables in the templates. By matching the right template to the right requirement, the model checker can be developed. This structure can be seen in Figure 20 and is further explained in the next section.





**Figure 20: Structure of using constraint templates for requirements checking**

After the templates are built up, the right elements and equations need to be created. This is defined by the client requirement. Determining the right equation is essential for a valid verification of a client requirement. Here for the knowledge of a domain expert is needed. The usage of the correct template, object and properties is determined here. This interaction is defined by the process in the verification. The interface which enables this interaction defines how useful a checker will be. Here for the alignment with the current verification is important. This will result in the eventual development of a prototype. This prototype will be developed with the use of Java. The development of a report on performance will be required to complete the total process of a rule check as well as a verification. This is investigated in section 5.3.4.

Aside from the total development of the checker, also the integration with the current process management tool Relatics is evaluated briefly. Here the requirements of the client are stored with a connection to the system breakdown structure and the functions. In this way the place of the checker related to the total design process in can be evaluated.

## 5.3 Results

The definition of the performance of a design is measured with the rule checking environment which is defined in this section. This is done per process step of developing a rule checker as stated in section 3.3. This process follows verification as executed in the design process. Here for a definition of performance of an object is measured according to a requirement of a space. The interaction between a requirement and the performance of an object will become clearer during the development of the prototype. This interaction leads to the answer of the research question of what the implications of automated verification are.

### 5.3.1 Rule interpretation

To define a list of applicable requirements, the structure of requirements is looked upon. The built up of a requirement defines what kind of requirement structure types exist. From these types, varying rule templates can be defined for the requirement checker. Important is the difference between a requirement type and a requirement classification. A requirement type is the way a requirement is built up semantically and a classification is connected to what kind of class a requirement is used for. For example different requirement types vary in statement definition. A requirement about a value is different than a requirement about a relation. These various types define different statements. An example for the classification is related to the need which is described. A requirement about comfort is one type of need and a requirement about spatiality is another type of need. This breakdown structure of a requirement in types and in needs defines what a template eventually is going to be selected for verification. As shown in figure 20, this structure is completed by defining the right elements. Summarizing, building up the rules for the requirements checker will consist of defining the requirement classifications and types, the elements (classes) and the properties which are going to be used in the varying requirement templates.

#### 5.3.1.1 Requirements types

As defined in 3.2.3, a variety of requirements exists. As the requirements checker will need to be complete, an investigation on the various requirements in existing projects is needed. To define a proper list of the various requirements, the structure of developing a requirement must be followed. As defined in 3.2.3 a requirement is created to define a need of a client. A requirement is therefore a translation of a need which on itself is a translation from a certain value. This interaction is important to be defined clearly but is often forgotten during the design process. From a certain need, for example the need of a comfortable building a translation to a requirement will be made. At first, these requirements are only applying to spaces. As a space is not a tangible object, this space requirement needs to be answered for by the elements which define a space. A space is defined by the objects which create its borders. The interaction between a space and objects and together with that the iteration from a space requirement towards an object requirement is essential for defining a performance. This total structure of interactions between space requirements, spaces, object requirements and objects is stated in Figure 21.

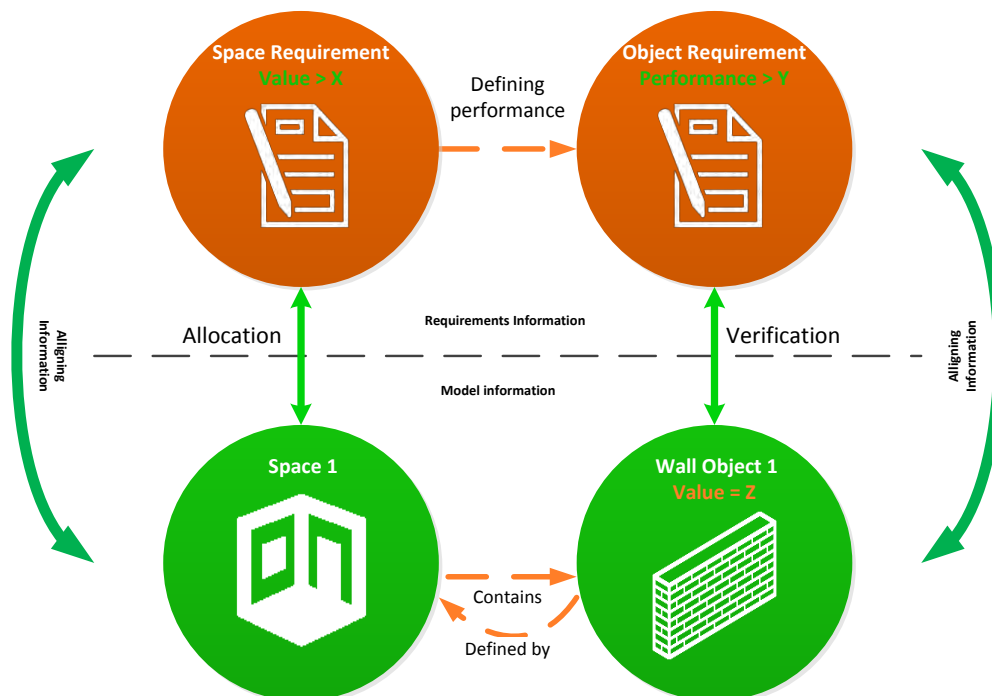


Figure 21: Interaction between information in requirements and objects

To create a complete list of applicable requirements, five existing projects have been investigated upon the various client requirements. The process of these projects is managed with the use of the project

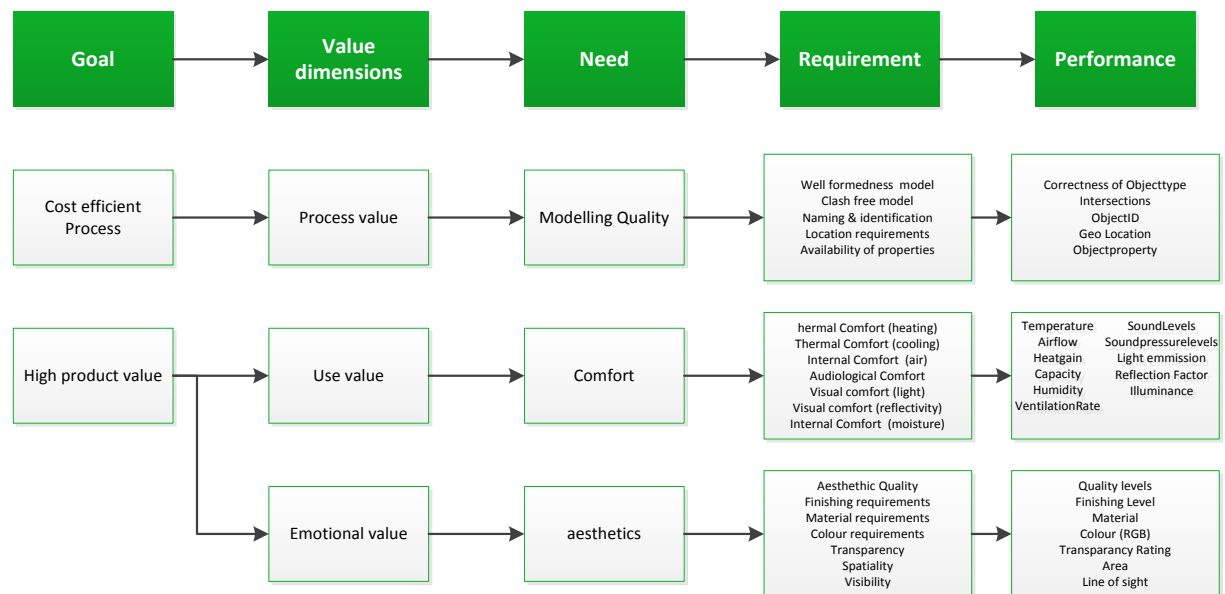
management tool Relatics<sup>2</sup>. In these environments the requirements of the clients have been translated into a manageable interface. In Table 6 a description is given about the projects which are used for the definition of the requirement types.

**Table 6: Description analyzed projects**

	Project 1	Project 2	Project 3	Project 4	Project 5
Type of building	<i>Government Building</i>	<i>Government building</i>	<i>Office Building</i>	<i>Education Building</i>	<i>Government Building</i>
Client type	<i>Government</i>	<i>Government</i>	<i>Developer</i>	<i>Educational institute</i>	<i>Government</i>
Size	<i>13.000m2</i>	<i>6.700m2</i>	<i>32.000m2</i>	<i>40.000m2</i>	<i>80.000m2</i>
Total Number of requirements	<i>4.000</i>	<i>1533</i>	<i>3.580</i>	<i>1.812</i>	<i>13.830</i>

In these data sets various requirements are coming forward. The goal of a requirement is always to define a certain need. As discussed in the literature review, a requirement starts from a certain function and value which is needed to be present in the building, these define certain needs like for example a comfortable building. This basic structure which is defined in section 2.2.3 is the basis for defining the hierarchy of requirements towards performance. An example of the result of this analysis on the various requirements can be seen in Figure 22. The total result of this structure has been stated in Annex C.

Aside from defining the various requirements, the datasets can also be used for analysis on the applicability towards automated verification. The datasets of requirements contain a vast amount of information which can define this applicability. A BIM can only be used for giving an answer to a requirement if the elements described in the requirement are also present in the BIM data. To be useful, the requirements must be iterated to a level which is definable in a BIM. A function of a building or a certain need can't be directly described in a BIM. Needs and function are eventually described by the performance of the element in the data. Therefore the data for a need must be iterated towards a performance of an object as otherwise interpretation will be required to verify a related requirement instead of a performance. Here for a definition of the elements, properties and values in the requirements must align with the similar variables in BIM.



**Figure 22: Requirements Type Classification**

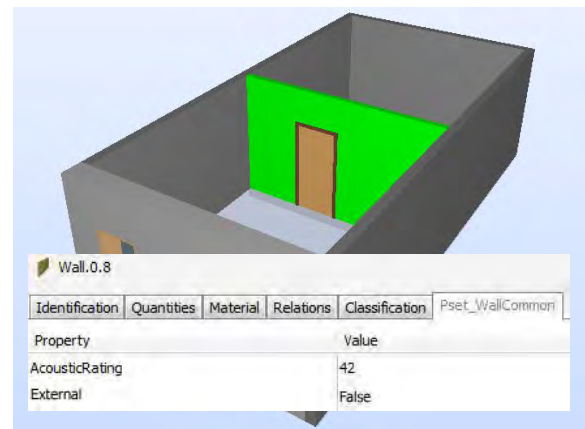
<sup>2</sup> <http://www.relatics.com/>

**Need: Acoustic Comfort**

Space Requirement: Spaces must have noise isolation to prevent noise nuisance

Object Requirement: All internal walls must have an acoustic rating of at least 42 dB

Code	Relation	Requirement	Characteristic	Equation	Waarde	Unit
EIS-00914	Internal Walls	4.10.1 Acoustic Comfort	Value	>	40	dB



Information in Requirement database

Information in BIM Model

**Figure 23: Example of iteration to create an equation with similar elements**

For example a need must be translated into a requirement where clear elements are described with a performance. The BIM model must contain the same elements with properties which describe this performance. An example of this equation of similar objects can be seen in Figure 23. Here a requirement about acoustic comfort is iterated to a level of detail that it can be described in a BIM model. In this way the information in both sources is similar. This definition of when a comparison can be made gives a possibility to measure how much of the requirements in a requirements database can be verified using a BIM.

Furthermore aside from the usability of a BIM for checking a requirement also an overview in the various types of requirements can be given with this data set. This will give a further insight on the built up of these data sets and the opportunities to improve the usability for automated verification and alignment with the design process. The data analysis questions in Table 7 are set up to define the usability of the current requirement datasets.

**Table 7: Data Analysis Questions**

Data analysis question	How to analyze this question
What Percentage of the requirements;	
Is Object related?	Check "Relation to" variable if it is about an object in comparison with total amount of requirements
Is Space related?	Check "Applicable to space" variable in comparison with total amount of requirements
Is defined SMART?	Check "SMART" variable with total amount of requirements
Is usable in BIM?	Define if characteristic of database is BIM checkable, then compare variable total amount
Has a measurable value?	Define what measurable is and analyze the requirements value

The datasets in Relatics have been exported to excel. The datasets vary in their built up. In time the clients get more experience in working with a systems engineering approached project and therefor a variation in structuring the requirements occurs. To compare the data sets, the evaluation of the variables needs to be as identical as possible. The variables which need to be defined for the analysis are the definitions of object requirements, space related requirements, how SMART requirements are, how usable the requirements are in a BIM and what the values are in requirements.

The definitions for space and object variables are defined equally amongst the five projects as the definition of where the requirement is applicable to is defined in all the databases. The unique values are here filtered

and then defined if they are applicable to a space, an object or if they are related to a different element. The definition of a measurable value can be related to an existence of a relation or the existence of a numerical value which can be checked. The value variable is checked if these equations exist and then counted to define how many of the requirements are measurable. Lastly there are the definitions of how SMART the requirements are and if a requirement can be checked using a BIM. Retrieving the information of the variable SMART can be done in three projects easily as this variable is added to every requirement during the requirements analysis of the project. The applicability of this value depends on the interpretation of the person who defined if a requirement is smart or not. This needs to be looked upon to define if this variable definition is already useful.

The last variable which is the most interesting to investigate is the usability of a BIM for checking a requirement. A BIM is usable for checking if the data which defines the performance of the requirement is stored in the model. Therefore this data must be measurable to get a comparison which can give an outcome true or false. This should be possible without extended interpretation. Defining if the variable is measurable is done with the use of the characteristics which are added in four of the five projects. These characteristics are the actual performance indicators of an object. For example in project 3 a list has been defined with these performance indicators. An example is stated in Figure 24.

**Kenmerk Overview**

Naam kenmerk	Eenheid	BIM
Aantal	stuk(s)	Ja
Betonkwaliteit		Ja
Blijvende belasting	kN/m2	Ja
Breedte	mm	Ja
Diepte	mm	Ja
Dikte	mm	Ja
Geluidisolatie	dB	Ja
Geluidniveau	dB(A)	Ja
Hoogte	mm	Ja
Lichtsterkte E(B)	lux	Ja
Materiaal		Ja
Rc-waarde	m2.K/W	Ja
U-waarde	W/m2.K	Ja
Vloeroppervlak	m2	Ja

**Figure 24: Overview of Characteristics within project 3**

With the use of these performance indicators, it can easily be determined if the performance can be defined with the native modeling tool. This is the definition which is used to define what can be used for checking with a BIM. In this way the amount of requirements which can be used in a BIM is measured. The definition if a characteristic can be checked with a BIM is done with the comparison to the availability in the IFC schema. The ambiguity of the characteristics therefore determines the possibility of using a BIM for checking client specific requirements. If these characteristics are too ambiguous or not measurable, it will remain difficult to use the data for requirements checking.

Aside from these characteristics also the relations between spaces and objects can be checked within a BIM. These relations can be found within the requirements dataset if they are defined. The sum of these two types of requirements is the amount of requirements which can be checked with the use of a BIM. The outcome of the five questions is stated in Table 8.

The percentages are retrieved by counting the amount of requirements compared to the total amount of requirements. For the value, object and space requirements little to no interpretation plays part as the elements where the requirement is applying to, are defined in the database. The definition of SMART and BIM usability are subject to interpretation. The SMART variable is only defined in three projects. The quality of this definition is difficult to define as this is an interpretation of the requirement by a single person. The representation of this value remains therefore not reliable for representation.

Table 8: Data analysis come

	Project 1	Project 2	Project 3	Project 4	Project 5	Average
<i>Project Year</i>	2016	2015	2015	2014	2014	2014,8
<i>Total amount of Reqs.</i>	3982	1533	3530	4742	12984	5354,2
<i>Square meters</i>	13.000	6.700	32.000	40.000	80.000	34.340
<i>Requirements per m2</i>	0,306	0,229	0,110	0,119	0,162	0,19
<i>Value Requirements</i>	9,8%	10,4%	11,6%	12,5%	6,2%	<b>10,1%</b>
<i>Object Requirements</i>	22,0%	19,9%	41,8%	37,8%	14,5%	<b>27,2%</b>
<i>Space Requirements</i>	55,7%	31,4%	23,6%	64,1%	58,3%	<b>46,6%</b>
<i>Smart Requirements</i>	17,7%	73,8%	x	8,8%	x	<b>33,4%</b>
<i>Reqs. usable in BIM</i>	17,3%	13,7%	7,2%	15,1%	7,9%	12,2%

For the usability in BIM a different approach has been used. This requires an evaluation of the values of the requirements. The values can be a text, a numerical value or a relation. The numerical and relational value can be brought forward easily as these are added as single values of the requirement. The availability of the data in a BIM is defined by evaluating the value statement in comparison to the availability of the data in the IFC Schema. This was possible to do in Project 1 to 4 with the use of the characteristics of the value statements.

In project 5 the characteristics weren't defined as the method of adding a characteristic to a value wasn't used yet here. Here a definition of all the values is evaluated by checking the unique available values on their applicability. This applicability is defined by the available information in BIM. To define a list of requirements where BIM is possible to use for, Filters on the data set have been used. Firstly all the elements where the requirement is applicable to which isn't available in BIM are filtered out. For example objects like a desk are filtered out. A filter is also used for filtering out norms, activity related requirements and process related requirements. This delivers a list of 1126 values. For these values an evaluation per unique value has been made if they are applicable to be checked with use of a BIM.

Out of this data analysis conclusions are drawn up from the results. The percentage of value requirements is on average only 10%. This means that 10% of the requirements are only quantifiable. This can be one reason why the amount of requirements which are usable in BIM is low, as this variable is on average only 12,2%. This means that only 12,2% of the total set of requirements can be used to be verified with the use of a requirements checker. An increase in the amount of usable requirements can be seen clearly here along the course of the years the projects started. So the usability of using BIM for verifying requirements is increasing. This also can be interpreted in a way that the information streamlining between requirements database and BIM data is improving. A clear improvement is the usage of characteristics as these make it necessary to define a clear value.

Aside from requirements which are defined as values also the definition of where requirements are applying to gives some insight in the composition of a requirements database. The average of requirements which apply on objects is 27,2%. The variation in this question can be related to the moment when the requirements database is initiated. For example the requirements database for project 3 has been initiated in a more developed phase in comparison to project 1 where the database was initiated in a very early phase. The definition of objects in this phase is less developed as the design is still very conceptual. The iteration of requirements towards object related requirements is essential for the usage of BIM for verification as the models which are developed are mostly object related. Using a BIM for verifying more conceptual requirements in early phases can be more difficult. This is due to the fact that it is difficult to use a BIM for describing the usage of a space or a certain quality.

A clear conclusion can be made on the question regarding the Smartness of requirements. As defined in section 3.2.3 SMART stands for Specific, Measurable, Attainable, Realizable and Time bounded. This variable is only as worth as much as the quality of the interpretation of the person who defines if a requirement is smart or not. A conclusion can therefore not be drawn on the percentage of requirements which have the variable SMART.



In total the requirements databases still have a lot of requirements which aren't usable for verification with BIM. This is also due to the fact that a lot of the requirements are related to the usage of certain user and monitoring systems. For example requirements related to the operation of various systems like error reports, fire alarm monitoring or sensors are still difficult to manage with a BIM as these elements are often not described in the model. These requirements are essential for the operation of a building but aren't described in IFC.

Aside from system operation requirements there are also still a lot of requirements which are written down vaguely and multi-interpretable. In these cases contractors should improve the ability of managing these requirements by iterating the requirements into SMART and non-ambiguous statements and clients on the other should improve the way they deliver these requirement databases. Client validation in this process remains the key to ensure that the right product is delivered.

### 5.3.1.2 Requirements Ontology

In the architecture for the requirements checker in Figure 19 (section 5.2), the requirements ontology is displayed in layer 2. This ontology is needed to give meaning to all the requirements types as shown in structure of the requirements checker in figure 20. The BAM object type library will function as the requirements ontology. In this ontology the knowledge about the entities in a building is described. In this way a reusable concept for every building element is developed. For example for a door an OWL concept `bam:door` is created. These kinds of concepts can be used to relate to various sources of information. In this way the data which is needed to built up a requirement is available in a object type library. This is done with the use of the semantic web modelling tool Topbraid Composer<sup>3</sup>. For every different type of element of a building, a concept is created. For the elements classes are developed. The definition of objects and spaces is done with the use of varying classes. the properties which define the performance of the objects are also defined. These are explained in the next sub-sections.

#### 5.3.1.2.1 Space Classes

To define the space class, a look into the current practice of modeling with spaces is needed. In the current process of modelling within BAM, a space is defined as stated by the client. This definition of how a space is classified is stated in the properties of a space object in the attribute as a name. By following this procedure of naming the consistency amongst varying projects is not high. A clear definition of a space amongst the variety of spaces is much more favorable as connections between space requirements and object requirements can then be reused. Here for a classification of spaces is needed.

In the AEC sector, various classifications have been made. Some internationally used examples are Uniclass<sup>4</sup> and Omniclass<sup>5</sup>. These classifications are also defined upon spaces and have many common denominators. In the Netherlands spaces are also defined by certain institutes. For example STABU<sup>6</sup> has defined the STABU Bouwbreed. This definition enables creating documents with reference to a standard structure in spaces. Using this structure would be favorable but the structure of Bouwbreed is unfortunately not as complete as in comparison to Uniclass and Omniclass. These two classifications have a much more in depth definition of the various spaces. Using these classifications would also be favorable but these classifications aren't applicable to the Dutch building code.

For using classifications in the Netherlands, the Dutch Building Code is important to keep in mind. The building code applies varying rules towards different type of spaces. A movement space and a user space bring forward different applicable rules. This is important to keep in mind when defining a classification of spaces. The usage of the existing classes is therefore difficult. For this reason a hierarchy has been proposed based on the building code and with the depth of the Omniclass and Uniclass. The basic structure is shown in Figure 25. In this figure the subclasses are left out. The complete proposed schema of the space hierarchy has been stated in annex D.

---

<sup>3</sup> <http://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/>

<sup>4</sup> <https://toolkit.thenbs.com/articles/classification/>

<sup>5</sup> <http://www.omniclass.org/tables.asp>

<sup>6</sup> <http://www.stabu.org/producten/stabu-bouwbreed/>



This structure should be built up as an ontology. By using an ontology the information can be mapped to the other classifications. To build up this ontology, Topbraid Composer is used. These elements are defined with the applicable hierarchy as stated in Figure 22. In this way more meaning is given to the objects with the use of subclasses. The usage of these elements in the modelling depends on how the reference is created towards the ontology. In this thesis the current practice in modelling of BAM is followed. In the current practice an element is defined only as a name and is not linked to the ontology directly. An interesting improvement for BAM will be to use the classifications instead of naming. This will not be used in this thesis as this isn't applicable to the use cases provided by BAM.

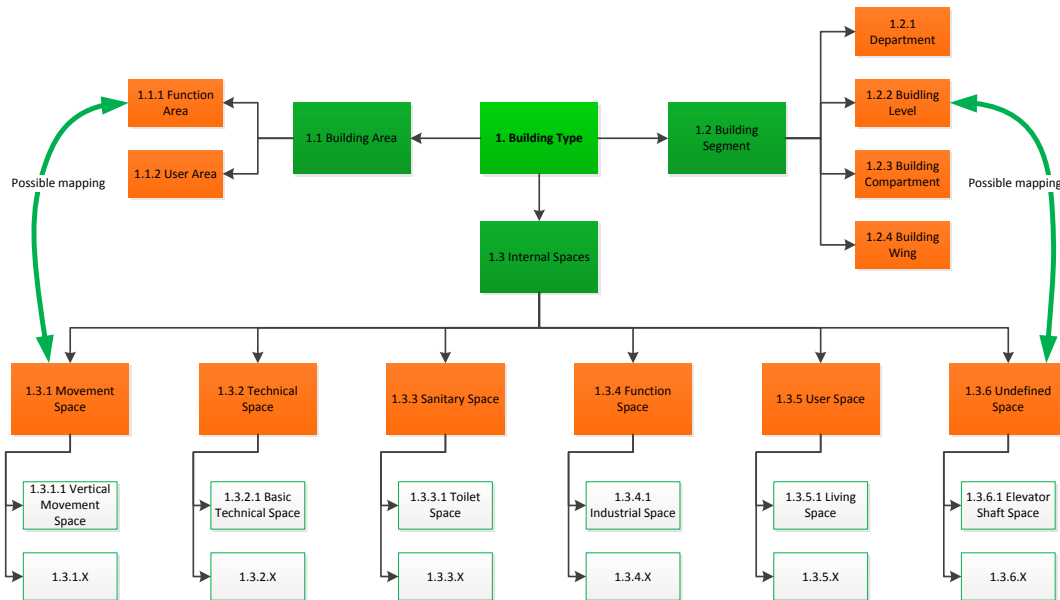


Figure 25: Space Hierarchy without subclasses

### 5.3.1.2.2 Object Classes

Aside from spaces also the objects need to be defined in the BAM Object Type Library (BAM OTL). Various classifications exist for the definition of objects. Aside from classifications also the modelling applications define the various types of objects. Using a consistent classification of elements is essential for creating useable data. Without consistency, using the data for other applications will be useless as the information will be incorrect or not manageable.

In the Netherlands the element classification NL-Sfb<sup>7</sup> has been defined by the Dutch association of architects (BNA). This classification is built up as a coding system where the varying elements are defined with the use of a number code (BNA, 2005). For example an internal non-constructive wall is defined as “(22.1) Internal walls; Non-constructive”. The definition of this classification is only worth as much if the classification is complete. The NL-Sfb classification is a classification defined in documents. This makes the usage not open for data. In the current practice within BAM, classification of elements during modelling happens with the use attributes. The reusability of this way of modelling is not high as a manual action of attaching the classification towards an element is needed. The usage of an object type library based on the semantic web standard makes it possible to create reusable links between modelled elements and the classification. The total list of created objects in the BAM OTL is stated in annex G.

### 5.3.1.2.3 Properties

The various types of properties can be used to describe the quality of the objects or spaces. This interaction is described in Figure 20. To make the properties usable, the right properties need to be obtained. The IFC schema has defined basic property sets for all elements. For example for door elements the property set “Wall common” exists (pSet\_WallCommon). In this property set ten properties have been defined. With the use of these properties various characteristics of a wall can be described. The IFC schema Property Sets don't describe all the required properties of a building. There are still properties which are described by a client in

<sup>7</sup> <http://www.stabu.org/diensten/nlsfb/>

the requirements which aren't possible to be defined in BIM models when using IFC. To see if these requirements of other properties are met happens with the selection of the products. The definition of which properties are going to be used depends on the input of the requirements. In this thesis a selection will be made to represent a variety of requirement types and constraints. This selection will define the to be used properties and is elaborated upon in 5.3.2.3.

### 5.3.1.3 Constraint templates

From the definition of the various types of requirements (sections 3.2.3, 4.3.2.2) and the analysis of requirement databases (section 5.3.1.1), the constraint templates have been defined. These templates are defined according to the basic structure of a requirement which is stated in Figure 20 (section 5.2). The completeness of this list of constraint templates defines how usable the total requirement checker is. When not all requirements are described with the use of these templates, the checker will be less valuable. The total list of constraint templates is stated in Table 9.

These constraint templates will need to be created in a way that enables the right objects, spaces and properties to be filled in the variables of the template. These elements are described already in the section 5.3.1.2. The creation of the templates is done with the use of SPIN. The use of a constraint violation in SPIN gives the possibility to build a statement that defines a violation of a rule. This rule is applied to the data set and brings forward the non-complying elements of a requirement. These constraint templates will make use of the BAM Object Type library to fill in the variables.

**Table 9: Constraint templates**

Constraint template	Description	Example
Type existence	An element type X must exist in a model	The building must have an elevator
Property Existence	A property Y an object must exist	All doors in the building must be self-closing
Cardinality	There should only be 1 element existing in a certain location	An user space should only be part of 1 fire compartment
Value	A property Y of an object X must have a value Z	All walls must have an acoustic rating of at least 65db
Space type existence	A space type X must exist in a model	The building must have a disabled toilet
Space type amount	The building must have at least X space type Y	The building must have 2 disabled toilets
Space type value	All spaces of space type X must have the value Y	All user spaces must have a net area of at least 6m <sup>2</sup>
Space – Space Relation	A space type X must be adjacent to a space type Y	An office space must be adjacent to a movement space
Space - object Containment	A space type X must contain an object Y	Every user space must contain a power socket
Space - object Containment amount	A space type X must contain Z objects Y	Every user space must contain at least 2 power sockets
Space - object relation; property	An object Y contained in space type X must have the property Z	Every desk in an office space must be adjustable in height
Space – Object relation: Value	An object X with property Z in Space type Y must have value A	A desk light in an office space must have a illuminance rating of at least x
Spaces - wall relation; value	A wall X with property A between spaces Y & Z must have value A	A wall between a user and a movement space must have an acoustic rating of at least 42dB
Spaces - object relation; value	An object X in a wall with property A between spaces Y & Z must have value A	A door in a wall between a user space and a movement space must have a Fire rating 3.

An example for a template is shown seen in Listing 4 and Listing 5. In this example the constraint template “Space – object relation; value” is used. This template needs to bring forward an object in a space. This object must have a property which has a certain value. This value is subjected to a certain operator which defines if the requirement is complying or not. The basic template can be seen in Listing 4 and a filled in example in Listing 5. With the use of Space classes, object classes and properties all the templates are made reusable. The usage of these templates is defined by the application and needs to be connected to the process of verifying the requirements. The total overview of all the SPIN construct statements for the constraint types can be seen in Annex F.

**Listing 4: Constraint Template Space-object relation: value**

```

CONSTRUCT {
  _:b0 a spin:ConstraintViolation .
  _:b0 spin:violationRoot ?s .
  _:b0 spin:violationLevel spin:Warning .
}
WHERE {
  ?s a bam:ObjectClass .
  ?s pset:isBoundaryOf ?o
  ?o a bam:SpaceClass .
  FILTER NOT EXISTS {
    ?s ?hasproperty ?w .
    FILTER operator(?w, ?a) .
  } .
}

```

**Listing 5: Constraint template example**

```

CONSTRUCT {
  _:b0 a spin:ConstraintViolation .
  _:b0 spin:violationRoot ?s .
  _:b0 spin:violationLevel spin:Warning .
}
WHERE {
  ?s a bam:Wall .
  ?s pset:isBoundaryOf ?o .
  ?o a bam:OfficeSpace .
  FILTER NOT EXISTS {
    bam:Wall pset:acousticRating ?o .
    FILTER >( ?o, 65) .
  } .
}

```

From the requirements data analysis and the definition of the constraint templates, a selection of requirements is now made for developing the prototype. These requirements are selected to represent the variety of requirements which is found in the requirements data analysis. Also taken into account with the selection is the answer of the interview question which requirements are most suitable or most interesting for automation (chapter 4). A selection of 10 requirements is made for further development of the requirements checker. These requirements are stated in a way that the performance or the relations are measurable. These requirements are stated in Table 10. The requirements which are selected are restricted to spaces and walls as this is the scope of this research. The elements which need to be defined in the BAM object type library are also stated in this overview. An extended overview of the selected requirements with the applicable querying can be found in annex E.

**Table 10: Selected requirements for prototype**

#	Need	Description	Used template
1	Comfort <i>Acoustic comfort</i>	<i>Value checking of a wall between two different spaces</i> <b>A wall between a User and a movement space must have a maximum acoustic rating of 65 dB</b>	Spaces – wall - object relation; value
2	Modelling quality <i>Well formed</i>	Data quality checking of objects; Classification coding <b>A wall must have the correct NL-SfB code (22.11)</b>	Value
3	Safety <i>Fire safety</i>	Material checking; Combustibility <b>A fire separation wall must be made of a material Non-Combustible</b>	Property existence
4	Spatiality <i>Area</i>	Gross area checking of space types <b>The gross area of an office type X must be at least 24m2</b>	Space type value
5	Aesthetics <i>Size</i>	Checking of size of objects in a wall in a certain space <b>A window in an external wall in an office space must have a size of 1000mm</b>	Space – object relation; value
6	Functionality <i>Headroom</i>	Wall Height Checking <b>Internal walls must be at least 2500mm high</b>	Value
7	Availability <i>Object Occurrence</i>	Object occurrence in a space checking <b>Every unique user space must contain a smoke detector</b>	Spaces – object Containment
8	Availability <i>Object quantity</i>	Amount of objects in a space checking <b>Every unique user space must contain at least 3 power sockets</b>	Spaces – object Containment amount
9	Safety <i>Fire Safety</i>	Value checking of an object in a certain Space type <b>The walls in an office space must have the property Fire resistance class</b>	Space – object relation; property
10	Costs <i>Unit costs</i>	Cost requirements checking <b>Internal walls metal stud must not exceed the cost of €X,- per m2</b>	Value

### 5.3.2 Building model preparation

The definition of how the requirements are defined and how they are going to be checked is defined in the rule preparation phase of rule checker development (section 5.3.1). In the building model preparation phase, the right elements are going to be made available. For this step, the proposed layers of section 5.2 are given meaning by defining the elements which are used in the checker. These elements are defined by the selected requirements stated in Table 10. Aside from this also the IFC model used for validation is made available to be checked upon by converting it into the semantic web standard RDF.

#### 5.3.2.1 IfcOWL ontology

For using the IFC schema in a semantic web environment, a translation of IFC into an ontology is needed to reference the right elements in the IFC schema. This translation of the IFC schema has been proposed by the various researches. Beetz has proposed a conversion of the IFC schema into an OWL representation (Beetz, 2009a). This research has been used for further development of an IfcOWL schema. This has resulted in the development of the IfcOWL schema which has been adopted by the BuildingSmart Alliance<sup>8</sup>. The IfcOWL ontology of the BuildingSmart Alliance is used to create the inference rules between the BAM OTL and the IfcOWL ontology.

#### 5.3.2.2 IFC to RDF Conversion

To make an IFC available for querying, the conversion of IFC to RDF must be executed. This enables the relations between the various elements in the IFC schema to be queried upon. This is done with the use of the IfcOWL ontology. Pauwels et al. have developed a convertor<sup>9</sup> from IFC to RDF which uses the IfcOWL ontology. This conversion will create an instantiation of every IFC element in an IFC file to an IfcOWL instance. The elements with the according relations are described as a set of triples in an Object – Predicate – Subject form. A converted IFC file will then be able to be used for querying with SPARQL.

#### 5.3.2.3 Inference rules

The link between the described objects in the library of natural language elements for the user get meaning by creating inference rules. Inference rules define the relation between the objects in the BAM OTL and the IfcOWL ontology. This will enable that an IFC file is readable for a user. By using the IfcOWL ontology, the relationship between the objects in the library and the ontology will remain reusable. A converted file is possible to be queried upon in the concepts which are defined in natural language with these inferences.

These inference rules are developed with the following steps. A concept in natural language is defined by the AEC domain experts for the OTL. For example the concept external wall can be described with logic reasoning as a wall object with the property External. To define this with the IfcOWL ontology, the right elements in the IFC schema need to be found to describe the object and the property. In IFC there are two definitions of a wall, an IfcWall and an IfcWallStandardCase. These two concepts in IFC need to be connected to the natural language concept internal wall. Aside from the objects also the property must be defined that it is internal. This is done by the IFC Property set of walls; IfcWall\_PsetCommon 'isExternal'. If this property is defined as true, the external wall is defined.

Listing 6: Inference rule for bam:FunctionalSpace

```
CONSTRUCT {
  ?s a bam:FunctionalSpace . }
WHERE {
  ?s a ifcowl:IfcSpace .
  ?s ifcowl:longName_
    IfcSpatialStructureElement ?y.
  ?y a ifcowl:IfcLabel .
  ?y express:hasString
    "SportsSpace" .
}
```

Listing 7: inference Rule for bam:InternalWall

```
CONSTRUCT {
  ?s a bam:InternalWall . }
WHERE {
  ?s a/rdfs:subClassOf* ifcowl:IfcWall .
  ?s pset:isExternal true .
}
```

<sup>8</sup> <http://www.buildingsmart-tech.org/future/linked-data/ifcowl>

<sup>9</sup> <https://github.com/mmlab/IFC-to-RDF-converter>

By creating an inference rule with SPARQL, a reusable statement is defined to connect the OTL and the IfcOWL ontology. This statement is defined by creating a Spin:Rule in Topbraid. An example for a space class is shown in Listing 6. An example for a wall class is shown in Listing 7. In this way all the natural language concepts are given meaning in the OTL. This enables the checker to be executed without knowledge in data schemas. The total overview of inference rules is given in Annex H. The total of these inference rules is made available in the BAM ontology. This ontology is built up for all the classes which are needed for the requirements checker. This ontology should be used eventually for all elements which can be occurring in a building model. The completeness of this ontology defines how useful it can be for checking as missing elements will cause problems in using the checker. For the purpose of investigating the implications of automation in this research, a scope has been given to look upon. Therefore only the wall, door and window object classes have been defined as well as the spaces. An overview of the ontology can be seen in Figure 26.

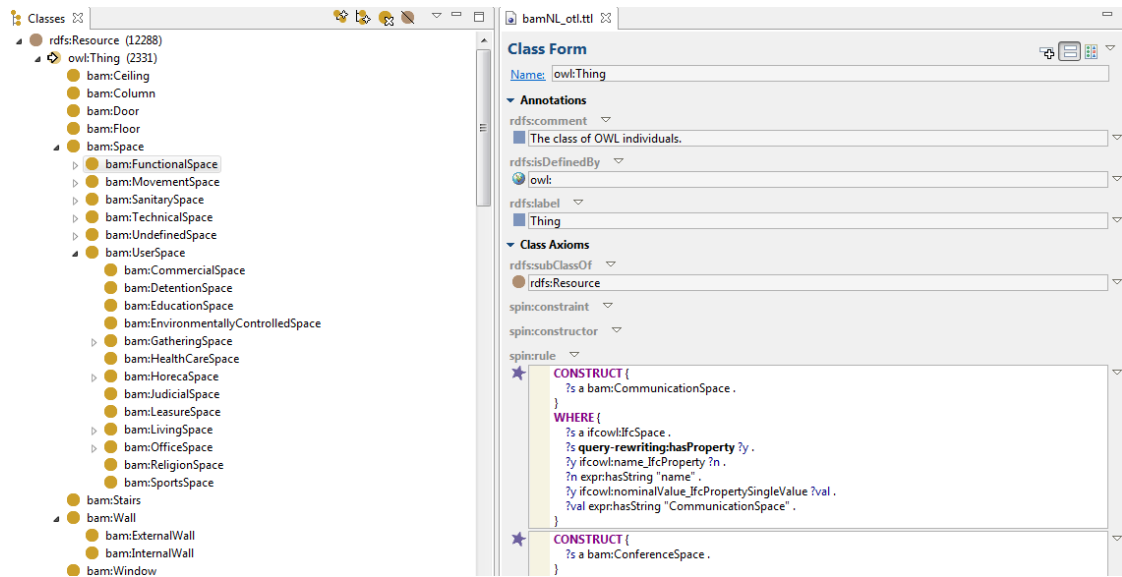


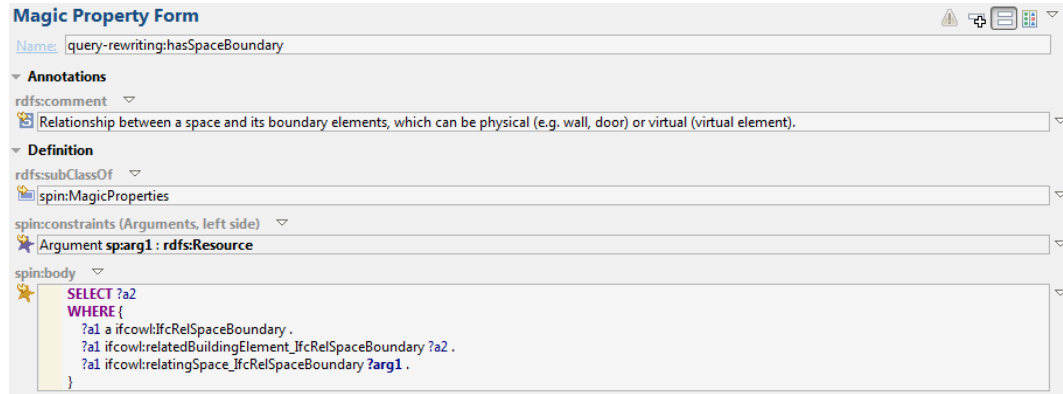
Figure 26: BAM Ontology in Topbraid Composer

Aside from the objects which are defined in the ontology, also the properties which are going to be checked need to be defined. These need to be made available for the requirements which are going to be queried. The requirements which are selected (Table 10) define which properties are created for the prototype. A complete overview of all possible properties of all elements in the ontology will be needed to create a complete prototype. The definition of the used properties is stated in Table 11. Here also a description is given about how they are defined in the data.

Table 11: Property definition

Property	Explanation	Source
Space Containment	<i>An element is contained in a space</i>	IFC:IfcRelContainedInSpatialStructure
Acoustic Rating	<i>An element has an acoustic rating</i>	IFC: Pset_AcousticRating
Classification Code	<i>An element has a Classification code</i>	IfcClassification
Combustibility	<i>An element must have the property combustible set to true or false</i>	IFC: Pset_Combustable
Gross area	<i>A space has a gross area of X</i>	IFC: Pset_Space GrossAreaPlanned
Placement in an element	<i>An element is placed in an element</i>	IFC: IfcRelVoidsElement & IfcRelFillsElement
Size	<i>An element has a certain size</i>	IFC: Pset_Size
Height	<i>An element has a certain height</i>	IFC: Pset_Height
Product containment	<i>An element exists in a space</i>	IFC:IfcRelContainedInSpatialStructure
Fire rating	<i>An element has a fire rating</i>	IFC: Pset_FireRating
Material	<i>An element has a certain material</i>	IFC: Pset_Material
Costs	<i>An element has a certain cost</i>	External Referencing

These properties need to be made available to be used in the checker. For accessing these properties a query upon IfcOWL is made. This is done with the use of magic properties. A magic property is a property function in SPIN that binds the variables on the left or the right side of the predicate (Knublauch, 2011). This enables the possibility of simplifying the queries which need to be built up. An example of one of these magic properties is stated in Figure 27. The complete overview of the magic properties used is stated in annex H.



**Magic Property Form**

Name: query-rewriting:hasSpaceBoundary

Annotations

rdfs:comment Relationship between a space and its boundary elements, which can be physical (e.g. wall, door) or virtual (virtual element).

Definition

rdfs:subClassOf spin:MagicProperties

spin:constraints (Arguments, left side)

Argument sparg1: rdfs:Resource

spin:body

```
SELECT ?a2
WHERE {
  ?a1 a ifcowl:IfcRelSpaceBoundary .
  ?a1 ifcowl:relatedBuildingElement_IfcRelSpaceBoundary ?a2 .
  ?a1 ifcowl:relatingSpace_IfcRelSpaceBoundary ?arg1 .
}
```

Figure 27: Magic Property Example as displayed in Topbraid Composer

#### 5.3.2.4 Data Quality

The way the data is modelled in the native software defines the data quality. Proper quality of BIM data is essential for using it for other purposes. Automated verification uses this BIM data and relies on the correctness and quality of the data. A data check is therefore needed to ensure that a check will bring the correct result. If the data is incorrect, the check will be useless. The investigation in data quality has been researched upon greatly. Also within AEC sector correct modelling has been identified as one of the crucial parts of proper implementation of BIM. For this research, the assumption is made that the quality of a model is appropriate. This assumption is one of the most important implications of using BIM data for automated verification. When the alignment of information in a requirement, in objects is done perfectly and the system of checking works flawless, the result of a check will still be useless if the data is modelled incorrectly.

#### 5.3.3 Rule execution

All the elements for executing the requirements checker are now defined. The next task is to build up the rule execution. In this execution all the elements which are defined are brought together as a functioning system. For the usage of the total system the basic structure in Figure 20 is used. This gives the following process;

1. Select the file which needs to be converted and checked
2. Select the constraint template according to the to be checked requirement
3. Select the right elements out of the bam OTL for filling in the template
4. Defining possible needed values according to the requirement
5. Execute the filled in template
6. Report the non-complying results

These steps define the total execution of the requirements checker. These steps are translated in a flowchart which is stated in Annex J. The way these steps are executed is by developing an interface which makes it possible to use all the elements which are created. This interface has the following aspects which need to be dealt with;

- Openness in usage of the interface and correctness of usage
- Alignment with the verification process
- Interlinkage with the requirements database

The openness in usage of the interface determines how free a user is in selecting templates and filling in the variables. For the academic world, the openness of the checker should be high to show the possibilities of the



checker. On the other hand there is the use of the checker by the end user in a construction project. The end-user will require a certain amount of freedom in use but also needs to be guided in the using the templates correctly. The definition of the verification process therefor defines how an end user will use the interface and what level of openness the end user requires.

The alignment with the verification process determines in what way a selection is made for a certain template. The starting point in verification is the requirements. The person, who executes the verification, looks into what the requirement actually is stating. From a requirement a certain question is asked to see if the design complies. Therefor the interface should also start with defining the question of what requirement is going to be checked. This will mean that not a template is selected but a requirement type is selected. The meaning of the requirement will determine what template is used. The requirement describes what kind of question is asked upon in the data. For example the need for comfort is explained by the various aspect requirements. A requirement about the acoustics of a room is determined by the value of the acoustic rating of the elements between various space types. In this statement already a property, a value and a space type is brought forward by the requirement. This determines which template is brought forward. To complete this intuitive selection path, the accompanying properties is connected to the selected need. In this way a selection path is created to eventually select the correct template. In this way the two defined structures of a requirement are used to define the to be used template with the correct elements.

With the questions about the built up of a requirement, the selected need and the accompanying properties the correct template is selected. This way of automated selection of a template prevents selecting the wrong template. Together with the selection of which objects and spaces, also the filling in of the templates is taken care of. To make sure that the right path is selected, a flowchart is proposed in Annex M to define what the selection path should be to access the right template. This flowchart is followed in a conceptual way in the interface. To prove the working of this idea an interface is created which follows this concept. This interface will only contain a few elements to illustrate the functioning of the system. A mock-up interface will therefore be created.

Lastly the interlinkage with the requirements database needs to be taken care for before creating the requirements checker. As the requirements are the starting point of the check, a link between the requirements database (in this case the Relatics environment) and the checker would be favorable. This would mean that when a requirement is going to be verified in the design to see if it is complying, the starting point would be the requirement in the database. In this way an already filled in version of a template could be connected to the requirements database.

The proposition which is made here, defines the working behind the checking process. This is a proposed implementation of the total system in the current workflow. This isn't required for the prototype. Therefor this part is only discussed conceptually. The steps in Listing 8 should be taken if the requirements database is the starting point.

#### **Listing 8: Workflow for checking process**

1. Define requirement type in requirements database
2. Define elements of requirement in database
3. Link elements and requirement type to input for Checker
4. Execute check
5. Write or link report back to requirements database

##### **5.3.3.1 Built up of the interface**

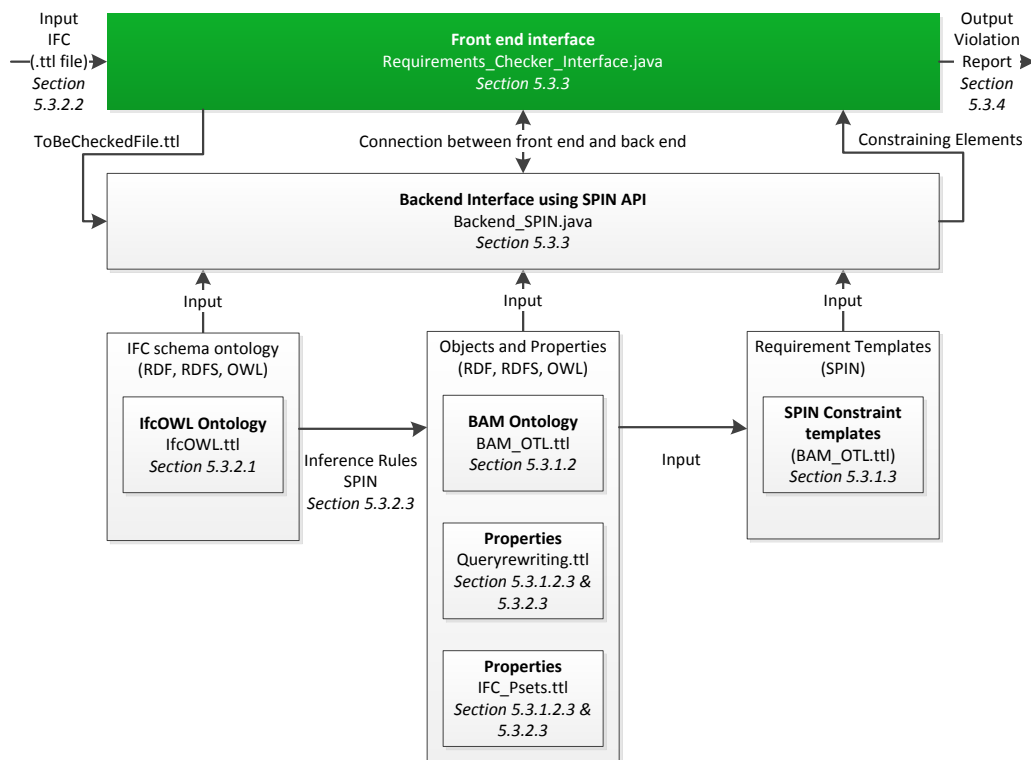
In the interface two requirements are used to prove the concept of the requirements checker. These requirements are brought forward by following the steps as proposed for the interface. The functionality of the interface must prove that requirements can be checked with the use of automated checking. Here for only the filled in queries are used as building up the modularized total system is already discussed upon in previous sections.



For developing the interface the following steps must be done;

1. Defining the steps in the execution
2. Making the backend checking process available with the SPIN API  
Input of BAM OTL, IfcOWL & Constraint templates
3. Making the inference rules available for the constraint checking
4. Making the constraint violation rules available (selected requirements only)
5. Test backend
6. Creating the outlook of the interface
7. Give the front end meaning by creating a link with the back end
8. Creating the report generation

The interface will use the parts which are developed in section 5.3.1 and 5.3.2 in the backend. The functioning of these parts in the total interface is illustrated in Figure 28.



**Figure 28: Overview Requirements Checker application**

The development of this checker is done with the use of the SPIN java application programming interface (API)<sup>10</sup>. The interface depends on the availability of the IfcOWL ontology, the needed objects, spaces and properties, the inference rules and the requirement templates. These are already defined in section 5.3.1 and 5.3.2 and are made available with the use of the SPIN API.

After these elements are made available in the interface, the input for the check is taken care of. The input for the check is an IFC file which is converted to RDF. This conversion could be integrated in a more developed interface. In this prototype interface a conversion is done beforehand. This is done with the IFC-to-RDF convertor of Pauwels et al. To use the converted file a button is developed in the interface to select the path of the file to load the file in to memory.

As the interface will remain a mockup to prove the concept, already filled in templates are used to make the rules executable. The rules which are used in the interface are stated in Listing 9 and Listing 10. These are elaborated upon in annex I.

<sup>10</sup> <http://topbraid.org/spin/api/>

**Listing 9: Filled in Constraint template for acoustic comfort of an internal wall**

Comfort *Value checking of a wall between two different spaces*  
 Acoustic **A wall between a User and a movement space must have a minimum**  
 comfort **acoustic rating of 65 dB**

```
CONSTRUCT {
    _:b0 a spin:ConstraintViolation .
    _:b0 spin:violationRoot ?s .
    _:b0 spin:violationLevel spin:Warning .
}
WHERE {
    ?s a bam:internalWall .
    ?s qrw:isBoundaryOf ?o .
    ?s qrw:isBoundaryOf ?o2 .
    ?o a bam:OfficeSpace .
    ?o2 a bam:HorizontalMovementSpace .
    FILTER NOT EXISTS {
        ?s pset:acousticRating ?o3 .
        FILTER (?o3 < 65) .
    } .
}
```

**Listing 10: Filled in Constraint Template for Power Socket Containment in a office space**

Availability *Object containment in a space checking*  
 Object **Every unique user space must contain a power socket**  
 Occurrence

```
CONSTRUCT {
    _:b0 a spin:ConstraintViolation .
    _:b0 spin:violationRoot ?s .
    _:b0 spin:violationLevel spin:Warning .
}
WHERE {
    ?s a bam:OfficeSpace .
    FILTER NOT EXISTS {
        ?s qrw:hasContainedProduct ?o .
        ?o a Bam:PowerSocket .
    } .
}
```

These rules are made available in the script of the tool by addressing their identification which is created in the BAM ontology as Constraint\_acoustic\_3 & constraint\_powersocket\_1. This is attained in the script and is stated in Listing 11.

**Listing 11: Code for accessing constraint templates**

```
List<Resource> constraints=new ArrayList<Resource>();
Resource constraint1=backend.spin.getResource(
"http://www.bam.com/bamNL_otl#constraint_acoustic_3");
Resource constraint2=backend.spin.getResource(
"http://www.bam.com/bamNL_otl#constraint_powersocket_1");
constraints.add(constraint1);
constraints.add(constraint2);
hashmap.put("Comfort",constraints);
hashmap.put("availability",constraints);
SPINModuleRegistry.get().init();
SPINModuleRegistry.get().registerAll(backend.spin , null);
```

The connection between the interface and these rules is created by the steps which are followed in selecting the right template. As the selected requirements are linked to a need and a requirement type, they must be connected to these selection buttons. This link of bringing up the right template should be modularized in a more developed interface. In this thesis, this is done by connecting the selection of the need to the pre-defined and filled in constraint templates. In this way for example when the button comfort is selected, the filled in constraint template for checking a wall on the acoustic rating is brought forward.

These elements which are now made executionable define the working of the system. Aside from these needed elements also the interface must be working. Here for java WindowBuilder<sup>11</sup> is used to create the interface which is available for the end user. An overview of the windows can be seen in Figure 29.

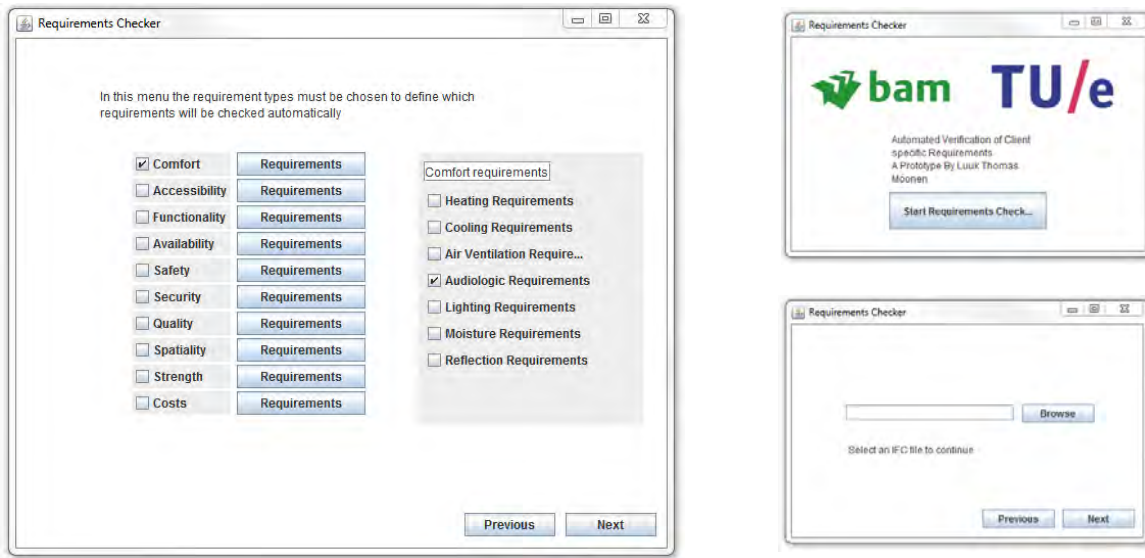


Figure 29: Overview of windows of the requirements checker

The windows of the complete interface are stated in annex L. The total script can be found in in Annex K.

#### 5.3.4 Rule Reporting

When the rules are made available, the execution of the checks can be done. This will bring the elements forward in the model which violate the constraint templates. These need to be retraceable for improving the model after the check. Here for the results of the constraint violations will need an identifier which is useable for the modeler. Here for a representation must be made in a viewer or an identifier must be added to make the element retraceable after a report is generated.

The generation of a report of violating elements is the output of the checker. In the prototype the constraining elements are stated in a turtle file. An example of the outcome can be seen in Listing 12.

#### Listing 12: Partial output of checking

```
[ a                                spin:ConstraintViolation ;
  spin:violationLevel  spin:Warning ;
  spin:violationRoot
<http://linkedbuildingdata.net/ifc/resources20161017_142426/IfcWallStandardCase_54898>
] .

[ a                                spin:ConstraintViolation ;
  spin:violationLevel  spin:Warning ;
  spin:violationRoot
<http://linkedbuildingdata.net/ifc/resources20161017_142426/IfcWallStandardCase_54946>
] .

[ a                                spin:ConstraintViolation ;
  spin:violationLevel  spin:Warning ;
  spin:violationRoot
<http://linkedbuildingdata.net/ifc/resources20161017_142426/IfcWallStandardCase_23032>
] .
```

<sup>11</sup> <https://eclipse.org/windowbuilder/>

In this outcome the URI's are given to elements which aren't complying. From this URI, the element can be found. This makes the process of design improvement possible. The reference to all information of an element can be found via these URI's. For example a GUID can be found to trace the element back in a viewer.

This outcome is just listed as the elements which aren't complying. To give further meaning to this outcome, comparing this outcome to the total amount of objects gives an indication on how well a design is built up. In this way a percentage of elements which complies can be given. In the use case validation this is shown in a real example.

Another way to report violating objects is to enable the usage of a viewer. This can be enabled by exporting the report as a BIM Collaboration Format (BCF) file. IFC viewers can visualize these reports into an overview of the elements which are constraining in a 3D representation. Enabling this viewer would require extensive additional interface programming to link the data of the URI's to geometrical objects. A representation is therefor made with the use of Solibri model viewer to show which elements are not complying. This is shown in Figure 30. The selected elements in green are the non-complying elements.

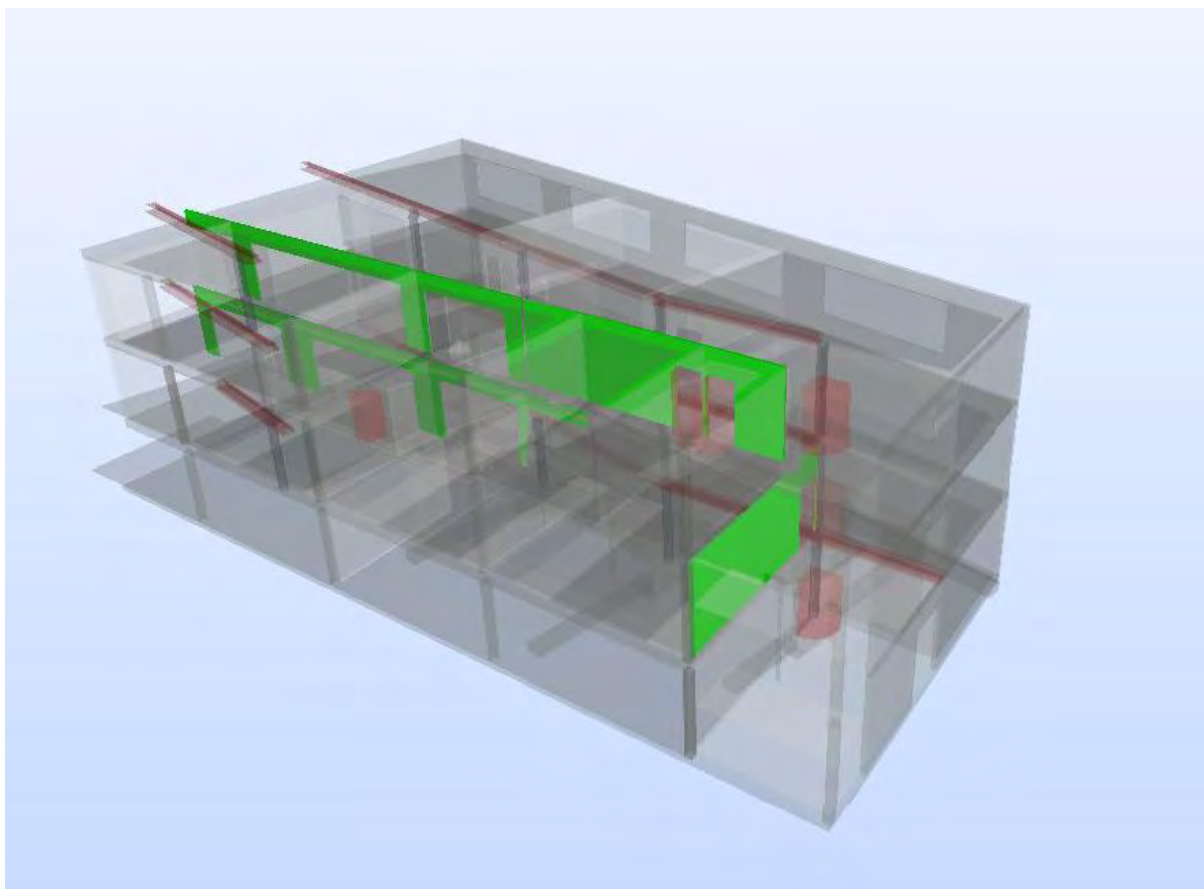


Figure 30: Visualization of checking report

## 5.4 Use case validation

To evaluate the usage of the requirements checker for client specific requirements, the checker is used for evaluating a building model of an existing project. This is project 4 of the data analysis. From this model, a part of the building is exported as an IFC file. A visual representation of this partial model can be seen in Figure 31.

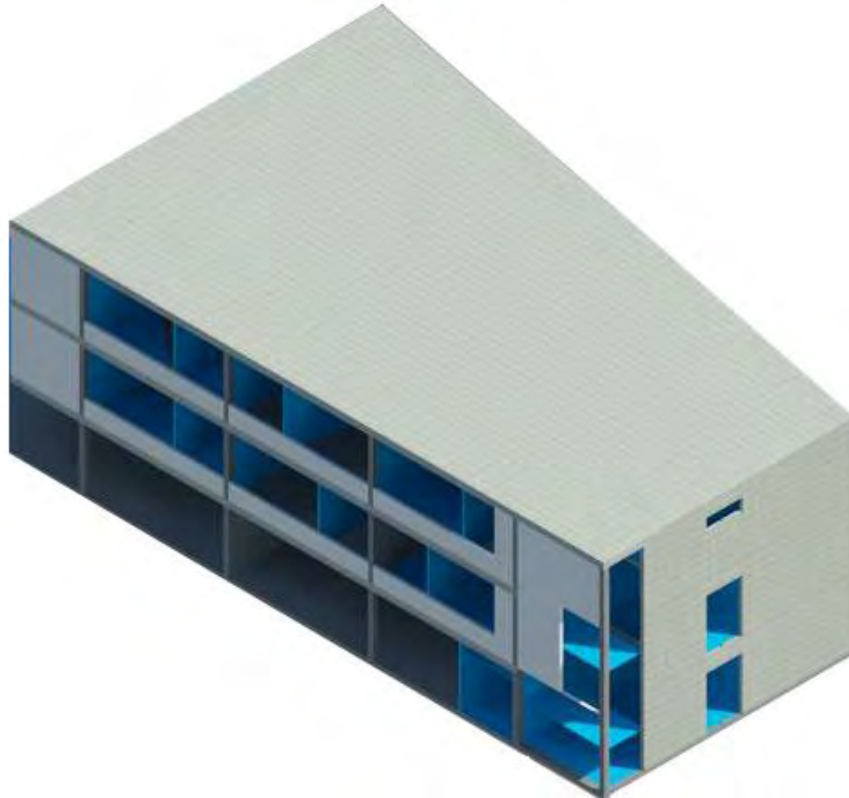


Figure 31: Project 4 model

The data of this model has been improved to ensure its usability for the requirements checker. This data quality improvement focusses on the classification of spaces. The uniform space classification which has been stated in section 5.3.1.2.1 has been used to make the spaces identifiable for the checker. Also the performance of the to be checked elements has been looked upon to ensure that a check can be executed.

The model is converted to RDF and used in the checker. This gives the result stated in Listing 13. Here it can be seen that 4 internal walls aren't complying with the requirement regarding acoustic comfort as stated in Listing 9. The total amount of internal walls between a meeting space and a horizontal movement space in this model is 12.

### Listing 13: Outcome of check for acoustic Comfort

```
[ a          spin:ConstraintViolation ;
  spin:violationLevel  spin:Warning ;
  spin:violationRoot
    <http://linkedbuildingdata.net/ifc/resources20161017_142426/IfcWallStandardCase_54898>
    <http://linkedbuildingdata.net/ifc/resources20161017_142426/IfcWallStandardCase_54946>
    <http://linkedbuildingdata.net/ifc/resources20161017_142426/IfcWallStandardCase_23032>
    <http://linkedbuildingdata.net/ifc/resources20161017_142426/IfcWallStandardCase_51465>
] .
```

In this way it shown that the model checker works upon a real existing project when minor adjustments are executed. The representation of a viewer shown in Figure 30 shows the outcome of listing 13 translated into a visual representation.

## 5.5 Discussion

In this thesis a prototype has been developed for a requirements checker based upon the semantic web standard. A standardized set of modularized constraint templates has been developed to describe the variety of requirements which can be found in a project. These constraint templates are based upon the literature review in chapter 3 and a data analysis on the requirements data of five projects of the Construction Company BAM in section 5.3.1.1. In this way a modularized set of constraint templates is created which can be filled in by the elements and their properties of a building. These elements are defined by the Object Type Library which has been proposed for BAM. In this way a library is created which can be reused as it is built up as an ontology which can be easily accessed through the web.

The prototype has delivered an easy to use and open way of accessing and assessing the data in a model towards requirements of the client. The usability of the checker reduces the need for understanding the complete data structure of the IFC schema. This makes the usage of IFC easier and less error prone.

The execution of the checker with the use case has shown that non complying elements can be brought forward if the IFC data is valid. For certain predefined requirements, the usage of this checker will already be usable as the data is described in the building model as well as in the requirements database in a similar way. These requirements have been identified in the data analysis of the requirements databases (section 5.3.1.1). The usage of this checker can reduce the amount of errors towards requirements and also can reduce the time it takes to check these kinds of requirements.

The usage of inference rules has made it possible to define elements in the complex IFC schema with the usage of natural language. This increases the understandability of building model data and for execution of these checks the need for employees with skills in data structure and the construction domain is reduced. The knowledge about accessing the right data in IFC data is needed once in the creation of the inference rules. After the creation of the Inference rules, the rules can be reused. Only for maintenance and improvement of the checker and the Object Type Library the combination of skills is needed.

The semantic web standard has been identified as a suitable way of describing the required relations proposed by requirements of the client. Triples are built up in certain semantic structure. When using this way of building up requirements, the person who defines the requirement in the database is compelled to define the requirement in a more measurable way. For example the statement “every user space must have at least 1 power socket” is very clear and translatable into a triple statement in comparison too “User spaces must have enough electrical appliances”. The usage of the semantic web therefor compels to define clear to measure triple statements. This reduced ambiguity can decrease misinterpretation and the thereby coming mistakes in the design process.

Using the semantic web standard for describing building model data, enables linking additional types of data towards the elements in a design. For example a cost database can easily be linked to the elements in the Object Type Library (OTL). A link between an element in IFC can then easily be linked to separate RDF data about element cost. Only a relation between the elements in the cost database and the model must be made. This opens up possibilities for adding knowledge easily to the OTL.

Aside from the benefits the implementation and development of the requirements checker has brought forward shortcomings and limitations. These can be categorized in issues in;

- Data quality
- Requirements definition
- Application development and usage

Firstly the most general way the requirements checker can be restricted in, is in the input data. The check upon IFC data is only useful if the IFC data is valid. This issue also is identified in the theoretical background (section 3.4). For the exports which have been used in the testing and validation issues have been identified in this data quality. If certain relations don't exist in the IFC data, a query will not be valid as the triple isn't



created in the conversion. For example if the relations between elements and spaces aren't existing, a check will not produce a valid answer.

Aside from existing relations, also the way an IFC is built up can vary. Describing objects and their properties can be done in various ways as defined in section 3.3.4.2. The validity of an IFC is there for difficult to measure. In the development of the checker different ways of defining data can be found. For example the way a certain space is given a space type definition in the data defines how an inference is created. Uniform and standardized ways of creating IFC's is needed to ensure the validity of the data. The methods used within BAM already prescribe standard ways of exporting Revit<sup>12</sup> data to IFC to overcome these issues but the execution can easily be done incorrect. The importance of data quality checking and standardization of defining data comes forward here clearly to ensure the usability of the requirements checker.

Another way the data consistency determines the quality of the requirement checker is in the way spaces are currently defined in a model. Mostly requirements are firstly built up as requirements for a certain space (Figure 21). The definition of spaces therefor defines how easily the objects in a certain spaces are accessed through a query. This definition closely relates to the built up of a majority of the constraint templates as the relation between a space and an object is defined. In the current projects a large variation in space definition is found. This variation complicates the way a space is accessed in checking. To access these spaces in every model in the same way, a standardized way of defining space types is necessary to ensure the functionality of the requirements checker. This has been proposed in section 5.3.1.2.1 but will require further development. If this development is done as a nationwide standardization, the effect will be much greater in comparison to a company level as an interpretation of the company will still be required to define the space types.

Besides data quality also the definition of requirements are influential on the functioning of the requirements checker. For making the requirements checker executionable, the requirements which are used have been improved on their semantics (section 5.3.1.1). The only way requirements can be checked is if the definition in the statement is iterated into a measurable equation. The elements which must be present in this equation must be equal in the data in requirements and in the data of the design. Therefore using this checker requires an improvement in working with requirements before checking will be beneficiary. This improvement focusses on how a performance is defined for requirement types. In the conclusion (Chapter 6) this improvement is elaborated upon.

When requirements are defined properly and a performance is available, an issue can arise in the way how to describe this performance in the data. The IFC schema already makes it possible to describe various types of elements and various properties, but the completeness of the data can produce problems which are difficult to overcome. The semantic web standard makes it possible to describe these properties or elements from a different data source. For example the possibility of accessing information about costs opens up big opportunities of using the semantic web standard for describing building models.

The checker also has a limitation in the execution in its current form. As the checker loads the files it uses into the memory of the computer. With small files this doesn't take large amounts of time, but with bigger models this may cause problems in use. This could be overcome to use a triple store to reduce the time it takes to execute the checks.

Lastly the requirement checker has also limitations in the application on itself. As the export of the results is not made visible in a viewer it is now difficult to see the results of the check. Here for extended development of the interface with a viewer should be executed. An export of the constraining elements towards BCF (BIM Collaboration Format) or IFC viewers should be made possible. An execution of SPARQL queries could be shown in the query viewer of Chi Zhang.

Aside from limitations in the possibilities of the requirements checker there are also limitations in the research itself. This research has been made manageable by defining a scope. As the scope of this research leaves out a variety of building elements, the applicability of the requirements checker can be questioned

---

<sup>12</sup> <http://www.autodesk.nl/products/revit-family/overview>



upon. This is mostly applicable to the system elements of mechanical and electrical installations as these are different in their built up in comparison with basic construction elements.

Also a limitation can be found in the development of the prototype. As this prototype is built up as a mock-up of the proposed system, the complete functioning of the modular system with the OTL and the Constraint templates is not tested. The functionality is therefore not completely tested and compared to the conventional verification process. Together with this comes the usability of IFC files and the made assumption of data quality. The quality of data is crucial for the checking of designs. The assumption made to test this prototype leaves out this element of uncertainty in real projects.

Another limitation which follows from the fact that the checker is a prototype comes forward from the proposition of modularity. The interface is built up as a mock-up. This has resulted in the fact that the interface is realized in a non-modular way. Therefore the selection of templates and filling in the templates isn't tested. This may bring forward issues in selecting the wrong templates or selecting the wrong elements. This can lead to invalid results of the queries.

Lastly a validation towards the end users hasn't been executed in this tool prototype. The validation in this research has evaluated the usability of the checker for real projects. Here the implications have been found in the data quality assurance. The applicability to the current verification process will only be questioned upon with the thesis presentation within the graduation company.

## 6. Conclusion

The goal of this research is investigating the implications of automated verification of client specific requirements using rule checking techniques and the semantic web standard. In the following pages the sub questions of this research are answered.

The theoretical background (chapter 3) of the design process has identified two crucial information streams in the subsequent phases of a construction project. Firstly there is the requirements information stream which follows from the client needs regarding a building. Secondly there is the object related information which is generated in the design of a building. The relation between a requirement and a design element is the exact area where verification takes place and the answers are given to translate the need of a client into a suitable building. Exactly in this area changes are occurring due to the implementation of integrated contracts. Integrated designs therefor ask for a different approach in managing information. In every step in the design process the iterations made must be related back to the original buildup of the as required system. This provides a connection to the original need of the client. The theoretical background has identified that a systems engineering approach closely aligns with these steps of iteration and the alignment between requirements and objects.

The qualitative research (chapter 4) has identified the main problems in the alignment of information in the design in the current processes of a construction company. These can be summarized as followed;

- The iteration steps from a need towards a performance are not followed properly which causes problems in allocating requirements to the right objects
- Standardization in naming spaces and objects is insufficient which makes it difficult to allocate requirements to these elements
- A lack in standardization in defining a requirement causes a great variation in definitions of requirements which causes high levels of interpretation
- Variation in level of detail in requirements make it difficult to make decisions on objects as the iteration level is unequal amongst applicable requirements

An important reason why these problems occur in the design phases can be traced back to the way a project is started. To clearly understand what the client requires an extensive requirement analysis is needed. The qualitative research has brought this forward as one of the main reasons why the information alignment isn't happening as it should be done. Together with the lack in standardization of requirements and the accompanying process, the inconsistency and variation causes a lot of rework in projects. A more standardized way in working with requirements and defining them is therefore researched upon with the development of the requirements checker.

Aside from the design process the theoretical background has given an insight in the current difficulties with rule checking applications and development. These difficulties have created the outlook for the requirements checker. The main conclusion which can be drawn up is that current automated rule checking efforts focus mainly on static hard coded execution of rules. This makes it difficult to apply to the flexibility of a requirements database and these checkers are not open for the laymen. The development of a rule checker and the creation of new rules are therefore only open for people with an expertise on data structures, programming and the construction domain. This combination makes the use of a rule checker less attractive and not open for the use in automated verification.

To create the rule checker, standardization and uniformity in requirements and information alignment has been researched upon. This has resulted in the evaluation of how requirements are built up. It is defined that the applicability and the type of a requirement can be standardized greatly when defining a performance. The steps of iteration from a need towards a performance are of great importance here, these steps are Goal - Need - (Space & Object) Requirement - Performance. When following these steps of iteration from need to performance clearly alongside the phases of a design, a more standardized way of defining requirements can be followed. These steps closely relate to the phases of development in a Systems engineering process and the increase in level of development alongside the subsequent phases in the design process.

The eventual iteration to what type of performance a certain requirement asks, defines what type a requirement is. In the model development a variety of requirement types is defined which correspond with various constraint templates. These requirement types are defined by the requirements data analysis which is done upon five existing projects of BAM. In this data analysis also an investigation is done into the applicability of a requirements checker. The result of this investigation shows that the percentage of requirements where a BIM can be used for verification is still low. Only an average of 12,2% of the total amount of requirements can be verified using a BIM. An increase in possible usage is clearly shown along the initiation years. This increase will make the value of a requirements checker using a BIM higher in the future.

The way a requirements database is built up and developed is closely related to how usable requirements are for checking with a BIM. The connection between requirements and objects in a BIM is made on a performance level at this moment. This is due to the fact that only the objects can give an answer to the requirements as spaces are non-tangible objects. The equality in information in requirements and objects is there for essential. In the design process defining a performance of an object must be focused upon more during requirements analysis.

In the development of the requirements checker, semantic web standards have been used to develop an open and reusable system. The theoretical background and the interviews have resulted in the pre conditions for this checker. From this starting point an architecture has been developed. This has resulted in a checker which is modular and able to describe a vast amount of requirements on a performance based level. This architecture has stated the use of an object type library (OTL). This object type library makes it possible to describe elements in a natural language which makes the understandability of the data higher. As this OTL is developed with the semantic web standard, the reusability is increased as the individual elements can be stored on the web instead of documents. Here for the semantic web standard has been very useful as the various types of information can be added and linked to the OTL.

This checker is developed as a mockup version which proves the concept which is described in the model development (Chapter 5). The mockup version has shown that the usage of the proposed architecture makes it possible to intuitively use the templates. These templates are defined according to the essence of a requirement. The essence of a requirement is what the performance is defined for (type existence, property, relation etc.). The result of the checker gives a list of elements which aren't complying towards requirements. The benefits of this checker can be found in the conclusive answer which is given about which elements aren't complying yet and the reduced time it takes in comparison with the conventional verification process.

The necessity of defining performance improves the way the requirements analysis is executed. This can be seen as a drawback as the checker only works if the data is defined to a certain level but it can also be seen as process improvement. This process improvement of defining performance makes standardization much easier as the variables in performance of objects remains always remain the same. For example a wall will always have an acoustic rating, a material type and many other performance indicators. Defining an Object Type Library where these performances are indicated already makes it easier to iterate the requirements of a client towards performance indicators. When these performances are always defined in a standardized way, the design can easily be verified on compliancy with requirements. This will initiate the possibility of not only using the checker for verification but for generating 100% complying designs.

Focusing on the specific execution of the rule checker, some conclusions can also be made. Firstly the requirements checker relies on the data quality of the input. Without valid input of properly exported and converted IFC files, a check will not be of any value. The manual way of verification which is now present relies on the knowledge of the user, the requirements checker relies on the quality of the data and on the completeness and up to dateness of the checker. A clear evaluation of the benefits of the checker is due to this reason difficult to measure as the maintenance and further development must be accounted for.

The requirements checker on itself has been tested is easy in use. This comes from the fact that the architecture which is proposed makes it possible that selecting the right templates and elements is made open towards the end user. The templates are used by the end user as an exercise where the blanks are filled

in with the use of the elements of the OTL. The interface will determine how easy to use the checker is. The prototype sketches this use but can't be put to the test extensively as it remains a mockup of the total system. The developed checker can improve the verification process provided that the initial process is more standardized and performance based. An eventual reduction in non-complying elements towards requirements in a BIM can then lead to a reduction in design errors and eventually costs.

In total can be concluded that the process improvements of standardization in defining requirements and performance is one of the most important improvements which is proposed in this research. This standardization in process management can improve the interaction between data in requirements and in the design. This improvement can lead to eventual reduction in failure costs and to a more efficient process.

Now that an overall conclusion is drawn up, the research question of this thesis can be answered. The research question is;

### **How can verification of client specific requirements be automated and improve the design process?**

The core of this automation lies mostly in the standardization of requirements and following the steps in iteration from need to performance. If this process is followed clearly and monitored closely, requirements and objects will be better allocated to each other and defined more clearly. A need for iterating requirements to better understandable non ambiguous statements with a certain obligation to define a statement where a "must" is present will make automation of verification possible.

The usage of the semantic web makes this improvement possible as the data elements are made available as unique identifiers which can be addressed and accessed by the checking application. This makes it possible to capture the knowledge about the elements in a building in an easy and retraceable way.

The main improvement of this automation lies in the standardization of defining requirements, the process of defining the associated performance and the way the elements are allocated and portrayed. This in total enables the checker to be executed. Here lies a big opportunity for the AEC sector to improve the understanding of the development of requirements and the accompanying information management which enables this process.

## 7. Recommendations

The recommendations resulting from this research can be divided in company & sector related recommendations and in an outlook for further research and development

### 7.1 Company Recommendations

The recommendations for the company can be seen as recommendations for AEC industry in working with requirements and aligning the information of BIM.

The first recommendation relates to the process management regarding requirements and the interlinkage between BIM and systems engineering. An improvement in requirements management is needed to clearly define the system of requirements allocated to objects. As defined in the research, the connection of an element in a 3D design tool and an element in a system design can only be made firstly on an element level. This implies that to address requirements also on this level of elements should be worked upon in buildings. Requirements should be allocated to the system design objects to make sure that the interfaces between the various requirements become clear. On a higher conceptual level, this should be done on a space level. The iteration step from space to object is where simulation knowledge is needed mostly. Verification of this interpretation now happens on the end of a phase, when this interpretation is defined in the iteration step from space requirement to object requirement, the design will have a complying starting point. This is also enables to focus more on the early phases of a design and to reduce variants and changes in later phases.

A second recommendation can be given regarding standardization. This comes in two fold. Firstly the structure of requirements has been researched which have brought forward two ways of defining a requirement. It is advisable to implement both structures in defining the requirements. Firstly adding the type of requirement in property, value, availability forces already to define a requirement in a non-ambiguous way. Secondly addressing the need and connecting it to requirements structures requirements. This gives a clear insight in which disciplines should be involved and what the interfaces are in the requirements.

Furthermore the standardization in information needs to improve. The iteration from a need to a space requirement and then to object requirement is not done consistently in projects. This therefor gives a variation in applicability of requirements as certain requirements are described as a need and some as object performance. The alignment in information of a design also needs to comply with these steps. In the process from a conceptual space type definition and relations is followed by a definition of the elements between the spaces. To make sure that the iteration step from space requirement to an object requirement is made properly, the definition of spaces in 3D model needs to be taken care for. As identified in the research a clear definition of spaces is missing. Here for a proposition is made for space types in a building. Setting up a standard of classifying spaces which follows the Dutch building standard makes it possible to define the space types where certain requirements are applying to. This will improve the iteration step from a space requirement to an object requirement. Also this will enable the usability of a BIM for verification of client specific requirements as the relation between spaces and objects is an important link to find the right elements. This classification in spaces should be developed sector-wide to ensure the usefulness and understandability.

A last recommendation relates to the implementation and improvement of an Object Type Library. Companies have already defined object type libraries to standardize the way of working. The way these libraries are built up varies amongst the sector greatly. The usage of a library based upon open standards is very useful to address this variety in built up as an open standard like the semantic web makes it possible to address many forms of data with the use of this standard. Via inference rules the connection between varying standards, all elements in a building can easily described. The addition of all properties and characteristics of this element make it possible to store a big amount of knowledge. The connection to other standards makes this library flexible and easy to extend. The usage of an OTL which addresses all the possible elements in a building can result in a connected database which opens up data exchange easily. When the library will be the main connection towards the different purposes data in various resources can be combined. For example the data of a 3D model can be connected via the OTL as amounts with a 3D

location to planning, costs, facility management or life cycle management. If this library is reused and extended in an open way, the development of the stored knowledge can result in continuous improvement. The development of this library should be however developed stepwise with firstly a usage for knowledge and secondly as project instancing database.

This development together with the total process improvement relate to a total development of a design process which is discussed in the next section.

## 7.2 Future development & research

This research towards client specific requirements has revealed firstly process improvements and secondly the implications of automated verification. The research has shown in summation the following possible developments;

- Standardization in defining requirements
- Improving the process of iteration of requirements
- Aligning requirements information to model information
- Difficulties in rule checkers and a proposition to overcome these difficulties
- The usage of Object Type libraries in design processes
- Prototype for automated verification of requirements

A future development from this research can be proposed in how to address projects. The transition in defining a performance before a design is iterated and products are chosen is essential in this development. The translation of space requirements to object performance is the most essential part where knowledge is needed. To transition the moment where this definition is made to an earlier moment in the design process can improve the quality of a design early on and potentially reduce errors. Aside from earlier on improving the design and ensuring compliancy with the needs of a client also the time for elaboration in subsequent phases will be reduced as the performance is already defined. This can increase the time available for early phases where simulation is needed. Generative and parametric design rely on this performance based definition of a design, this can be initiated more easily if the central focus on performance based designing is realized. A proposition for this process is as followed;

1. Investigation in the clients usage, needs and requirements
2. Simulation of relations, usage and structure of the building (space related)
3. Conceptualization of relations and structure of the building
4. Defining performance out of the conceptualization (space related)
5. Simulation of every discipline and testing to translate space performance to object performance
6. Performance based designing & engineering
7. Performance based product selection (parametric)

This development is an important extension which follows up from this research related to process improvement and integration of BIM and requirements data. The usage of an Object type library can be a very important part of this development. An OTL could be used as a platform which would work as a database. In the development of the checker a file is made understandable with the use of the BAM OTL. The IFC data is instantiated as elements of the OTL. This enables to see how many elements of certain type are apparent. Adding extra information to this OTL regarding properties or costs could make this OTL very useful for data management. When all elements of a model are instantiated in the database, the usage of information can be made easily visible for various purposes. For example for facility management the correct information could be easily accessed if the elements which are going to be checked are referenced in the instances in the platform. The semantic web standard makes this development possible. This also enables the usage of various software possible. Further research in this possibility is very interesting to investigate process management and improvement of the usage of object type libraries based on the semantic web standard.

A further investigation relates also to this process Improvement but more to the checking application. The extension of templates will remain an important part of improving the requirement checker. Here

opportunities lie in the completeness and to investigate how these templates could be used in real projects in the construction sector. Aside from further developing a checker with the use of templates also the essence of the checkers could be researched upon further. A checker is now used after a design is developed. A checker could be made more beneficial in two steps. Firstly as active checking during designing and secondly as not a checker but as parametric solution generator. The first proposition is mostly interface related research. The second proposition sketches a new purpose for the usage of rule checkers. The usage of a knowledge system is required for this purpose to store the parametric values, here for an OTL could be useful.

On the requirements side of this proposed development also a further investigation can be done in future research. The implementation of model based system engineering would align with the possibility of performance based designing. Model based systems engineering also aims to first conceptually define a model based on performances where understanding the problem is key. The possibilities with model based systems engineering in the construction sector can be very interesting to do research upon. This can closely relate to the development of parametric designing.



## 8. References

- Abanda, F. H., Zhou, W., Tah, J. H. M., & Cheung, F. (2013). Exploring the Relationships Between Linked Open Data and Building Information Modelling. *Sustainable Building Conference*, 176–185.
- Allemang, D., & Hendler, J. (2011). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL* (T. Green & R. Day, Eds.) (Second, Vol. 95). Waltham: Elsevier Ltd. <http://doi.org/10.2105/AfPH.2005.070169>
- BAMinfra. (2008). *SE-wijzer: Handleiding Systems Engineering*. Retrieved from [http://www.leidraadse.nl/assets/files/images/BN/bestanden/BAM\\_SE-wijzer.pdf](http://www.leidraadse.nl/assets/files/images/BN/bestanden/BAM_SE-wijzer.pdf)
- Beetz, J. (2009a). Building Product Catalogues on the Semantic Web. *Proceedings of the 26th International Conference on Information Technology in Construction CIB W78*.
- Beetz, J. (2009b). *Facilitating distributed collaboration in the AEC / FM sector using Semantic Web Technologies*. Eindhoven. Retrieved from <https://pure.tue.nl/ws/files/2966330/200911977.pdf>
- Beetz, J., Coebergh van den Braak, W., Botter, R., Zlatanova, S., & de Laat, R. (2015). Interoperable data models for infrastructural artefacts - a novel IFC extension method using RDF vocabularies exemplified with quay wall structures for harbors. *eWork and eBusiness in Architecture, Engineering and Construction*, 135–140. <http://doi.org/http://dx.doi.org/10.1061/9780784413616.071>
- Bhatt, M., Hois, J., & Kutz, O. (2012). Ontological modelling of form and function for architectural design. *Applied Ontology*, 7(3), 233–267. <http://doi.org/10.3233/AO-2012-0104>
- BIMForum. (2015). Level of Development Specification. *BIM Forum*, (April), 195. Retrieved from [www.bimforum.org/lod](http://www.bimforum.org/lod)
- BNA. (2005). *NL/Sfb-Tabellen Inclusief gereviseerde Elementenmethode '91*. Amsterdam. Retrieved from [http://www.stabu.org/wp-content/uploads/2015/07/NL-SfB\\_BNA\\_Boek\\_2005-ISBN-10-90-807626-3-6.pdf](http://www.stabu.org/wp-content/uploads/2015/07/NL-SfB_BNA_Boek_2005-ISBN-10-90-807626-3-6.pdf)
- BNA, NLIingenieurs, & ONRI. (2009). *Standaardtaakbeschrijving DNR-STB 2009*. Amsterdam. Retrieved from [http://www.bna.nl/fileadmin/user\\_upload/Helpdesk/bureauezaken/Standaardtaakbeschrijving\\_2009\\_de f.pdf](http://www.bna.nl/fileadmin/user_upload/Helpdesk/bureauezaken/Standaardtaakbeschrijving_2009_de f.pdf)
- Bouwend Nederland. (2014). *Praktische Leidraad voor geïntegreerd samenwerken met de UAVgc in de woning- en utiliteitsbouw*. Zoetermeer. Retrieved from <http://www.bouwendnederland.nl>
- Bruijn, J. de, Fensel, D., Kerrigan, M., Keller, U., Lausen, H., & Scicluna, J. (2008). *Modeling Semantic Web Services* (1st editio). Berlin: Springer.
- BuildingSmart. (2013). NEN-ISO 16739:2013.
- BuildingSmart. (2016). MVD Overview Summary. Retrieved July 26, 2016, from <http://www.buildingsmart-tech.org/specifications/mvd-overview/mvd-overview-summary>
- Chao-Duivis, M., Koning, A., & Ubink, A. (2013). *A Practical Guide to Dutch Building Contracts* (3rd Editio). The Hague.
- Chen, L., & Luo, H. (2014). A BIM-based construction quality management model and its applications. *Automation in Construction*, 46, 64–73. <http://doi.org/10.1016/j.autcon.2014.05.009>
- Chipman, T., Liebich, T., & Weise, M. (2016). mvdXML, 1.1, 49. Retrieved from <http://www.buildingsmart-tech.org/downloads/mvdxml/mvdxml-1.1/final/mvdxml-1-1-documentation>
- Curry, E., O'Donnell, J., Corry, E., Hasan, S., Keane, M., & O'Riain, S. (2013). Linking building data in the cloud: Integrating cross-domain building data using linked data. *Advanced Engineering Informatics*, 27(2), 206–219. <http://doi.org/10.1016/j.aei.2012.10.003>
- Dimyadi, J., & Amor, R. (2013). Automated Building Code Compliance Checking – Where is it at ? *Proceedings of CIB WBC 2013*, 172–185. <http://doi.org/10.13140/2.1.4920.4161>

- Dimyadi, J., Pauwels, P., Spearpoint, M., Clifton, C., & Amor, R. (2015). Querying a Regulatory Model for Compliant Building Design Audit. *Proc. of the 32nd CIB W78 Conference 2015, 27th-29th October 2015, Eindhoven, The Netherlands*, (October), 139–148. <http://doi.org/10.13140/RG.2.1.4022.6003>
- Douglass, B. P. (2013). *Systems engineering best practices: Model-based requirement analysis*. New York. Retrieved from <http://www-01.ibm.com/support/docview.wss?uid=swg27023356&aid=1>
- Eadie, R., Browne, M., Odeyinka, H., Mckeown, C., & McNiff, S. (2013). Automation in Construction BIM implementation throughout the UK construction project lifecycle: An analysis. *Automation in Construction*, 36, 145–151. <http://doi.org/10.1016/j.autcon.2013.09.001>
- Eastman, C., Lee, J. min, Jeong, Y. suk, & Lee, J. kook. (2009). Automatic rule-based checking of building designs. *Automation in Construction*, 18(8), 1011–1033. <http://doi.org/10.1016/j.autcon.2009.07.002>
- Eastman, C., Teicholz, P., Sacks, R., & Liston, K. (2011). *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*.
- Emes, M. R., Smith, A., & Marjanovic-Halburd, L. (2012). Systems for construction: lessons for the construction industry from experiences in spacecraft systems engineering. *Intelligent Buildings International*, 4(2), 67–88. <http://doi.org/10.1080/17508975.2012.680428>
- Feigenbaum, L. (2009). SPARQL By Example. W3C, 1–110. Retrieved from <https://www.w3.org/2009/Talks/0615-qbe/>
- Glinz, M. (2005). Rethinking the Notion of Non-Functional Requirements. *Proceedings of the Third World Congress for Software Quality*, (September), 55–64.
- Glinz, M., & Wieringa, R. J. (2007). Guest editors' introduction: Stakeholders in requirements engineering. *IEEE Software*, 24(2), 18–20. <http://doi.org/10.1109/MS.2007.42>
- Hjelseth, E., & Nisbet, N. (2010). Overview of concepts for model checking. *Proceedings of the CIB W78 2010: 27th International Conference –Cairo, Egypt*, 16–18.
- Hull, E. ., Jackson, K., & Dick, J. (2006). *Requirements Engineering. Requirements Engineering* (Vol. 13). <http://doi.org/10.1145/336512.336523>
- INCOSE. (2007). *Systems engineering handbook: A Guide for System Life Cycle Processes and Activities*. (C. Haskins, Ed.) (3.1 ed.).
- INCOSE. (2015). *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. (D. Walden, G. Roedler, K. Forsberg, D. Hamelin, & T. Shortell, Eds.) (4th ed.). Hoboken, NY: John Wiley and Sons.
- ISO/IEC/IEEE 15288. (2015). ISO/IEC/IEEE 15288:2015 Systems and software engineering – System life cycle processes.
- Kasim, T., Li, H., Rezgui, Y., & Beach, T. (2013). AUTOMATED SUSTAINABILITY PROCESS : PROOF OF CONCEPT 1 COMPLIANCE CHECKING The Need for Automated Sustainability Compliance Checking, (October), 30–31.
- Kim, T. W., Kim, Y., Cha, S. H., & Fischer, M. (2015). Automated updating of space design requirements connecting user activities and space types. *Automation in Construction*, 50(C), 102–110. <http://doi.org/10.1016/j.autcon.2014.12.010>
- Kiviniemi, A. (2005). Requirements management interface to building product models. *VTT Publications*, (572). Retrieved from <http://cife.stanford.edu/sites/default/files/TR161.pdf>
- Knublauch, H. (2011). SPIN - Modeling Vocabulary. W3C, (February 2011), 1–13. Retrieved from <http://www.w3.org/Submission/spin-modeling/>
- Krijnen, T., & van Berlo, L. (2016). Methodologies for requirement checking on building models. Retrieved from <http://bimserver.org/wp-content/uploads/sites/6/2016/06/ddss-2016-krijnen-vanberlo.pdf>

- Lenferink, S., Tillema, T., & Arts, J. (2013). Towards sustainable infrastructure development through integrated contracts: Experiences with inclusiveness in Dutch infrastructure projects. *International Journal of Project Management*, 31(4), 615–627. <http://doi.org/10.1016/j.ijproman.2012.09.014>
- Liebich, T., Adachi, Y., Forester, J., Hyvarinen, J., Richter, S., Chipman, T., ... Wix, J. (2013). Industry Foundation Classes Release 4 (IFC4) Documentation. Retrieved from <http://www.buildingsmart-tech.org/ifc/IFC4/final/html/>
- Lu, W., Fung, A., Liang, C., & Rowlinson, S. (2015). Demystifying construction project time-effort distribution curves: a BIM and non-BIM comparison Weisheng. Retrieved from <http://ascelibrary.org/doi/abs/10.1061/9780784413517.034>
- Lu, W., Fung, A., Peng, Y., Liang, C., & Rowlinson, S. (2014). Cost-benefit analysis of Building Information Modeling implementation in building projects through demystification of time-effort distribution curves. *Building and Environment*, 82, 317–327. <http://doi.org/10.1016/j.buildenv.2014.08.030>
- Malsane, S., Matthews, J., Lockley, S., Love, P. E. D., & Greenwood, D. (2015). Development of an object model for automated compliance checking. *Automation in Construction*, 49(PA), 51–58. <http://doi.org/10.1016/j.autcon.2014.10.004>
- Marchant, A. B. (2010). Obstacles to the Flow of Requirements Verification. *Systems Engineering*, 13(1). Retrieved from <http://doi.org/10.1002/sys.20127>
- Ministry of Infrastructure and the Environment. (2005). *Handreiking Functioneel Specificeren opq Handreiking Functioneel Specificeren*. Retrieved from [http://www.coinsweb.nl/downloads/Handreiking\\_functioneel\\_specificeren.pdf](http://www.coinsweb.nl/downloads/Handreiking_functioneel_specificeren.pdf)
- National Institute of Building Sciences. (2011). *BuildingSMART Alliance*. Retrieved from <http://www.buildingsmartalliance.org/index.php/nbims>
- Nawari, N. O., & Alsaffar, A. (2015). Practical Approaches for Computable Building Codes. *Proc. of the 32nd CIB W78 Conference 2015, 27th-29th October 2015, Eindhoven, The Netherlands*, 3(6), 569–576. <http://doi.org/10.13189/cea.2015.030601>
- Nederlands Normalisatie-instituut. (1993). *NEN 2574 - Construction drawings. Arrangement of data on building drawings*.
- Pauwels, P., & Terkaj, W. (2016). EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction*, 63, 100–133. <http://doi.org/10.1016/j.autcon.2015.12.003>
- Pauwels, P., Van Deursen, D., De Roo, J., Van Ackere, T., De Meyer, R., Van de Walle, R., & Van Campenhout, J. (2011). Three-dimensional information exchange over the semantic web for the domain of architecture, engineering, and construction. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 25(04), 317–332. <http://doi.org/10.1017/S0890060411000199>
- Pauwels, P., Van Deursen, D., Verstraeten, R., De Roo, J., De Meyer, R., Van De Walle, R., & Van Campenhout, J. (2011). A semantic rule checking environment for building performance checking. *Automation in Construction*, 20(5), 506–518. <http://doi.org/10.1016/j.autcon.2010.11.017>
- Pels, H., Beek, J., & Otter, A. (2013). Systems Engineering as a First Step to Effective Use of BIM. *Product Lifecycle Management for Society*, 409, 651–662. [http://doi.org/10.1007/978-3-642-41501-2\\_64](http://doi.org/10.1007/978-3-642-41501-2_64)
- ProRail. (2015). Handboek Systems Engineering ( SE ) Overzicht in processen , informatie en technieken, (April), 1–149. Retrieved from <http://www.leidraadse.nl/downloads>
- Radulovic, F., Poveda-Villalón, M., Vila-Suero, D., Rodríguez-Doncel, V., García-Castro, R., & Gómez-Pérez, A. (2015). Guidelines for Linked Data generation and publication: An example in building energy consumption. *Automation in Construction*, 57, 178–187. <http://doi.org/10.1016/j.autcon.2015.04.002>
- Rijkswaterstaat. (2015). Procesbeschrijving systems engineering voor RWS projecten, (Juni). Retrieved from <http://www.leidraadse.nl/downloads>

- Rijkswaterstaat, Bouwend Nederland, ProRail, & NLI ingenieurs. (2009). *Leidraad voor Systems Engineering binnen de GWW-sector*. Retrieved from <http://www.leidraadse.nl/downloads>
- Schaap, H., Bouwman, J., & Willems, P. (2008). *COINS-referentiekader voor functioneel specificeren*. Retrieved from [www.coinsweb.nl](http://www.coinsweb.nl)
- Scheithauer, D., Esep, I., & Forsberg, K. (2013). V-Model Views, (Köhler 1947), 502–516. Retrieved from <https://content.hitseng.eu/knowledge/pubs/downloads/vmv.pdf>
- Schneider, F., & Berenbach, B. (2013). A literature survey on international standards for systems requirements engineering. *Procedia Computer Science*, 16, 796–805. <http://doi.org/10.1016/j.procs.2013.01.083>
- Shishko, R., & Aster, R. (2007). *NASA systems engineering handbook*. Retrieved from <http://adsabs.harvard.edu/full/1995NASSP6105.....S>
- Solihin, W., & Eastman, C. (2015). A Knowledge Representation Approach to Capturing BIM Based Rule Checking Requirements Using Conceptual Graph. *Proc. of the 32nd CIB W78 Conference 2015, 27th-29th October 2015, Eindhoven, The Netherlands*.
- Solihin, W., & Eastman, C. (2015). Classification of rules for automated BIM rule checking development. *Automation in Construction*, 53, 69–82. <http://doi.org/10.1016/j.autcon.2015.03.003>
- Sparrius, A. (2014). The life cycle of a requirement. *INCOSE International Symposium*, 24(S1), 417–436. Retrieved from <http://onlinelibrary.wiley.com/dianus.lib.tue.nl/doi/10.1002/j.2334-5837.2014.00031.x/full>
- US Department of Defense Systems Management College. (2001). Systems Engineering Fundamentals. 22060-5565, (January), 222. Retrieved from [http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide\\_01\\_01.pdf](http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf)
- Visser, A. (2011). *Handboek Specificeren*. Ede: CROW. Retrieved from <http://www.crow.nl/publicaties/handboek-specificeren>
- W3C. (2011). What is the Semantic Web, 1. <http://doi.org/10.1016/B978-0-12-385965-5.10001-9>
- Walraven, A., & de Vries, B. (2009). From demand driven contractor selection towards value driven contractor selection. *Construction Management and Economics*, 27(6), 597–604. <http://doi.org/10.1080/01446190902933356>
- Werkgroep Leidraad Systems Engineering. (2007). *Leidraad voor Systems Engineering binnen de GWW-sector v.1*, 73. Retrieved from <http://www.leidraadse.nl/downloads>
- Young Jr., N. W., Jones, S. a, & Bernstein, H. M. (2007). Interoperability in the Construction Industry, 36. Retrieved from <http://www.aia.org/aiaucmp/groups/aia/documents/pdf/aia077485.pdf>
- Zhang, C., & Beetz, J. (2015). Model Checking on the Semantic Web: IFC Validation Using Modularized and Distributed Constraints. *Proc. of the 32nd CIB W78 Conference 2015, 27th-29th October 2015, Eindhoven, The Netherlands*, 819–827.
- Zhang, C., Beetz, J., & Vries, B. De. (2013). Towards Model View Definition on Semantic Level: A State of the Art Review, (July 2015), 1–10. Retrieved from [https://www.researchgate.net/publication/260763029\\_Towards\\_model\\_view\\_definition\\_on\\_semantic\\_level\\_a\\_state\\_of\\_the\\_art\\_review](https://www.researchgate.net/publication/260763029_Towards_model_view_definition_on_semantic_level_a_state_of_the_art_review)
- Zhang, C., Beetz, J., & Weise, M. (2015). Interoperable Validation for IFC Building Models Using Open Standards. *Journal of Information Technology in Construction*, 20(2015), 24–39. Retrieved from [https://www.researchgate.net/publication/271823060\\_Interoperable\\_validation\\_for\\_IFC\\_building\\_models\\_using\\_open\\_standards](https://www.researchgate.net/publication/271823060_Interoperable_validation_for_IFC_building_models_using_open_standards)
- Zhang, S., Teizer, J., Lee, J. K., Eastman, C. M., & Venugopal, M. (2013). Building Information Modeling (BIM) and Safety: Automatic Safety Checking of Construction Models and Schedules. *Automation in Construction*, 29, 183–195. <http://doi.org/10.1016/j.autcon.2012.05.006>

Zhong, B. T., Ding, L. Y., Luo, H. B., Zhou, Y., Hu, Y. Z., & Hu, H. M. (2012). Ontology-based semantic modeling of regulation constraint for automated construction quality compliance checking. *Automation in Construction*, 28, 58–70. <http://doi.org/10.1016/j.autcon.2012.06.006>

## Annex A – Interview Questions

The interview report with the transcription is available upon request. This is due to the confidentiality of the comments. Below the questions of the interviews are stated.

The following people have been interviewed

Date	Place	Name	Function
25-4-2016	Bunnik	Jeroen Mackaij	BIM Advisor
29-4-2016	Bunnik	Kobus van der Zwaal	SE advisor
2-5-2016	Bunnik	Ruud Verstegen	SE advisor
2-5-2016	Bunnik	Jaco Prins	BIM Advisor
4-5-2016	Utrecht	Gilby Moelands	SE advisor Infrastructure
9-5-2016	Bunnik	Robin van Esch	Project leader BIM
11-5-2016	Bunnik	Bert Leeuwis	SE Advisor
11-5-2016	Bunnik	Jeroen van Beek	Process manager
13-5-2016	Bunnik	Anne-Marie van Dijk	Project Coordinator
17-5-2016	Bunnik	Jeroen Harink	BIM Advisor
25-5-2016	Den Haag	Hans van Hoven	Plan Developer

### Design Process

1. What is your function within BAM?
2. Where do mistakes in the design process occur most often?
3. How can these mistakes be prevented?
4. How should the design process be changed to effectuate this prevention?
5. Who is responsible for these changes?
6. What kind of requirements causes the most errors in the design process?

### Systems Engineering

7. What is your definition of Systems Engineering?
8. What are the biggest benefits of Systems Engineering
9. What is the biggest problem with the implementation of Systems Engineering in the AEC industry?
10. What is good method to create a System breakdown structure in relation to the link between objects and spaces?
11. What is your definition of BIM?
12. How do you see a connection between BIM and Systems Engineering?
13. When looking to this connection, what are the biggest limitations to realise this connection?
14. Where lies the future in the field of Systems Engineering?

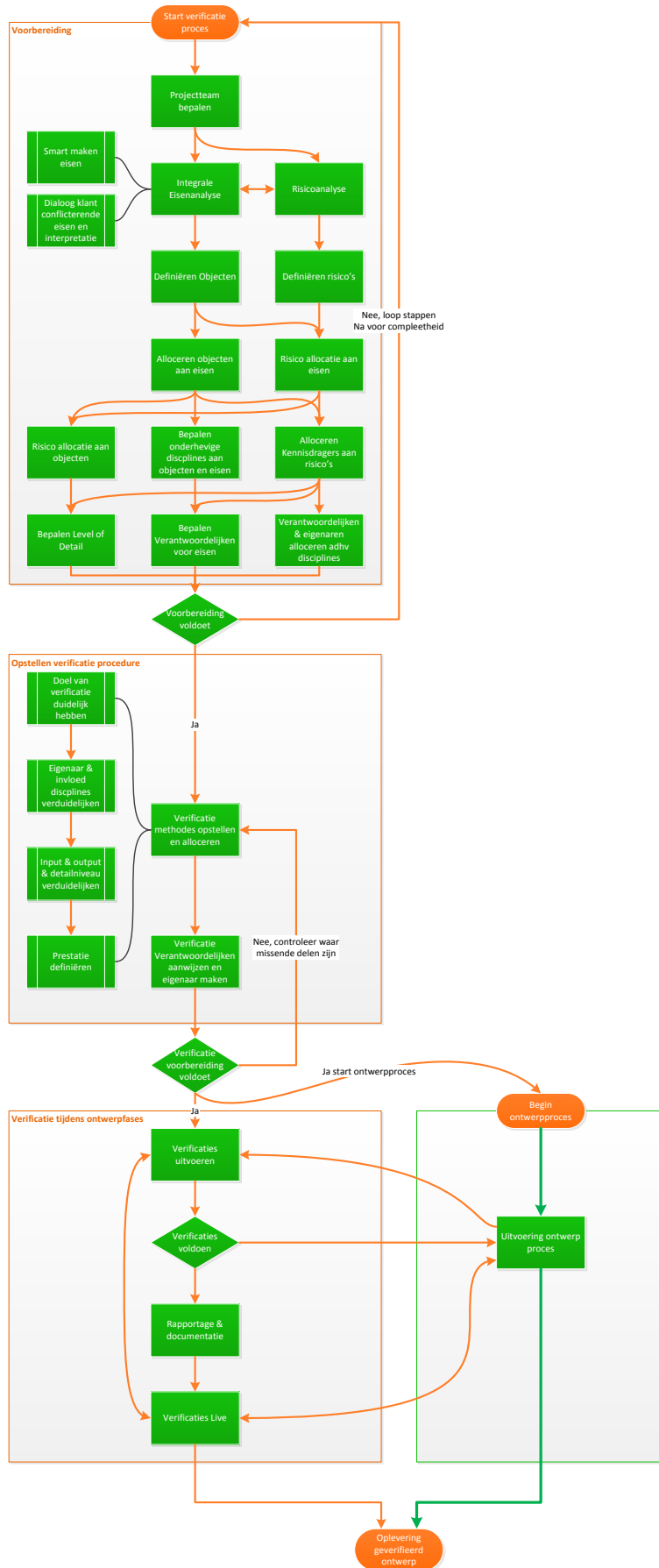
### Verification

15. What are the most costly mistakes if they need to be redone?
16. How should verification take place in the design process?
17. What is your definition of a good verification process?
18. What goes wrong most often in a verification process?
19. What are the most important parts of a verification process?
20. When and how often should verification take place?
21. Who is and who should be responsible for verification during the design process?
22. Which type of requirements are the hardest to verify?

### Automation of verification

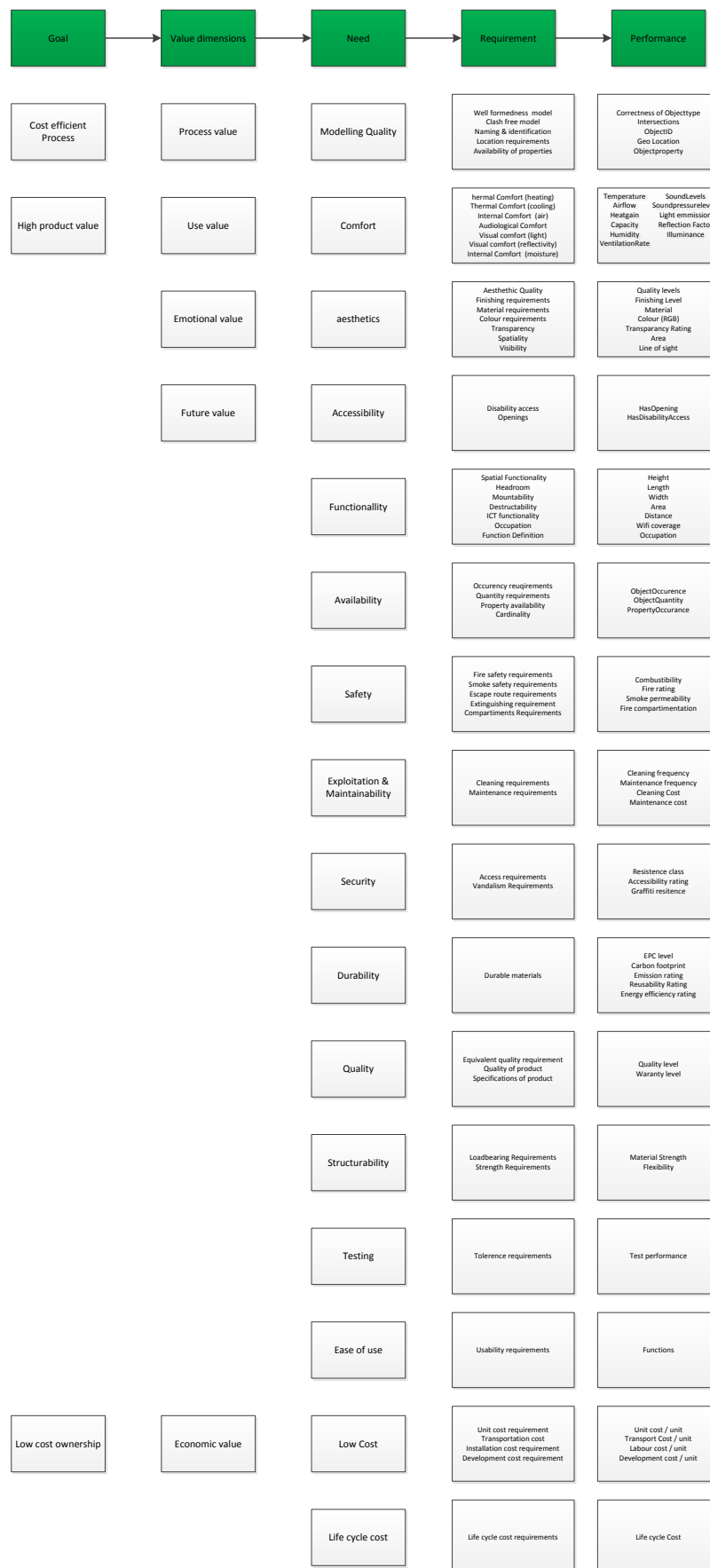
23. Can verification be done with the use of rule checking techniques?
24. Is automated verification the future?
25. How do you imagine that a translation of requirements in to rules should be done?
26. What kind of automated requirement check would you think is the most useful?
27. How should this automated requirements checker work?

## Annex B – Verification Process



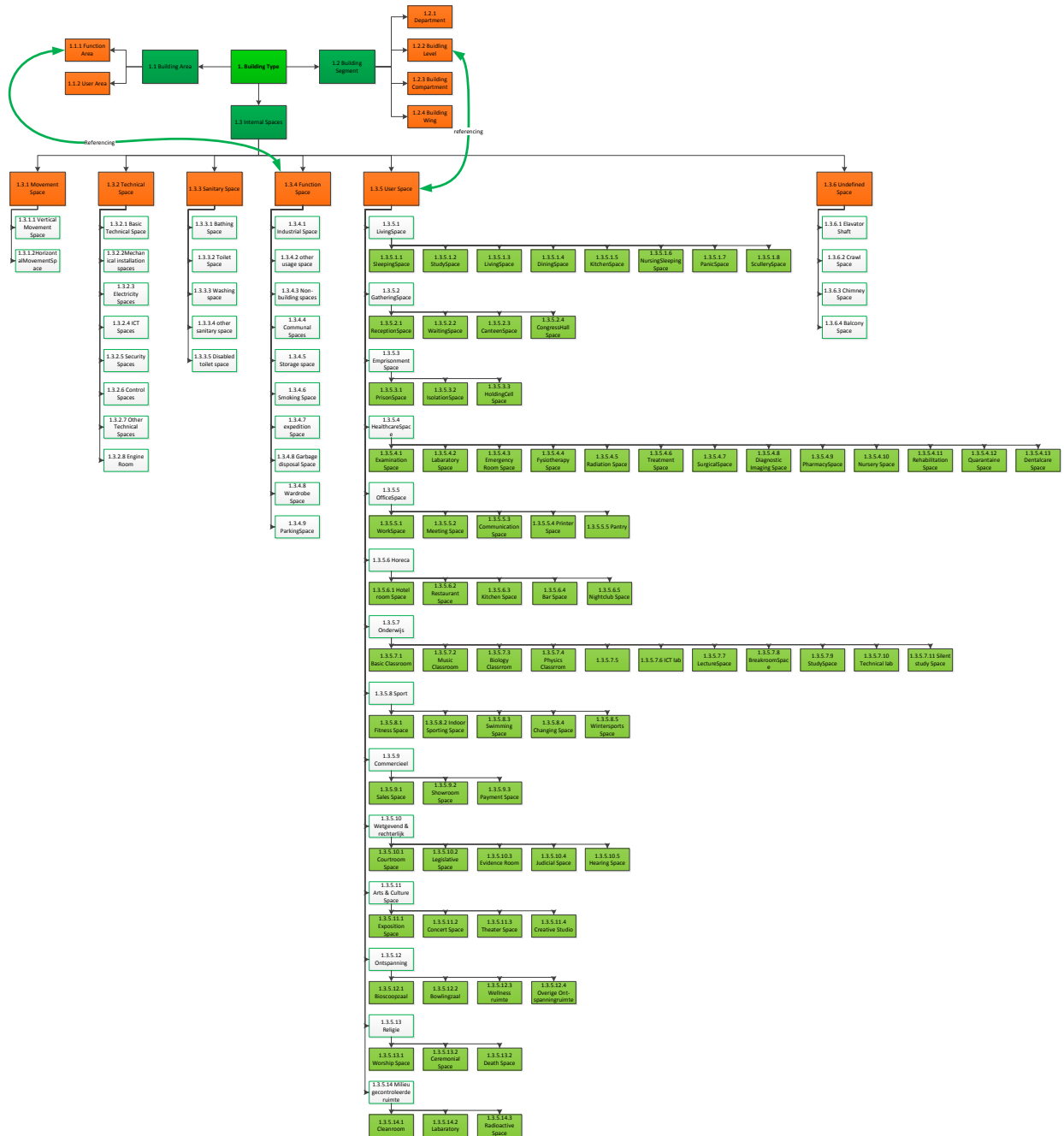


## Annex C – Requirements Hierarchy



## Annex D – Space Classification

This classification is here represented as a figure to show the hierarchy. In Annex G this hierarchy is translated to the BAM OTL and represented as an ontology. The turtle file is available upon request.



## Annex E – Selected Requirements

These requirements have been selected for the further development of the requirement checker. From these requirements two requirements have been implemented in the prototype.

Need	Description	Used template
Comfort <i>Acoustic comfort</i>	Value checking of a wall between two different spaces <b>A wall between a User and a movement space must have a maximum acoustic rating of 65 dB</b>	Spaces – wall - object relation; value
	Objects: Wall, User Space, Movement Space Property: Acoustic Rating Value: 65 dB	
Modelling quality <i>Well formed</i>	Data quality checking of objects; Classification coding <b>A wall must have the correct NL-SfB code (22.11)</b>	Value
	Objects: Wall Property: Classification Code Value: 22.11	
Safety <i>Fire safety</i>	Material checking; Combustibility <b>A fire separation wall must be made of a material Non-Combustible</b>	Property existence
	Objects: Fire Separation Wall Property: Non-Combustible	
Spatiality <i>Area</i>	Gross area checking of space types <b>The gross area of an office type X must be at least 24m2</b>	Space type value
	Objects: Office Space Property: Area Value 24 m2	
Aesthetics <i>Size</i>	Checking of size of objects in a wall in a certain space <b>A window in an external wall in an office space must have a size of 1000mm</b>	Space - object relation; value
	Objects: Window, External Wall, Office Space Property: Size Value: 1000 mm	
Functionality <i>Headroom</i>	Wall Height Checking <b>Internal walls must be at least 2500mm high</b>	Value
	Objects: Internal Wall Property: Height Value: 2500 mm	
Availability <i>Object Occurrence</i>	Object occurrence in a space checking <b>Every unique user space must contain a smoke detector</b>	Spaces - object Containment
	Objects: User Space, Smoke Detector Property: Containment	

Need	Description	Used template
Availability <i>Object quantity</i>	Amount of objects in a space checking <b>Every unique user space must contain at least 3 power sockets</b>	Spaces - object Containment amount
	Objects: User Space, power sockets Property: Amount Value: 3	
Safety <i>Fire Safety</i>	Value checking of an object in a certain Space type <b>The walls in an office space must have the property Fire resistance class</b>	Space - object relation; property
	Objects: Wall, Office Space Property: Fire Resistance	
Costs <i>Unit costs</i>	Cost requirements checking <b>Internal walls metal stud must not exceed the cost of €X,- per m2</b>	Value
	Objects: Internal Metal stud Wall Property: Cost Value: €/m2 Value: 3	

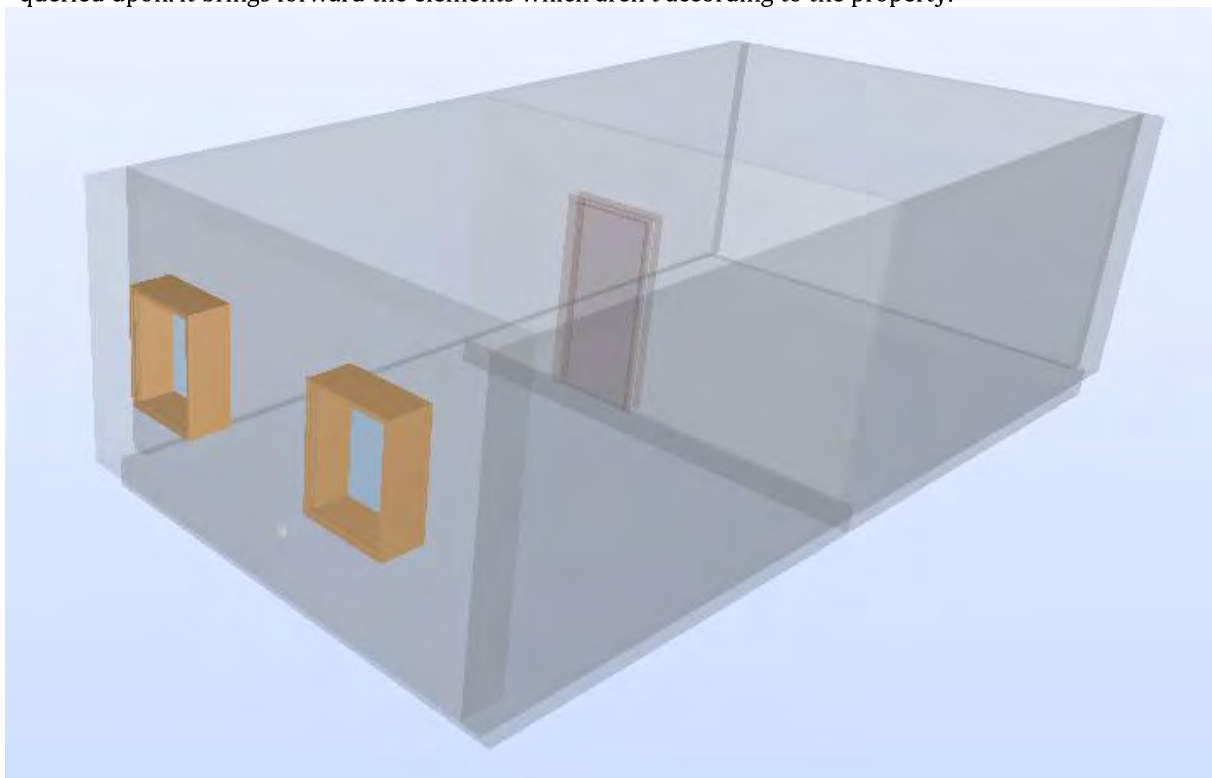
## Annex F – SPIN Constraint Templates

In this annex the Constraint templates are stated and an example is given.

### Template 1: Type Existence

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ?ObjectClass .   FILTER NOT EXISTS {     ?s ?p ?o .     ?o a ?D .   } . } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:window .   FILTER NOT EXISTS {     ?s qrw:containedIn ?o .     ?o a bam:ExternalWall .   } . } </pre>
<b>Arguments which must be defined</b>	<b>ObjectClass</b> = a certain object from the BAM OTL <b>p</b> (predicate) = a condition where the existence is applying to <b>D</b> = an element where the objectclass has a condition to

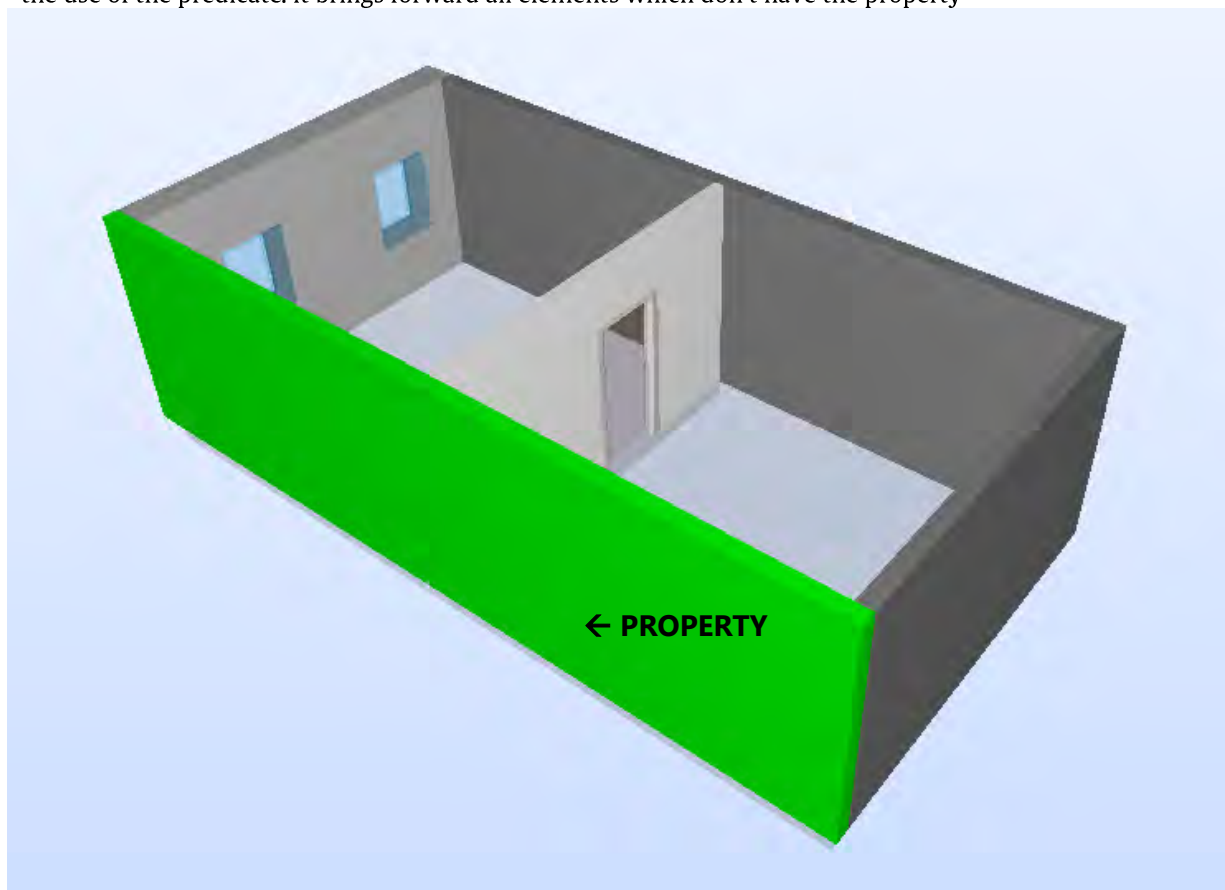
This template checks the existence of a certain type. In the example the existence of a window in a wall is queried upon. It brings forward the elements which aren't according to the property.



## Template 2: Property Existence

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ?ObjectClass .   FILTER NOT EXISTS {     ?s ?p ?o .   } . } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:wall .   FILTER NOT EXISTS {     ?s qrw:HasClassification ?o .   } . } </pre>
Arguments which must be defined	<b>ObjectClass</b> = a certain object from the BAM OTL <b>p</b> (predicate) = a property which must be present as a relation of the object

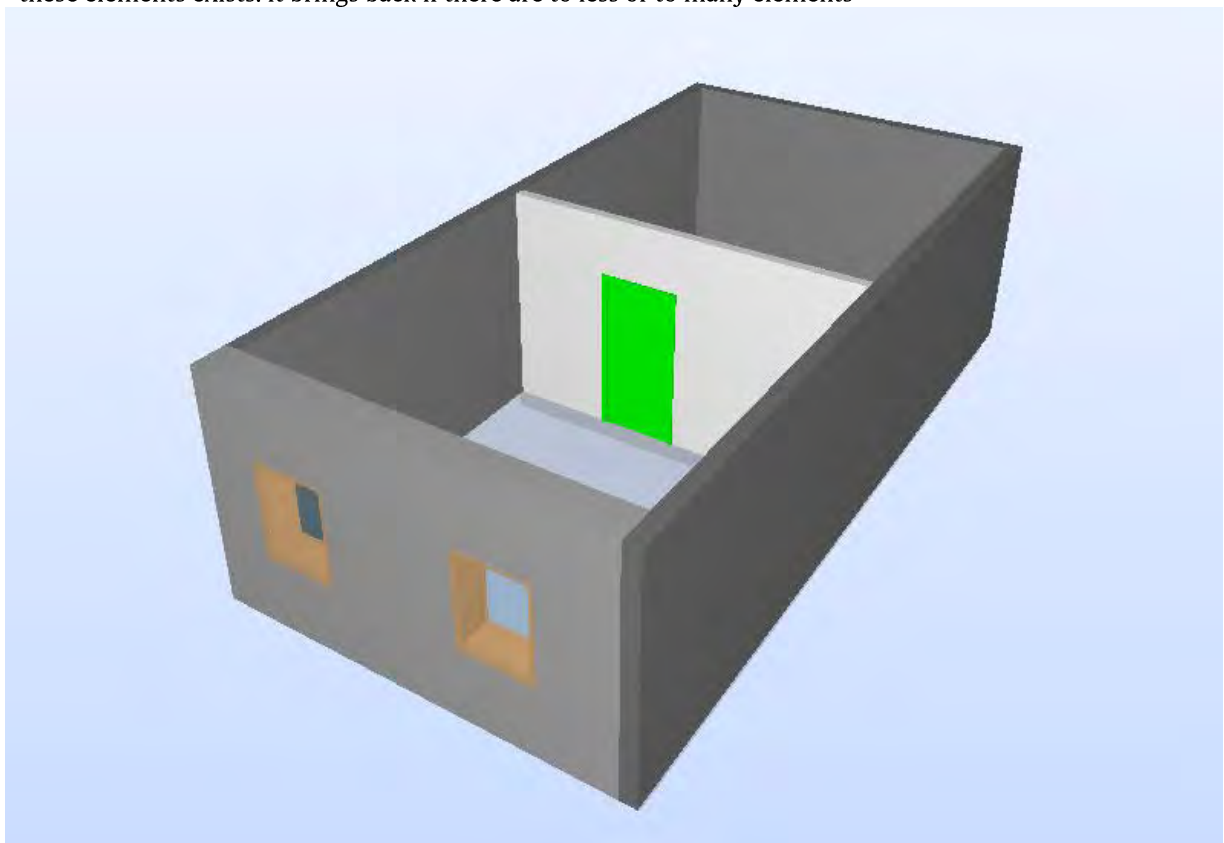
This template checks if a property exists. It queries an element and then a property is queried upon with the use of the predicate. It brings forward all elements which don't have the property



## Template 3: Cardinality

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ?ObjectClass .   FILTER {     (op:count(?s, ?p, ?D) &gt; ?n)        (op:count(?s, ?p, ?D) &lt; ?m)) .   } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:Door .   FILTER {     (op:count(bam:Door, pset:compartmentation, bam:Space) &gt; 1)    (op:count bam:Door, pset:compartmentation, bam:Space) &lt; 1)) .   } </pre>
Arguments which must be defined	<p><b>ObjectClass</b> = a certain object from the BAM OTL</p> <p><b>p</b> (predicate) = a property which must be present as a relation of the object</p> <p><b>D</b> = an element where the objectclass has a condition to</p> <p><b>n</b> = amount of objects minimum</p> <p><b>m</b> = amount of objects maximum</p>

This template checks if a certain object exist exactly. It queries an element and then filters how much of these elements exists. It brings back if there are to less or to many elements

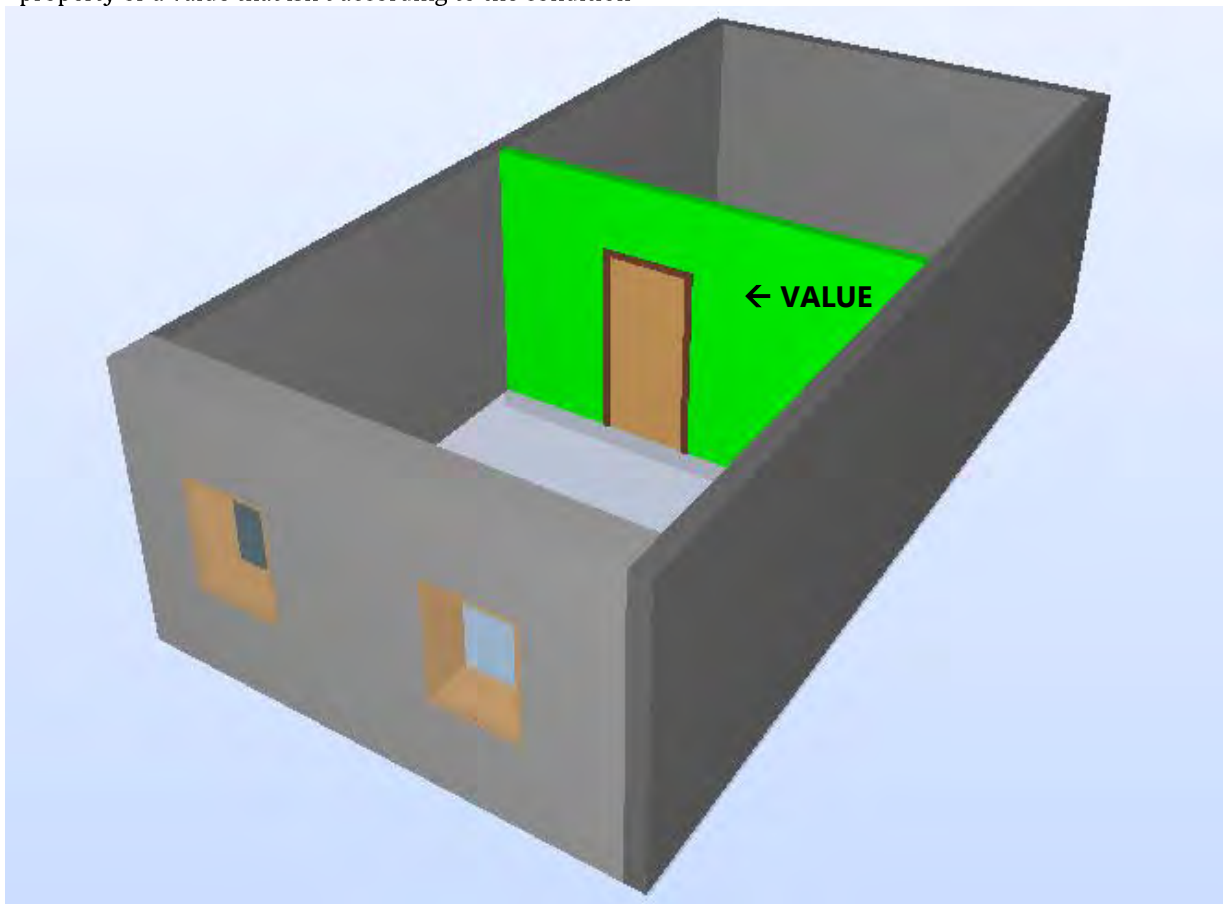




## Template 4: Value

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ?ObjectClass .   FILTER NOT EXISTS {     ?s ?p ?o .     FILTER operator(?o, ?a) .   } . } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:Wall .   FILTER NOT EXISTS {     bam:Wall pset:acousticRating ?o.     FILTER &gt;(?o, 65) .   } . } </pre>
Arguments which must be defined	<p><b>ObjectClass</b> = a certain object from the BAM OTL</p> <p><b>p</b> (predicate) = a property which must be present as a relation of the object</p> <p><b>operator</b> = the condition for the value (=, &lt;, &gt; etc.)</p> <p><b>a</b> = Value of the property</p>

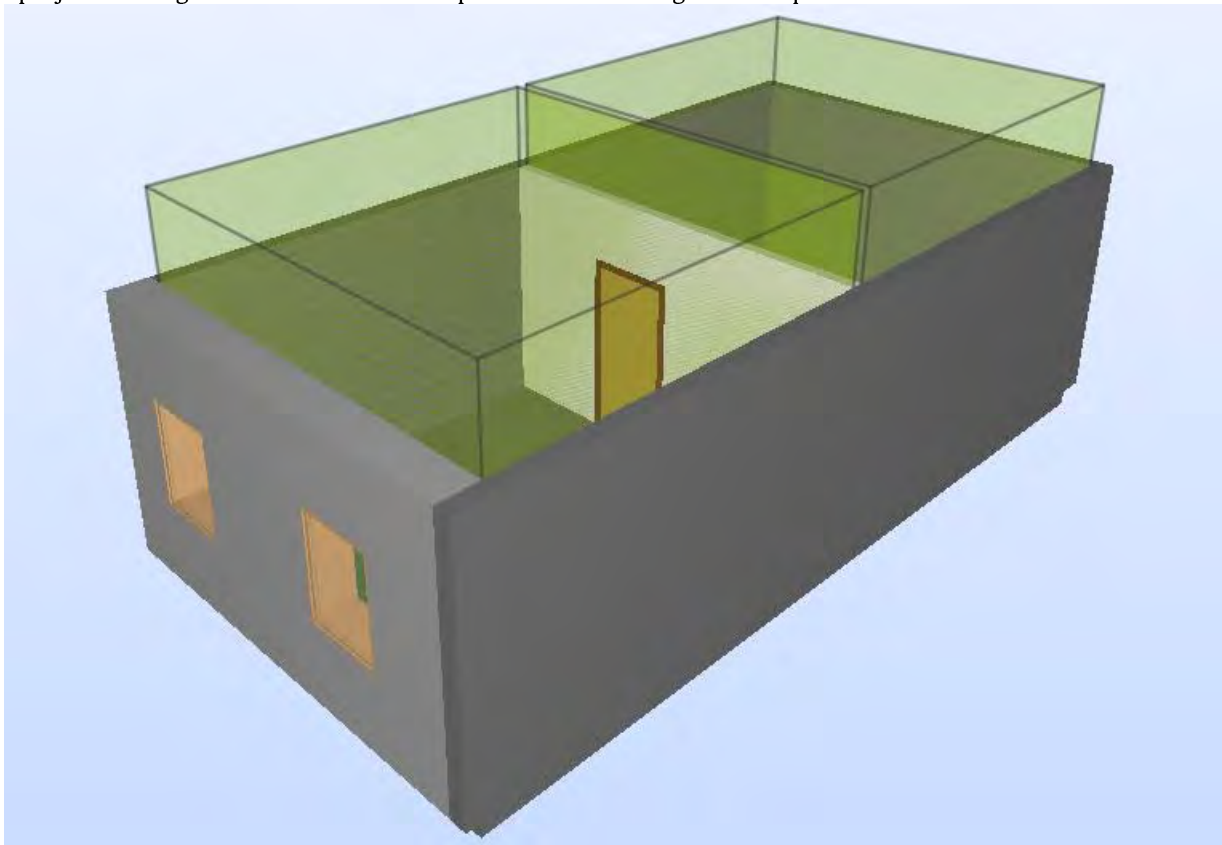
This template checks if an object has a property with a certain value. It queries an object and filters if the value of property is according to a certain value. It brings forward all elements which don't have the property or a value that isn't according to the condition



Template 5: Space type amount

Space type amount	Example
<pre>CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ?SpaceClass .   FILTER {     (op:count(?s, ?p, ?D) &gt; ?n)        (op:count(?s, ?p, ?D) &lt; ?m) .   } . }</pre>	<pre>CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:OfficeSpace .   FILTER {     (op:count(       qrw:ContainedIn       ?bam:Project) &gt; x    (Bam:Officespace       qrw:ContainedIn       ?bam:Project) &lt; x).   } . }</pre>
Arguments which must be defined	<p><b>SpaceClass</b> = a certain space from the BAM OTL</p> <p><b>p</b> = a predicate which must be present as a relation of the object to the total project</p> <p><b>D</b> = an element where the space has a condition to</p> <p><b>n</b> = required value for the amount of spaces</p> <p><b>m</b> = required value for the amount of spaces</p>

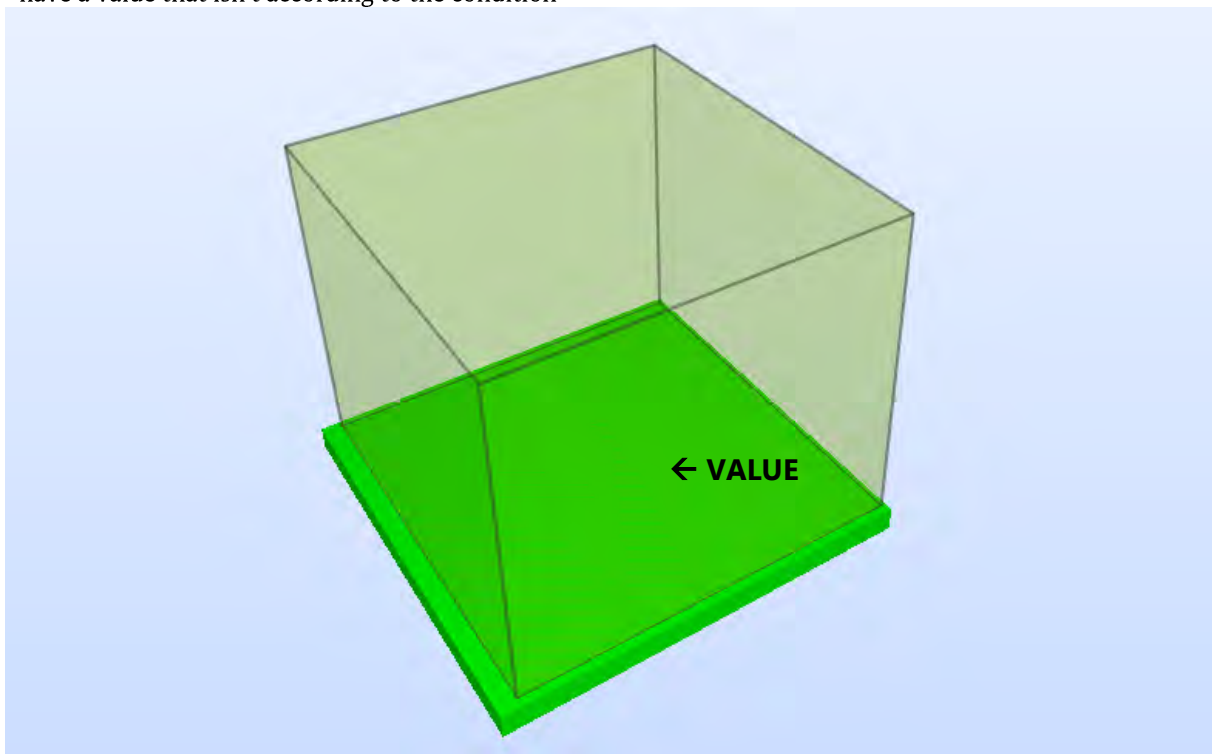
This template checks if a certain amount of a space type exists in the model. It queries a spacetype in the project. It brings back if the amount of spaces isn't according to the required value



## Template 6: Space type value

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ?SpaceClass .   FILTER NOT EXISTS {     ?s ?p ?o .     FILTER operator (?o, 20) .   } . } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:OfficeSpace .   FILTER NOT EXISTS {     Bam:OfficeSpace pset:GrossPlannedArea ?o .     FILTER &gt;(?o, 20) .   } . } </pre>
Arguments which must be defined	<p><b>SpaceClass</b> = a certain space from the BAM OTL</p> <p><b>p</b> = a predicate which must be present as a relation between the space and the property</p> <p><b>operator</b> = the condition for the value (=, &lt;, &gt; etc.)</p> <p><b>a</b> = Value of the property</p>

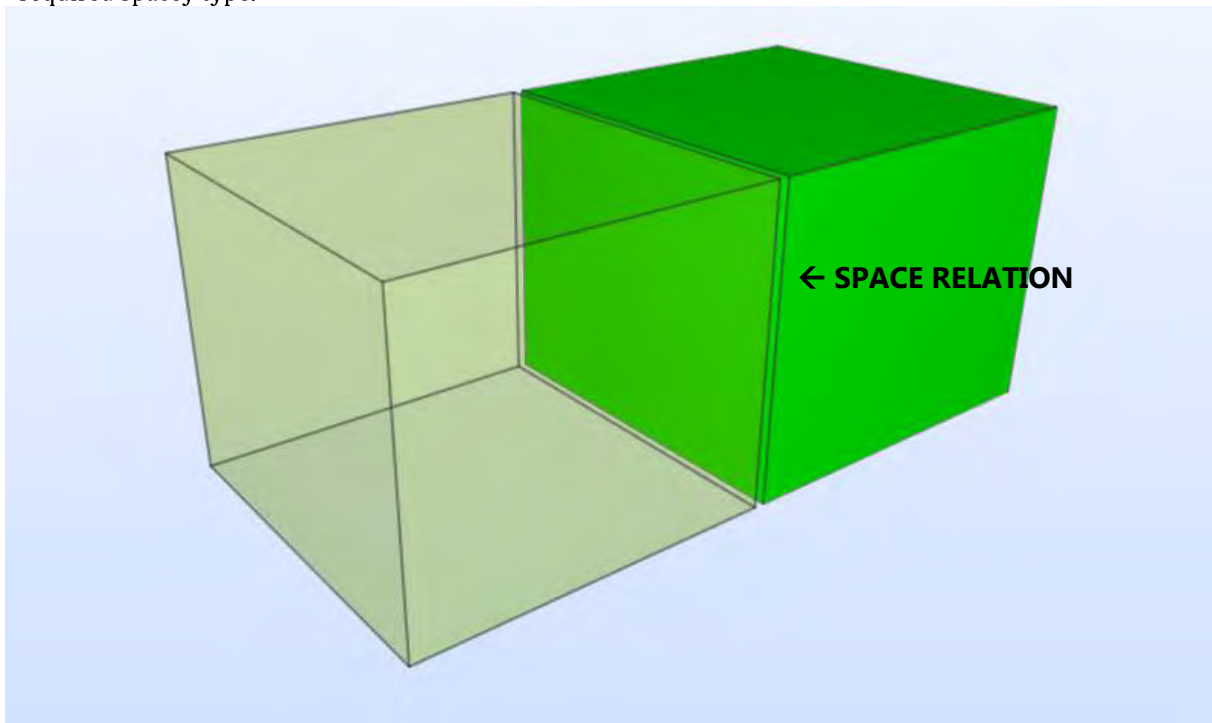
This template checks if a space has a property with a certain value. It queries a space and filters if the value of property is according to a certain value. It brings forward all spaces which don't have the property or have a value that isn't according to the condition



## Template 7: Space – Space Relation

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ?SpaceClass .   FILTER NOT EXISTS {     ?s ?p ?o .     ?o a ?D .   } . } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:OfficeSpace .   FILTER NOT EXISTS {     Bam:OfficeSpace qrw:nextspace ?o     ?o a bam:VerticalMovementSpace .   } . } </pre>
Arguments which must be defined	<p><b>ObjectClass</b> = a certain object from the BAM OTL</p> <p><b>p</b> (predicate) = a property which must be present as a relation of the object</p> <p><b>operator</b> = the condition for the value (=, &lt;, &gt; etc.)</p> <p><b>a</b> = Value of the property</p>

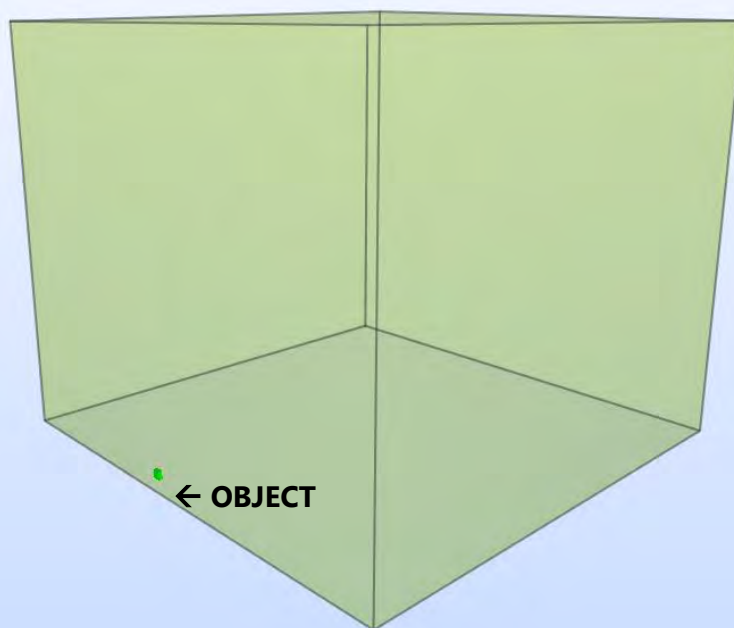
This template checks if a certain space type is adjacent to another space type. It queries the relation between two different space types. It brings forward all primary space types which haven't got an adjacent required space type.



## Template 8: Space – object Containment

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ? SpaceClass .   FILTER NOT EXISTS {     ?s ?p ?o .     ?o a ?objectClass .   } . } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:OfficeSpace .   FILTER NOT EXISTS {     bam:officeSpace hasContainedProduct ?o     ?o a Bam:PowerSocket   } . } </pre>
Arguments which must be defined	<p><b>SpaceClass</b> = a certain space from the BAM OTL</p> <p><b>p</b> = a predicate which must be present as a relation between a space and an object</p> <p><b>ObjectClass</b> = a certain object from the BAM OTL</p>

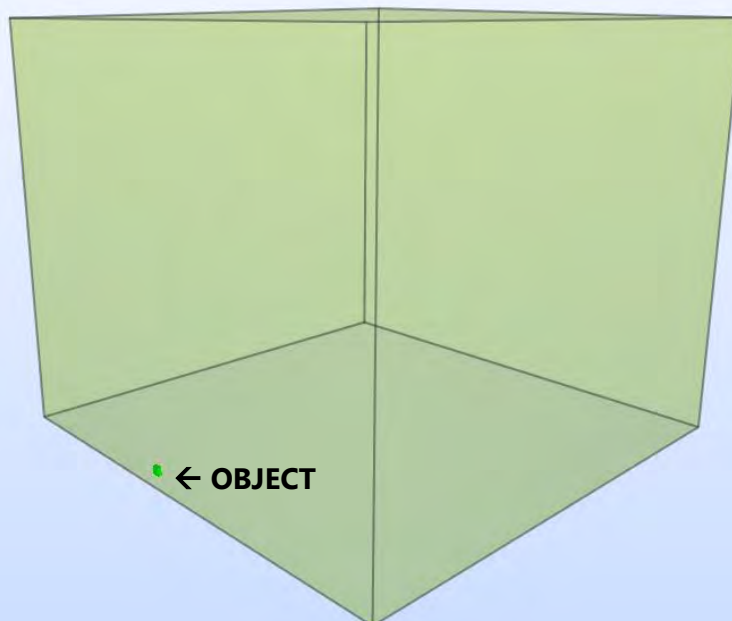
This template checks if an object is contained in a space. It queries a space and filters out the spaces which don't contain the object. It brings forward all spaces which don't contain the objects.



## Template 9: Space – object Containment amount

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ? SpaceClass .   FILTER (op:count(?o, ?p, ?D) &gt; ?n)      (op:count(?o, ?p, ?D) &lt; ?m)) } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:OfficeSpace .   FILTER {     op:count(bam:officeSpace hasContainedProduct Bam:PowerSocket) &gt; ?n)    (op:count(bam:officeSpace hasContainedProduct Bam:PowerSocket) &lt; ?m)) } . } </pre>
Arguments which must be defined	<p><b>SpaceClass</b> = a certain space from the BAM OTL</p> <p><b>p</b> (predicate) = a property which must be present as a relation of the object</p> <p><b>D</b> = an object which is contained in the Spaceclass</p> <p><b>n</b> = amount of objects minimum</p> <p><b>m</b> = amount of objects maximum</p>

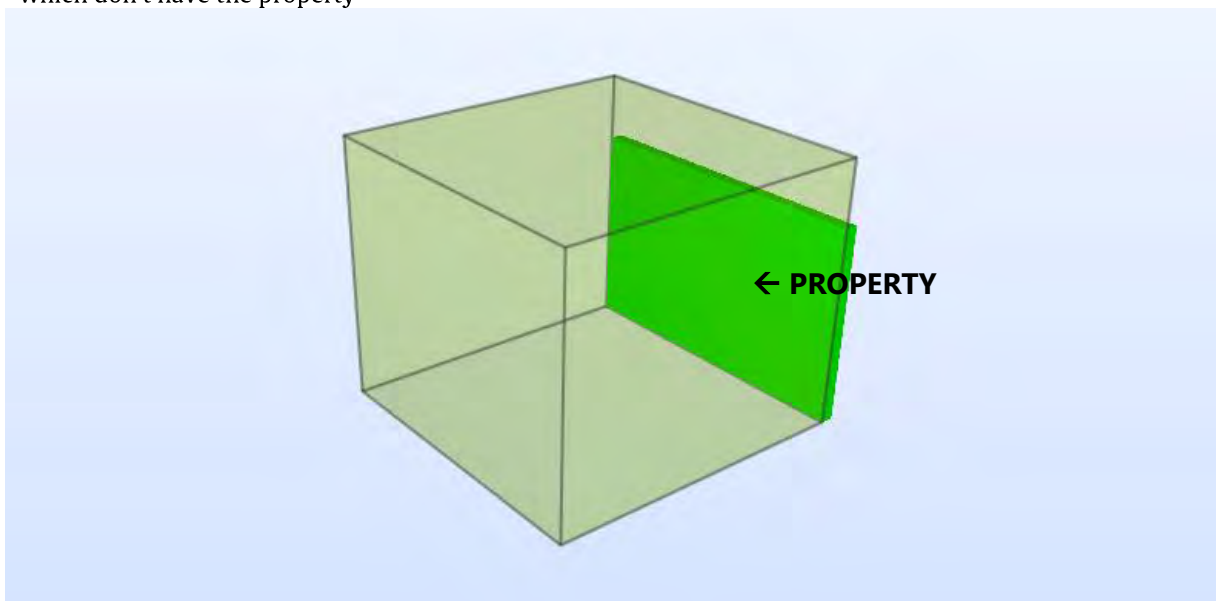
This template checks if an object is contained in a space. It queries a space and filters out the spaces which don't contain the object. It brings forward all spaces which don't contain the objects.



## Template 10: Space - object relation; property

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ?objectClass .   ?s qrw:isBoundaryOf ?o .   ?o a ?spaceClass .   FILTER NOT EXISTS {     ?s ?p2 ?o2 .   } . } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:InternalWall .   ?s qrw:isBoundaryOf ?o .   ?o a bam:OfficeSpace .   FILTER NOT EXISTS {     Bam:InternalWall pset:Combustable true   } . } </pre>
Arguments which must be defined	<p><b>SpaceClass</b> = a certain space from the BAM OTL</p> <p><b>ObjectClass</b> = a certain object from the BAM OTL</p> <p><b>p</b> = a predicate which must be present as a property of the contained object</p>

This template checks if an object contained in a certain space has a property. It queries objects which are present in a space and filters out the objects which don't have the property. It brings forward all objects which don't have the property

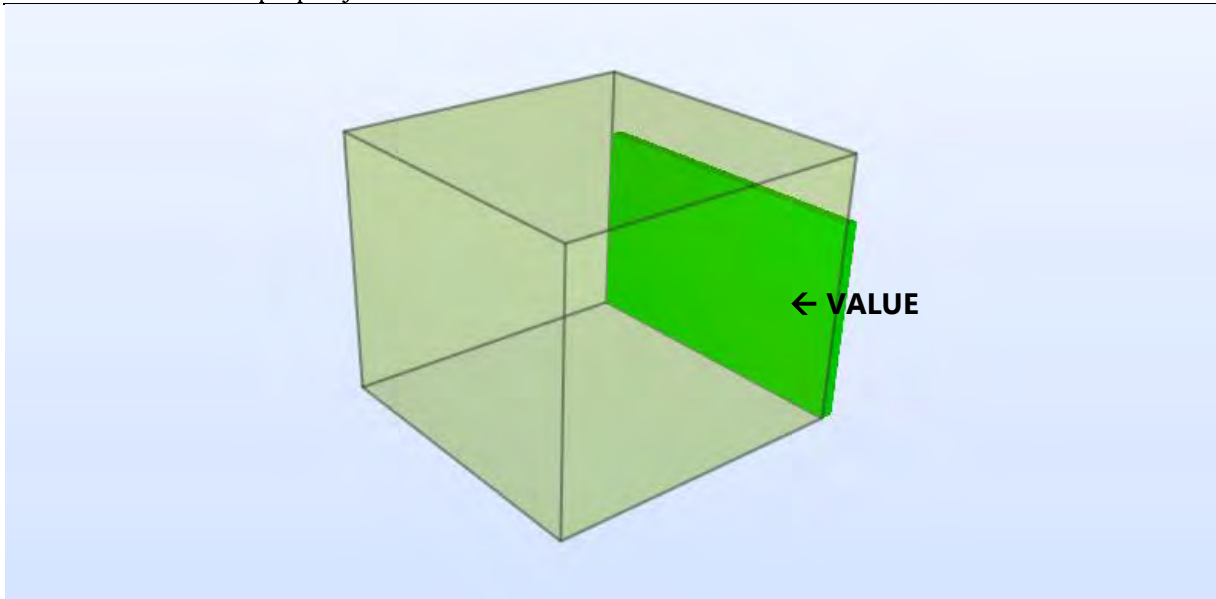




Template 11: Space - object relation; value

Statement	Example
<pre>CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s qrw:isBoundaryOf ?o .   ?s a ?objectClass .   ?o a ?spaceClass .   FILTER NOT EXISTS {     ?s ?p2 ?o2 .     FILTER operator(?o2, ?a) .   } . }</pre>	<pre>CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s qrw:isBoundaryOf ?o .   ?s a bam:wall   ?o a bam:OfficeSpace .   FILTER NOT EXISTS {     ?s pset:acousticRating ?o2 .     FILTER (?o2 &gt; 45) .   } . }</pre>
Arguments which must be defined	<b>SpaceClass</b> = a certain space from the BAM OTL <b>ObjectClass</b> = a certain object from the BAM OTL <b>p</b> = predicate as a property of the contained object <b>operator</b> = the condition for the value (=, <, > etc.) <b>a</b> = a value

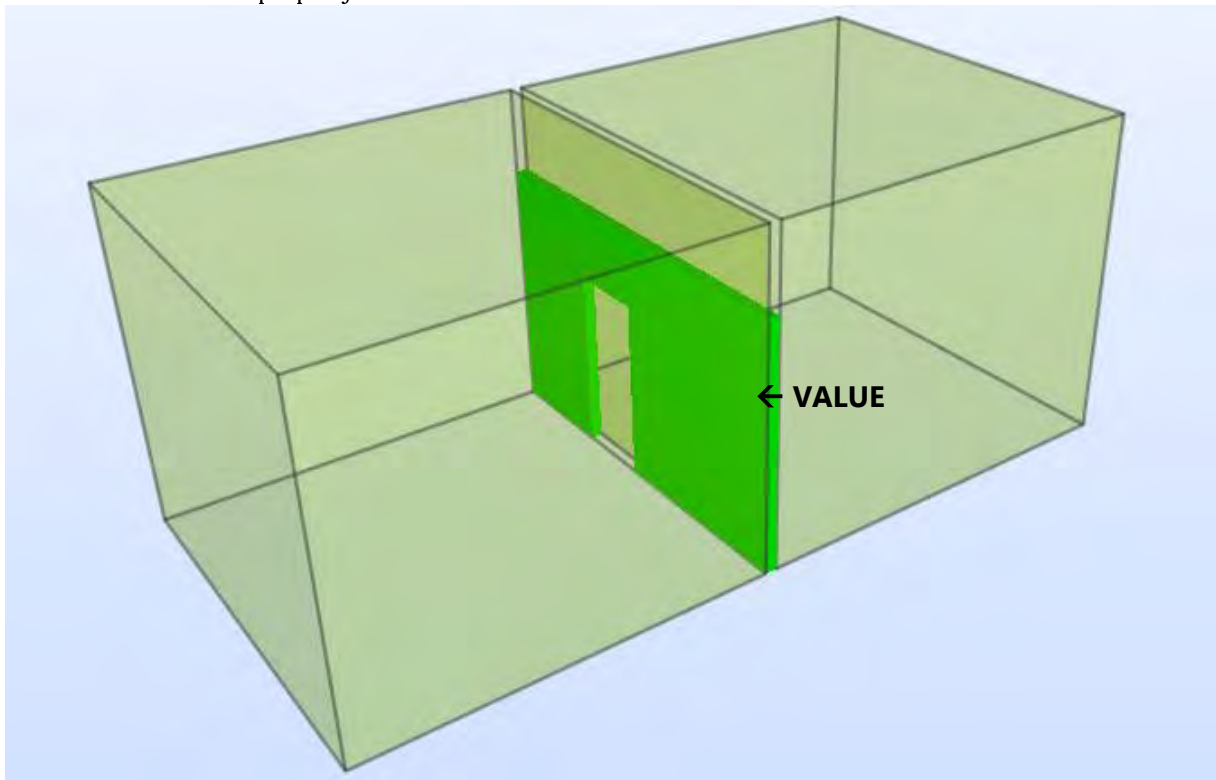
This template checks if an object contained in a certain space has a property. It queries objects which are present in a space and filters out the objects which don't have the property. It brings forward all objects which don't have the property



## Template 12: Spaces - wall relation; value

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ?objectClass .   ?s qrw:isBoundaryOf ?o .   ?o a ?spaceClass(X) .   ?s qrw:isBoundaryOf ?o2 .   ?o2 a ?spaceClass(Y) .   FILTER NOT EXISTS {     ?s ?p ?o3 .     FILTER operator(?o3, ?a) .   } . } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:internalWall .   ?s qrw:isBoundaryOf ?o .   ?o a bam:OfficeSpace .   ?s qrw:isBoundaryOf ?o2 .   ?o2 a bam:HorizontalMovementSpace .   FILTER NOT EXISTS {     ?s pset:acousticRating ?o3 .     FILTER (?o3 &gt; 55) .   } . } </pre>
Arguments which must be defined	<p><b>SpaceClass</b> = a certain space from the BAM OTL (X&amp;Y)</p> <p><b>ObjectClass</b> = a certain object from the BAM OTL</p> <p><b>p</b> = predicate as a property of the contained object</p> <p><b>operator</b> = the condition for the value (=, &lt;, &gt; etc.)</p> <p><b>a</b> = a value</p>

This template checks if an object contained in a certain space has a property. It queries objects which are present in a space and filters out the objects which don't have the property. It brings forward all objects which don't have the property



---

Template 13: Spaces - object relation; value

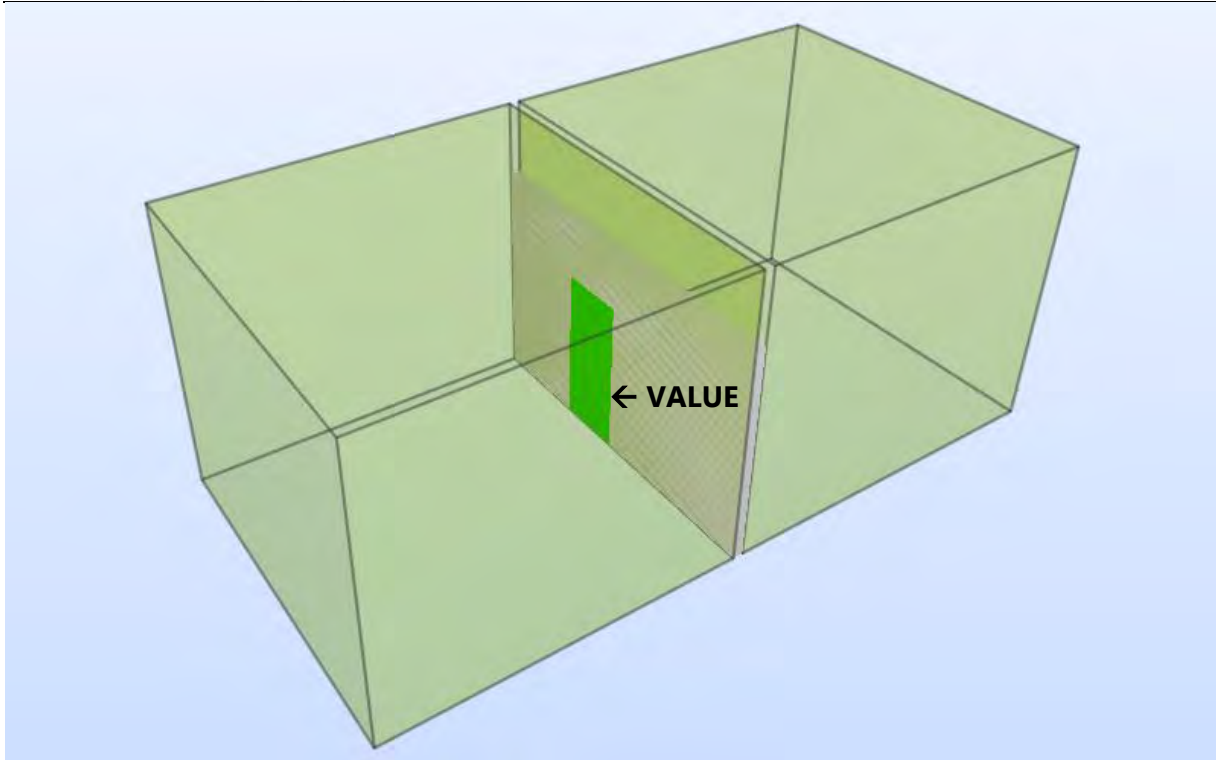
---

Statement	Example
<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a ?objectClass(X) .   ?s qrw:isPlacedIn ?o .   ?o a ?objectClass(Y) .   ?s qrw:isBoundaryOf ?o2 .   ?o2 a ?spaceClass .   FILTER NOT EXISTS {     ?s ?p ?o3 .     FILTER operator(?o3, ?a) .   } . } </pre>	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:Window ..   ?s qrw:isPlacedIn ?o .   ?o a bam:Wall.   ?s qrw:isBoundaryOf ?o2 .   ?o2 a bam:OfficeSpace .   FILTER NOT EXISTS {     ?s pset:Size ?o3 .     FILTER (?o3 &gt; 1000) .   } . } </pre>
Arguments which must be defined	<p><b>ObjectClass</b> = a certain object from the BAM OTL (X&amp;Y)</p> <p><b>SpaceClass</b> = a certain space from the BAM OTL</p> <p><b>p</b> = predicate as a property of the contained object</p> <p><b>operator</b> = the condition for the value (=, &lt;, &gt; etc.)</p> <p><b>a</b> = a value</p>

---

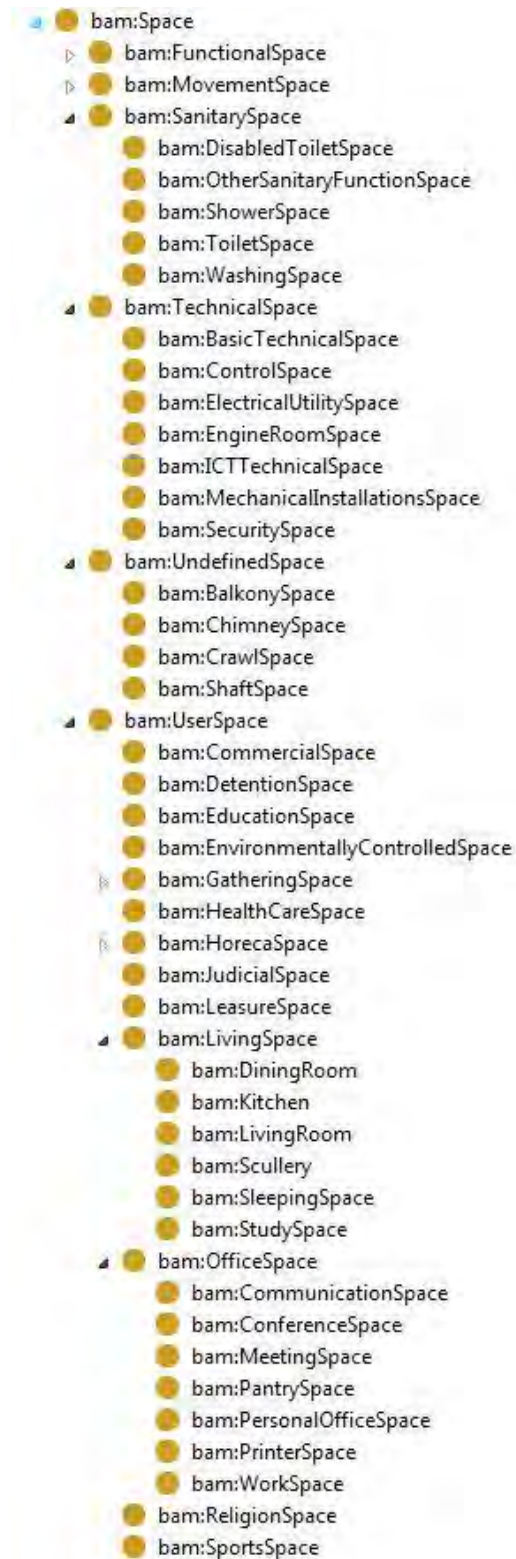
This template checks if an object contained in a certain space has a property. It queries objects which are present in a space and filters out the objects which don't have the property. It brings forward all objects which don't have the property

---



## Annex G – BAM OTL

Firstly the overview in Topbraid of all created elements can be seen. After that the notation in turtle can be seen.



```
# baseURI: http://www.bam.com/bamNL_otl
# imports: http://bimsparql.org/pset#
# imports: http://bimsparql.org/query-rewriting
# imports: http://www.buildingsmart-tech.org/ifcOWL/IFC2X3_TC1
# prefix: bam

@prefix bam: <http://www.bam.com/bamNL_otl#> .
@prefix ifcowl: <http://www.buildingsmart-tech.org/ifcOWL/IFC2X3_TC1#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix query-rewriting: <http://bimsparql.org/query-rewriting#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sp: <http://spinrdf.org/sp#> .
@prefix spin: <http://spinrdf.org/spin#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.bam.com/bamNL_otl>
  rdf:type owl:Ontology ;
  owl:imports <http://bimsparql.org/pset#> ;
  owl:imports <http://bimsparql.org/query-rewriting#> ;
  owl:imports <http://www.buildingsmart-tech.org/ifcOWL/IFC2X3_TC1#> ;
  owl:versionInfo "Created with TopBraid Composer"^^xsd:string ;
.
bam:BalkonySpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UndefinedSpace ;
.
bam:BasicTechnicalSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:TechnicalSpace ;
.
bam:CafeSpace
  rdf:type owl:Class ;
  rdfs:label "Cafe space"^^xsd:string ;
  rdfs:subClassOf bam:HorecaSpace ;
.
bam:CanteenSpace
  rdf:type owl:Class ;
  rdfs:label "Canteen space"^^xsd:string ;
  rdfs:subClassOf bam:GatheringSpace ;
.
bam:Ceiling
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
bam:ChimneySpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UndefinedSpace ;
.
bam:Column
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
bam:CommercialSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
```

```

bam:CommonSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:FunctionalSpace ;
.
bam:CommunicationSpace
  rdf:type owl:Class ;
  rdfs:label "Communication space"^^xsd:string ;
  rdfs:subClassOf bam:OfficeSpace ;
.
bam:ConferenceSpace
  rdf:type owl:Class ;
  rdfs:label "Conference space"^^xsd:string ;
  rdfs:subClassOf bam:OfficeSpace ;
.
bam:ControlSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:TechnicalSpace ;
bam:CrawlSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UndefinedSpace ;
.
bam:DetentionSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:DiningRoom
  rdf:type owl:Class ;
  rdfs:label "Dining room"^^xsd:string ;
  rdfs:subClassOf bam:LivingSpace ;
.
bam:DisabledToiletSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:SanitarySpace ;
.
bam:HorecaKitchen
  rdf:type owl:Class ;
  rdfs:label "Horeca kitchen"^^xsd:string ;
  rdfs:subClassOf bam:HorecaSpace ;
.
bam:HorecaSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:HorizontalMovementSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:MovementSpace ;
.
bam:HotelRoom
  rdf:type owl:Class ;
  rdfs:label "Hotel room"^^xsd:string ;
  rdfs:subClassOf bam:HorecaSpace ;
.
bam:ICTTechnicalSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:TechnicalSpace ;
.
bam:Door
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
bam:EducationSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:ElectricalUtilitySpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:TechnicalSpace ;
.
bam:EngineRoomSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:TechnicalSpace ;
.
bam:EntranceSpace
  rdf:type owl:Class ;
  rdfs:label "Entrance
space"^^xsd:string ;
  rdfs:subClassOf bam:GatheringSpace ;
.
bam:EnvironmentallyControlledSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:ExpeditionSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:FunctionalSpace ;
.
bam:ExternalWall
  rdf:type owl:Class ;
  rdfs:subClassOf bam:Wall ;
.
bam:Floor
  rdf:type owl:Class ;
.
bam:FunctionalSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:Space ;
.
bam:GatheringSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:HealthCareSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;

```

```

bam:IndustrialSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:FunctionalSpace ;
.
bam:InternalWall
  rdf:type owl:Class ;
  rdfs:subClassOf bam:Wall ;
.
bam:JudicialSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:Kitchen
  rdf:type owl:Class ;
  rdfs:label "Kitchen"^^xsd:string ;
  rdfs:subClassOf bam:LivingSpace ;
.
bam:LeisureSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:LivingRoom
  rdf:type owl:Class ;
  rdfs:label "Living room"^^xsd:string ;
  rdfs:subClassOf bam:LivingSpace ;
bam:PrinterSpace
  rdf:type owl:Class ;
  rdfs:label "Printer space"^^xsd:string ;
  rdfs:subClassOf bam:OfficeSpace ;
.
bam:ReceptionSpace
  rdf:type owl:Class ;
  rdfs:label "Reception space"^^xsd:string ;
  rdfs:subClassOf bam:GatheringSpace ;
.
bam:ReligionSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:RestaurantSpace
  rdf:type owl:Class ;
  rdfs:label "Restaurant space"^^xsd:string ;
  rdfs:subClassOf bam:HorecaSpace ;
.
bam:SanitarySpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:Space ;
.
bam:Scullery
  rdf:type owl:Class ;
  rdfs:label "Scullery"^^xsd:string ;
  rdfs:subClassOf bam:LivingSpace ;
.
bam:SecuritySpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:TechnicalSpace ;
.
bam:SmokingSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:FunctionalSpace ;
.
bam:Space
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
bam:SportsSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:Stairs
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
bam:StorageSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:FunctionalSpace ;
.
bam:StudySpace
  rdf:type owl:Class ;
  rdfs:label "Study space"^^xsd:string ;
  rdfs:subClassOf bam:LivingSpace ;
.
bam:TechnicalSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:Space ;
.
bam:ToiletSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:SanitarySpace ;
.
bam:UndefinedSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:Space ;
.
bam:UserSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:Space ;
.
bam:VerticalMovementSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:MovementSpace ;
.

```



```

bam:ShaftSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UndefinedSpace ;
.
bam:ShowerSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:SanitarySpace ;
.
bam:SleepingSpace
  rdf:type owl:Class ;
  rdfs:label "Sleeping space"^^xsd:string ;
  rdfs:subClassOf bam:LivingSpace ;
.
bam:WaitingSpace
  rdf:type owl:Class ;
  rdfs:label "Waiting space"^^xsd:string ;
  rdfs:subClassOf bam:GatheringSpace ;
.
bam:Wall
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
bam:WashingSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:SanitarySpace ;
.
bam:WasteDisposalSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:FunctionalSpace ;
.
bam:Window
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
bam:WorkSpace
  rdf:type owl:Class ;
  rdfs:label "Working space"^^xsd:string ;
  rdfs:subClassOf bam:OfficeSpace ;
.
bam:LivingSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:MechanicalInstallationsSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:TechnicalSpace ;
.
bam:MeetingSpace
  rdf:type owl:Class ;
  rdfs:label "Meeting space"^^xsd:string ;
  rdfs:subClassOf bam:OfficeSpace ;
.
bam:MovementSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:Space ;
.
bam:NightClubSpace
  rdf:type owl:Class ;
  rdfs:label "Night club
space"^^xsd:string ;
  rdfs:subClassOf bam:HorecaSpace ;
.
bam:OfficeSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:UserSpace ;
.
bam:OtherSanitaryFunctionSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:SanitarySpace ;
.
bam:PantrySpace
  rdf:type owl:Class ;
  rdfs:label "Pantry space"^^xsd:string ;
  rdfs:subClassOf bam:OfficeSpace ;
.
bam:ParkingSpace
  rdf:type owl:Class ;
  rdfs:subClassOf bam:FunctionalSpace ;
bam:PersonalOfficeSpace
  rdf:type owl:Class ;
  rdfs:label "Personal office
space"^^xsd:string ;
  rdfs:subClassOf bam:OfficeSpace ;
.
bam:PresentationSpace
  rdf:type owl:Class ;
  rdfs:label "Presentation
space"^^xsd:string ;
  rdfs:subClassOf bam:GatheringSpace ;

```

## Annex H – SPIN Inference Rules

Firstly the objects are defined with a spin:construct rule in owl:thing. Secondly the properties are defined with use of a spin:magicProperty

*// Definition for a Functional Space//*

```
CONSTRUCT {  
    ?s a bam:FunctionalSpace .  
}  
WHERE {  
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .  
    ?s a ifcowl:IfcSpace .  
    ?y a ifcowl:IfcLabel .  
    ?y express:hasString "FunctionalSpace" .  
}
```

*// Definition for a User Space//*

```
CONSTRUCT {  
    ?s a bam:UserSpace .  
}  
WHERE {  
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .  
    ?s a ifcowl:IfcSpace .  
    ?y a ifcowl:IfcLabel .  
    ?y express:hasString "UserSpace" .  
}
```

*// Definition for a Movement Space//*

```
CONSTRUCT {  
    ?s a bam:MovementSpace .  
}  
WHERE {  
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .  
    ?s a ifcowl:IfcSpace .  
    ?y a ifcowl:IfcLabel .  
    ?y express:hasString "MovementSpace" .  
}
```

*// Definition for a Technical Space//*

```
CONSTRUCT {  
    ?s a bam:TechnicalSpace .  
}  
WHERE {  
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .  
    ?s a ifcowl:IfcSpace .  
    ?y a ifcowl:IfcLabel .  
    ?y express:hasString "TechnicalSpace" .  
}
```

*// Definition for a Sanitary Space//*

```
CONSTRUCT {  
    ?s a bam:SanitarySpace .  
}  
WHERE {  
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .  
    ?s a ifcowl:IfcSpace .  
    ?y a ifcowl:IfcLabel .  
    ?y express:hasString "SanitarySpace" .  
}
```

```
// Definition for an Undefined Space//
CONSTRUCT {
    ?s a bam:undefinedSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "UndefinedSpace" .
}

// Definition for an ToiletSpace//
CONSTRUCT {
    ?s a bam:ToiletSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "ToiletSpace" .
}

// Definition for an DisabledToiletSpace//
CONSTRUCT {
    ?s a bam:DisabledToiletSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "DisabledToiletSpace" .
}

// Definition for WashingSpace//
CONSTRUCT {
    ?s a bam:WashingSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "WashingSpace" .
}

// Definition for a ShowerSpace//
CONSTRUCT {
    ?s a bam:ShowerSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "ShowerSpace" .
}

// Definition for an OtherSanitarySpace//
CONSTRUCT {
    ?s a bam:OtherSanitarySpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "OtherSanitarySpace" .
}
```

```
// Definition for an CommercialSpace//
CONSTRUCT {
    ?s a bam:CommercialSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "CommercialSpace" .
}

// Definition for an DetentionSpace//
CONSTRUCT {
    ?s a bam:DetentionSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "DetentionSpace" .
}

// Definition for an EducationSpace//
CONSTRUCT {
    ?s a bam:EducationSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "EducationSpace" .
}

// Definition for an EnvironmetalyControlledSpace//
CONSTRUCT {
    ?s a bam:EnvironmentallyControlledSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "EnvironmentallyControlledSpace" .
}

// Definition for a HealthcareSpace//
CONSTRUCT {
    ?s a bam:HealthcareSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "HealthcareSpace" .
}

// Definition for a HorecaSpace//
CONSTRUCT {
    ?s a bam:HorecaSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "HorecaSpace" .
}
```

```
// Definition for an JudicialSpace//
CONSTRUCT {
    ?s a bam:JudicialSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "JudicialSpace" .
}

// Definition for an LeasureSpace//
CONSTRUCT {
    ?s a bam:LeasureSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "LeasureSpace" .
}

// Definition for a LivingSpace//
CONSTRUCT {
    ?s a bam:LivingSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "LivingSpace" .
}

// Definition for a MeetingSpace//
CONSTRUCT {
    ?s a bam:MeetingSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "MeetingSpace" .
}

// Definition for a OfficeSpace//
CONSTRUCT {
    ?s a bam:OfficeSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "OfficeSpace" .
}

// Definition for a ReligionSpace//
CONSTRUCT {
    ?s a bam:ReligionSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "ReligionSpace" .
}
```

```
// Definition for a SportsSpace//
CONSTRUCT {
    ?s a bam:SportsSpace .
}
WHERE {
    ?s ifcowl:longName_IfcSpatialStructureElement ?y .
    ?s a ifcowl:IfcSpace .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString "SportsSpace" .
}

// Definition for a Wall//
CONSTRUCT {
    ?s a bam:Wall .
}
WHERE {
    ?s a/rdfs:subClassOf* ifcowl:IfcWall .
}

// Definition for an Internal Wall //
CONSTRUCT {
    ?s a bam:InternalWall .
}
WHERE {
    ?s a/rdfs:subClassOf* ifcowl:IfcWall .
    ?s pset:isExternal false .
}

// Definition for an External Wall //
CONSTRUCT {
    ?s a bam:ExternalWall .
}
WHERE {
    ?s a/rdfs:subClassOf* ifcowl:IfcWall .
    ?s pset:isExternal true .
}

// Definition for a FireCompartmentWall //
CONSTRUCT {
    ?s a bam:Wall .
}
WHERE {
    ?s a/rdfs:subClassOf* ifcowl:IfcWall .
    ?s pset:Compartment true .
}

// Definition for a Window//
CONSTRUCT {
    ?s a bam:Window .
}
WHERE {
    ?s a ifcowl:IfcWindow .
}

// Definition for a PowerSocket//
CONSTRUCT {
    ?s a bam:PowerSocket .
}
WHERE {
    ?s ifcowl:name_IfcRoot ?y .
    ?s a ifcowl:IfcFlowTerminal .
    ?y a ifcowl:IfcLabel .
    ?y express:hasString ?n .
    FILTER CONTAINS(?n, "63_CGA_wcd") .
}
```

## Annex I – SPIN Constraint template queries

In the back end of the tool the following SPIN Constraint queries are used;

Comfort <i>Acoustic comfort</i>	<i>Value checking of a wall between two different spaces</i>	Spaces – wall – object relation; value
	<b>A wall between a User and a movement space must have a minimum acoustic rating of 65 dB</b>	
	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:internalWall .   ?s qrw:isBoundaryOf ?o .   ?s qrw:isBoundaryOf ?o2 .   ?o a bam:OfficeSpace .   ?o2 a bam:HorizontalMovementSpace .   FILTER NOT EXISTS {     ?s pset:acousticRating ?o3 .     FILTER (?o3 &lt; 65) .   } . } </pre>	
Availability <i>Object Occurrence</i>	<i>Object containment in a space checking</i>	Space – object Containment
	<b>Every unique user space must contain a power socket</b>	
	<pre> CONSTRUCT {   _:b0 a spin:ConstraintViolation .   _:b0 spin:violationRoot ?s .   _:b0 spin:violationLevel spin:Warning . } WHERE {   ?s a bam:OfficeSpace .   FILTER NOT EXISTS {     ?s qrw:hasContainedProduct ?o .     ?o a Bam:PowerSocket .   } . } </pre>	

These templates are translated in the following way in the .TTL (turtle) file;

Acoustic Comfort;

```

rdf:type sp:Construct ;
sp:templates (
  [
    sp:object spin:ConstraintViolation ;
    sp:predicate rdf:type ;
    sp:subject _:b90104 ;
  ]
  [
    sp:object [
      sp:varName "s"^^xsd:string ;
    ] ;
    sp:predicate spin:violationRoot ;
    sp:subject _:b90104 ;
  ]
)

```



```

    ]
    [
        sp:object spin:Warning ;
        sp:predicate spin:violationLevel ;
        sp:subject _:b90104 ;
    ]
) ;
sp:where (
    [
        sp:object [
            sp:varName "o"^^xsd:string ;
        ] ;
        sp:predicate query-rewriting:isBoundaryOf ;
        sp:subject [
            sp:varName "s"^^xsd:string ;
        ] ;
    ]
    [
        sp:object [
            sp:varName "o2"^^xsd:string ;
        ] ;
        sp:predicate query-rewriting:isBoundaryOf ;
        sp:subject [
            sp:varName "s"^^xsd:string ;
        ] ;
    ]
    [
        sp:object bam:InternalWall ;
        sp:predicate rdf:type ;
        sp:subject [
            sp:varName "s"^^xsd:string ;
        ] ;
    ]
    [
        sp:object bam:OfficeSpace ;
        sp:predicate rdf:type ;
        sp:subject [
            sp:varName "o"^^xsd:string ;
        ] ;
    ]
    [
        sp:object bam:HorizontalMovementSpace ;
        sp:predicate rdf:type ;
        sp:subject [
            sp:varName "o2"^^xsd:string ;
        ] ;
    ]
    [
        rdf:type sp:Filter ;
        sp:expression [
            rdf:type sp:notExists ;
            sp:elements (
                [
                    sp:object [
                        sp:varName "o3"^^xsd:string ;
                    ] ;
                ] ;
                sp:predicate <http://bimsparql.org/pset#acousticRating> ;
            ) ;
        ] ;
    ]
) ;

```

```

        sp:subject [
            sp:varName "s"^^xsd:string ;
        ] ;
    ]
    [
        rdf:type sp:Filter ;
        sp:expression [
            rdf:type sp:gt ;
            sp:arg1 [
                sp:varName "o3"^^xsd:string ;
            ] ;
            sp:arg2 69 ;
        ] ;
    ]
    ) ;
] ;
) .

```

Availability: Power socket;

```

bam:constraint_powersocket_1
    rdf:type sp:Construct ;
    sp:templates (
        [
            sp:object spin:ConstraintViolation ;
            sp:predicate rdf:type ;
            sp:subject _:b26654 ;
        ]
        [
            sp:object [
                sp:varName "s"^^xsd:string ;
            ] ;
            sp:predicate spin:violationRoot ;
            sp:subject _:b26654 ;
        ]
        [
            sp:object spin:Warning ;
            sp:predicate spin:violationLevel ;
            sp:subject _:b26654 ;
        ]
    ) ;
    sp:where (
        [
            rdf:type sp:TriplePath ;
            sp:object bam:Space ;
            sp:path [
                rdf:type sp:SeqPath ;
                sp:path1 rdf:type ;
                sp:path2 [
                    rdf:type sp:ModPath ;
                    sp:modMax -2 ;
                    sp:modMin 0 ;
                    sp:subPath rdfs:subClassOf ;
                ] ;
            ] ;
            sp:subject [

```

```

        sp:varName "s"^^xsd:string ;
    ] ;
]
[
    rdf:type sp:Filter ;
    sp:expression [
        rdf:type sp:notExists ;
        sp:elements (
            [
                sp:object [
                    sp:varName "o"^^xsd:string ;
                ] ;
                sp:predicate query-rewriting:hasContainedProduct ;
                sp:subject [
                    sp:varName "s"^^xsd:string ;
                ] ;
            ]
            [
                sp:object bam:PowerSocket ;
                sp:predicate rdf:type ;
                sp:subject [
                    sp:varName "o"^^xsd:string ;
                ] ;
            ]
        ) ;
    ] ;
]
) .

```

## Annex J – Tool Flowchart



## Annex K – Script Tool

Firstly the front end of the tool is described in the following script

Secondly the back end of the script is described as followed

### Backend script

```
package backend_constraints;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.topbraid.spin.inference.DefaultSPINRuleComparator;
import org.topbraid.spin.inference.SPINInferencesWithoutConstructor;
import org.topbraid.spin.inference.SPINRuleComparator;
import org.topbraid.spin.system.SPINModuleRegistry;
import org.topbraid.spin.util.CommandWrapper;
import org.topbraid.spin.util.SPINQueryFinder;
import org.topbraid.spin.vocabulary.SPIN;

import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.vocabulary.OWL;

public class Backend {

    Model model;
    Model ifcowl;
    OntModel spin;
    OntModel bam;

    static HashMap<String, List<Resource>> hashmap = new HashMap<String,
List<Resource>>();

    public static Backend init() {

        // input Spin files, bam OTL and IFCOWL

        Backend backend = new Backend();
        InputStream ifcowl =
Backend.class.getResourceAsStream("resources/IFC2X3_TC1.ttl");
        backend.ifcowl = ModelFactory.createDefaultModel();
        backend.ifcowl.read(ifcowl, null, "TTL");

        InputStream bamotl =
Backend.class.getResourceAsStream("resources/bam_OTL.ttl");
        backend.bam = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        backend.bam.read(bamotl, null, "TTL");
```

```

        InputStream pset =
Backend.class.getResourceAsStream("resources/experiment.ttl");
        backend.spin = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        backend.spin.read(pset, null, "TTL");

        InputStream qrw =
Backend.class.getResourceAsStream("resources/query_rewriting.ttl");
        backend.spin.read(qrw, null, "TTL");

        List<Resource> constraints = new ArrayList<Resource>();
        Resource constraint1 =
backend.spin.getResource("http://www.bam.com/bamNL_otl#constraint_acoustic_3");
        constraints.add(constraint1);
        hashmap.put("Comfort", constraints);
        SPINModuleRegistry.get().init();
        SPINModuleRegistry.get().registerAll(backend.spin, null);

        return backend;
    }

    public void readModel(String s) {

        // read file into memory

        model = ModelFactory.createDefaultModel();
        InputStream input;
        try {
            input = new FileInputStream(s);
            model.read(input, null, "TTL");
        } catch (FileNotFoundException e1) {
            e1.printStackTrace();
        }
    }

    private void addConstraint(Model spin, Resource constraint) {

        // add constraints out of the ttl file

        spin.add(OWL.Thing, SPIN.constraint, constraint);
    }

    public void selectConstraints(List<String> strings) {

        // selection of the constraints which can be connected to the buttons.

        for (String s : strings) {
            List<Resource> constraints = hashmap.get(s);
            for (Resource constraint : constraints) {
                addConstraint(bam, constraint);
            }
        }
    }

    public void execute() {
        // Run all constraints queries, when button start checker is executed

        long startInference = System.currentTimeMillis();
        Model newTriples = ModelFactory.createDefaultModel();
        OntModel ontModel =
ModelFactory.createOntologyModel(OntModelSpec.RDFS_MEM_RDFS_INF);
        ontModel.addSubModel(newTriples);
        ontModel.add(ifcowl);
        ontModel.add(model);
        ontModel.add(bam);
    }

```

```

        spin.add(bam);
        Map<Resource, List<CommandWrapper>> cls2Query =
SPINQueryFinder.getClass2QueryMap(spin, ontModel, SPIN.rule,
        false, false);
        SPINRuleComparator comparator = new DefaultSPINRuleComparator(ontModel);
        SPINInferencesWithoutConstructor.runWithoutConstructors(ontModel,
newTriples, cls2Query, null, null, false,
        SPIN.rule, comparator, null);
        long endInference = System.currentTimeMillis();
        float inferenceTime = (float) (endInference - startInference);
        System.out.println("Inference Time: " + inferenceTime / 1000);
        System.out.println("New Triples size: " + newTriples.size());
        String q = "prefix bam: <http://www.bam.com/bamNL_otl#> \n"+"prefix spin:
<"+SPIN.BASE_URI+"> \n"
        + "prefix query-rewriting: <http://bimsparql.org/query-
rewriting#>\n"
        + "prefix pset:<http://bimsparql.org/pset#>\n" + "CONSTRUCT
{\n" + " _:b0 a spin:ConstraintViolation .\n"
        + " _:b0 spin:violationRoot ?s .\n" + " _:b0 spin:violationLevel
spin:Warning .\n" + "}\n" + "WHERE {\n"
        + "?s query-rewriting:isBoundaryOf ?o .\n" + "?s query-
rewriting:isBoundaryOf ?o2 .\n"
        + "?s a bam:InternalWall .\n" + "?o a bam:MeetingSpace .\n" +
"?o2 a bam:HorizontalMovementSpace .\n"
        + "FILTER NOT EXISTS {\n" + "?s pset:acousticRating ?o3 .\n" +
"FILTER (?o3 > 50) .\n" + "}\n" + "}"
        Query query = QueryFactory.create(q);
        QueryExecution qexec = QueryExecutionFactory.create(query, ontModel);
        Model output = qexec.execConstruct();
        FileOutputStream out;
        try {
            out = new FileOutputStream (new File (new File
(System.getProperty("user.home"), "/Desktop"), "output.ttl"));
            output.write(out, "TTL");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
}

```

## Front end script

### Window 1

```

package FrontEnd;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

import javax.swing.JLabel;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JTextPane;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Color;

```

```

public class Image extends JFrame {

    private JPanel contentPane;

    /**

```

```

    * Launch the application.
    */
    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Image frame = new Image();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Image() {
        setTitle("Requirements Checker");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(700, 380, 450, 300);
        contentPane = new JPanel();
        contentPane.setBackground(Color.WHITE);
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblNewLabel = new JLabel("Requirements Checker");
        lblNewLabel.setIcon(new
ImageIcon("C:\\Users\\l.moonen\\workspace\\AutomatedVerificationTool\\Images\\bam.jpg"));
        lblNewLabel.setBounds(23, 21, 175, 82);
        contentPane.add(lblNewLabel);

        JLabel lblNewLabel_1 = new JLabel("");
        lblNewLabel_1.setIcon(new
ImageIcon("C:\\Users\\l.moonen\\workspace\\AutomatedVerificationTool\\Images\\logo-tue-
150x150.png"));
        lblNewLabel_1.setBounds(245, 27, 158, 66);
        contentPane.add(lblNewLabel_1);

        JButton btnStartRequirementsChecker = new JButton("Start Requirements
Checker");
        btnStartRequirementsChecker.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                dispose();
                SelectionOfRequirements nw = new SelectionOfRequirements();
                SelectionOfRequirements.NewScreen();
            }
        });
        btnStartRequirementsChecker.setBounds(120, 190, 193, 46);
        contentPane.add(btnStartRequirementsChecker);

        JTextPane txtpnAutomatedVerificationOf = new JTextPane();
        txtpnAutomatedVerificationOf.setEditable(false);
        txtpnAutomatedVerificationOf.setText("Automated Verification of Client
specific Requirements\r\nA Prototype By Luuk Thomas Moonen\r\n2016");
        txtpnAutomatedVerificationOf.setBounds(120, 114, 193, 65);
        contentPane.add(txtpnAutomatedVerificationOf);
    }
}

```



## Window 2

```
package FrontEnd;

import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Color;

import java.io.File;

import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JButton;
import javax.swing.JTextField;
import javax.swing.JTextPane;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.JFileChooser;

public class SelectionOfRequirements {

    private JFrame frmRequirementsChecker;
    private static JTextField textField;
    private File selectedFile;

    /**
     * Launch the application.
     */
    public static void NewScreen() {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    SelectionOfRequirements window = new
SelectionOfRequirements();
                    window.frmRequirementsChecker.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public SelectionOfRequirements() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */

    private void initialize() {
        frmRequirementsChecker = new JFrame();
        frmRequirementsChecker.getContentPane().setBackground(Color.WHITE);
        frmRequirementsChecker.setTitle("Requirements Checker");
        frmRequirementsChecker.setBounds(700, 380, 450, 300);
        frmRequirementsChecker.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frmRequirementsChecker.getContentPane().setLayout(null);

        JButton btnLoadFile = new JButton("Browse");
        btnLoadFile.setBounds(285, 98, 89, 23);
        frmRequirementsChecker.getContentPane().add(btnLoadFile);
    }
}
```

```

        btnLoadFile.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                JFileChooser fileChooser = new JFileChooser();
                FileNameExtensionFilter filter = new
FileNameExtensionFilter("IFC or TTL Files", "TTL", "IFC");
                fileChooser.setFileFilter(filter);
                fileChooser.setCurrentDirectory(new
File(System.getProperty("user.home")));
                int result = fileChooser.showOpenDialog(null);
                if (result == JFileChooser.APPROVE_OPTION) {
                    selectedFile = fileChooser.getSelectedFile();
                    textField.setText(selectedFile.getAbsolutePath());
                }
            }

        });

        textField = new JTextField();
        textField.setBounds(67, 99, 208, 20);
        frmRequirementsChecker.getContentPane().add(textField);
        textField.setColumns(10);

        JTextPane txtpnAnIfcFile = new JTextPane();
        txtpnAnIfcFile.setEditable(false);
        txtpnAnIfcFile.setText("Select an IFC or a TTL file to continue");
        txtpnAnIfcFile.setBounds(67, 143, 251, 48);
        frmRequirementsChecker.getContentPane().add(txtpnAnIfcFile);

        JButton btnNext = new JButton("Next");
        btnNext.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                if (selectedFile != null) {
                    RequirementsSelection nw = new
RequirementsSelection();
                    nw.NewScreen(textField.getText());
                    frmRequirementsChecker.dispose();

                } else {
                    JOptionPane.showMessageDialog(frmRequirementsChecker,
"No IFC file has been loaded.");
                }

            }

        });
        btnNext.setBounds(335, 228, 89, 23);
        frmRequirementsChecker.getContentPane().add(btnNext);

        JButton button = new JButton("Previous");
        button.setBounds(236, 228, 89, 23);
        frmRequirementsChecker.getContentPane().add(button);
    }
}

```

### Window 3

```

package FrontEnd;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JCheckBox;
import javax.swing.JTextPane;
import javax.swing.JButton;

```

```

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JPanel;
import javax.swing.JTextField;
import java.awt.Color;

public class RequirementsSelection {

    private JFrame frmRequirementsChecker;
    private JTextField txtComfortRequirements;
    static String textField;

    /**
     * Launch the application.
     */
    public static void NewScreen(String textfield) {
        textField = textfield;
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    RequirementsSelection window = new
RequirementsSelection();
                    window.frmRequirementsChecker.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public RequirementsSelection() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frmRequirementsChecker = new JFrame();
        frmRequirementsChecker.getContentPane().setBackground(Color.WHITE);
        frmRequirementsChecker.setTitle("Requirements Checker");
        frmRequirementsChecker.setBounds(600, 300, 653, 536);
        frmRequirementsChecker.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frmRequirementsChecker.getContentPane().setLayout(null);

        final JCheckBox chckbxNewCheckBox = new JCheckBox("Comfort");
        chckbxNewCheckBox.setSelected(true);
        chckbxNewCheckBox.setBounds(113, 111, 97, 23);
        frmRequirementsChecker.getContentPane().add(chckbxNewCheckBox);

        JCheckBox chckbxNewCheckBox_1 = new JCheckBox("Accessibility");
        chckbxNewCheckBox_1.setBounds(113, 137, 97, 23);
        frmRequirementsChecker.getContentPane().add(chckbxNewCheckBox_1);

        JCheckBox chckbxFunctionality = new JCheckBox("Functionality");
        chckbxFunctionality.setBounds(113, 163, 97, 23);
        frmRequirementsChecker.getContentPane().add(chckbxFunctionality);

        final JCheckBox chckbxAvailability = new JCheckBox("Availability");
        chckbxAvailability.setBounds(113, 189, 97, 23);
        frmRequirementsChecker.getContentPane().add(chckbxAvailability);

        JCheckBox chckbxSafety = new JCheckBox("Safety");
    }
}

```

```

chckbxSafety.setBounds(113, 215, 97, 23);
frmRequirementsChecker.getContentPane().add(chckbxSafety);

JCheckBox chckbxSecurity = new JCheckBox("Security");
chckbxSecurity.setBounds(113, 241, 97, 23);
frmRequirementsChecker.getContentPane().add(chckbxSecurity);

JCheckBox chckbxQuality = new JCheckBox("Quality");
chckbxQuality.setBounds(113, 267, 97, 23);
frmRequirementsChecker.getContentPane().add(chckbxQuality);

JCheckBox chckbxSpatiality = new JCheckBox("Spatiality");
chckbxSpatiality.setBounds(113, 293, 97, 23);
frmRequirementsChecker.getContentPane().add(chckbxSpatiality);

JCheckBox chckbxStrength = new JCheckBox("Strength");
chckbxStrength.setBounds(113, 319, 97, 23);
frmRequirementsChecker.getContentPane().add(chckbxStrength);

JCheckBox chckbxCosts = new JCheckBox("Costs");
chckbxCosts.setBounds(113, 345, 97, 23);
frmRequirementsChecker.getContentPane().add(chckbxCosts);

JTextPane txtpnInThisMenu = new JTextPane();
txtpnInThisMenu.setEditable(false);
txtpnInThisMenu.setText("In this menu the requirement types must be chosen
to define which\r\nrequirements will be checked automatically");
txtpnInThisMenu.setBounds(79, 45, 417, 46);
frmRequirementsChecker.getContentPane().add(txtpnInThisMenu);

JButton button = new JButton("Next");
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(chckbxNewCheckBox.isSelected() ||
chckbxAvailability.isSelected()){
            ObjectSelection nw = new ObjectSelection();
            ObjectSelection.NewScreen(textField);
            frmRequirementsChecker.dispose();
        } else {
            JOptionPane.showMessageDialog(frmRequirementsChecker,
"TODO");
        }
    }
});
button.setBounds(538, 464, 89, 23);
frmRequirementsChecker.getContentPane().add(button);

JButton button_1 = new JButton("Previous");
button_1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
    }
});
button_1.setBounds(439, 464, 89, 23);
frmRequirementsChecker.getContentPane().add(button_1);

JButton btnRequirements = new JButton("Requirements");
btnRequirements.setBounds(216, 111, 126, 23);
frmRequirementsChecker.getContentPane().add(btnRequirements);

JButton button_2 = new JButton("Requirements");
button_2.setBounds(216, 137, 126, 23);
frmRequirementsChecker.getContentPane().add(button_2);

JButton button_3 = new JButton("Requirements");
button_3.setBounds(216, 163, 126, 23);
frmRequirementsChecker.getContentPane().add(button_3);

```

```

        JButton button_4 = new JButton("Requirements");
        button_4.setBounds(216, 189, 126, 23);
        frmRequirementsChecker.getContentPane().add(button_4);

        JButton button_5 = new JButton("Requirements");
        button_5.setBounds(216, 215, 126, 23);
        frmRequirementsChecker.getContentPane().add(button_5);

        JButton button_6 = new JButton("Requirements");
        button_6.setBounds(216, 241, 126, 23);
        frmRequirementsChecker.getContentPane().add(button_6);

        JButton button_7 = new JButton("Requirements");
        button_7.setBounds(216, 267, 126, 23);
        frmRequirementsChecker.getContentPane().add(button_7);

        JButton button_8 = new JButton("Requirements");
        button_8.setBounds(216, 293, 126, 23);
        frmRequirementsChecker.getContentPane().add(button_8);

        JButton button_9 = new JButton("Requirements");
        button_9.setBounds(216, 319, 126, 23);
        frmRequirementsChecker.getContentPane().add(button_9);

        JButton button_10 = new JButton("Requirements");
        button_10.setBounds(216, 345, 126, 23);
        frmRequirementsChecker.getContentPane().add(button_10);

        JPanel panel = new JPanel();
        panel.setBounds(393, 111, 221, 267);
        frmRequirementsChecker.getContentPane().add(panel);
        panel.setLayout(null);

        JCheckBox chckbxHeating = new JCheckBox("Heating Requirements");
        chckbxHeating.setBounds(6, 36, 167, 23);
        panel.add(chckbxHeating);

        JCheckBox chckbxNewCheckBox_2 = new JCheckBox("Air Ventilation
Requirements");
        chckbxNewCheckBox_2.setBounds(6, 88, 167, 23);
        panel.add(chckbxNewCheckBox_2);

        JCheckBox chckbxNewCheckBox_3 = new JCheckBox("Cooling Requirements");
        chckbxNewCheckBox_3.setBounds(6, 62, 167, 23);
        panel.add(chckbxNewCheckBox_3);

        JCheckBox chckbxAudiologicRequirements = new JCheckBox("Audiologic
Requirements");
        chckbxAudiologicRequirements.setSelected(true);
        chckbxAudiologicRequirements.setBounds(6, 114, 167, 23);
        panel.add(chckbxAudiologicRequirements);

        JCheckBox chckbxLightingRequirements = new JCheckBox("Lighting
Requirements");
        chckbxLightingRequirements.setBounds(6, 140, 167, 23);
        panel.add(chckbxLightingRequirements);

        txtComfortRequirements = new JTextField();
        txtComfortRequirements.setEditable(false);
        txtComfortRequirements.setText("Comfort requirements");
        txtComfortRequirements.setBounds(6, 9, 125, 20);
        panel.add(txtComfortRequirements);
        txtComfortRequirements.setColumns(10);

```

```

        JCheckBox chckbxMoistureRequirements = new JCheckBox("Moisture
Requirements");
        chckbxMoistureRequirements.setBounds(6, 166, 167, 23);
        panel.add(chckbxMoistureRequirements);

        JCheckBox chckbxReflectionRequirements = new JCheckBox("Reflection
Requirements");
        chckbxReflectionRequirements.setBounds(6, 192, 167, 23);
        panel.add(chckbxReflectionRequirements);
    }
}

```

#### Window 4

```

package FrontEnd;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JCheckBox;
import javax.swing.JTextPane;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Color;

public class ObjectSelection {

    private JFrame frmRe;
    static String textField;

    /**
     * Launch the application.
     */
    public static void NewScreen(String textfield) {
        textField = textfield;
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    ObjectSelection window = new ObjectSelection();
                    window.frmRe.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public ObjectSelection() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frmRe = new JFrame();
        frmRe.getContentPane().setBackground(Color.WHITE);
        frmRe.setTitle("Requirements Checker");
        frmRe.setBounds(600, 300, 586, 501);
    }
}

```

```

frmRe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frmRe.getContentPane().setLayout(null);

JCheckBox chckbxWalls = new JCheckBox("Externall Walls");
chckbxWalls.setBounds(162, 92, 142, 23);
frmRe.getContentPane().add(chckbxWalls);

JCheckBox chckbxInternalWalls = new JCheckBox("Internal Walls");
chckbxInternalWalls.setSelected(true);
chckbxInternalWalls.setBounds(162, 118, 142, 23);
frmRe.getContentPane().add(chckbxInternalWalls);

JCheckBox chckbxFloors = new JCheckBox("Floors");
chckbxFloors.setBounds(162, 144, 142, 23);
frmRe.getContentPane().add(chckbxFloors);

JCheckBox chckbxStairs = new JCheckBox("Stairs");
chckbxStairs.setBounds(162, 170, 142, 23);
frmRe.getContentPane().add(chckbxStairs);

JCheckBox chckbxRoofs = new JCheckBox("Roofs");
chckbxRoofs.setBounds(162, 196, 142, 23);
frmRe.getContentPane().add(chckbxRoofs);

JCheckBox chckbxConstruction = new JCheckBox("Construction");
chckbxConstruction.setBounds(162, 222, 142, 23);
frmRe.getContentPane().add(chckbxConstruction);

JCheckBox chckbxExternalOpenings = new JCheckBox("Openings");
chckbxExternalOpenings.setBounds(162, 248, 142, 23);
frmRe.getContentPane().add(chckbxExternalOpenings);

JTextPane txtpnInThisMenu = new JTextPane();
txtpnInThisMenu.setEditable(false);
txtpnInThisMenu.setText("In this menu the object types must be chosen for
the selected requirements. ");
txtpnInThisMenu.setBounds(10, 11, 417, 46);
frmRe.getContentPane().add(txtpnInThisMenu);

JButton button = new JButton("Previous");
button.setBounds(370, 430, 89, 23);
frmRe.getContentPane().add(button);

JButton button_1 = new JButton("Next");
button_1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        frmRe.dispose();
        ValueInput nw = new ValueInput ();
        ValueInput.NewScreen(textField);

    }
});
button_1.setBounds(469, 430, 89, 23);
frmRe.getContentPane().add(button_1);

JCheckBox chckbxInternalOpenings = new JCheckBox("Mechanical Facilities");
chckbxInternalOpenings.setBounds(162, 274, 142, 23);
frmRe.getContentPane().add(chckbxInternalOpenings);

JCheckBox chckbxElectronicFacilities = new JCheckBox("Electronic
Facilities");
chckbxElectronicFacilities.setBounds(162, 300, 142, 23);
frmRe.getContentPane().add(chckbxElectronicFacilities);

```

```
JCheckBox chckbxFoundationFacilities = new JCheckBox("Foundation
Facilities");
chckbxFoundationFacilities.setBounds(162, 326, 142, 23);
frmRe.getContentPane().add(chckbxFoundationFacilities);

JButton btnInSpace = new JButton("In Space");
btnInSpace.setBounds(370, 92, 162, 23);
frmRe.getContentPane().add(btnInSpace);

JPanel panel = new JPanel();
panel.setBounds(352, 78, 208, 288);
frmRe.getContentPane().add(panel);
panel.setLayout(null);

JRadioButton rdbtnUserSpace = new JRadioButton("User Space");
rdbtnUserSpace.setBounds(23, 50, 137, 23);
panel.add(rdbtnUserSpace);

JRadioButton rdbtnFunctionalSpace = new JRadioButton("Functional Space");
rdbtnFunctionalSpace.setBounds(23, 76, 137, 23);
panel.add(rdbtnFunctionalSpace);

JRadioButton rdbtnSanitarySpace = new JRadioButton("Sanitary Space");
rdbtnSanitarySpace.setBounds(23, 102, 137, 23);
panel.add(rdbtnSanitarySpace);

JRadioButton rdbtnTechnicalSpace = new JRadioButton("Movement Space");
rdbtnTechnicalSpace.setBounds(23, 128, 137, 23);
panel.add(rdbtnTechnicalSpace);

JRadioButton rdbtnUndefinedSpace = new JRadioButton("Undefined Space");
rdbtnUndefinedSpace.setBounds(23, 154, 137, 23);
panel.add(rdbtnUndefinedSpace);

JRadioButton rdbtnTechnicalSpace_1 = new JRadioButton("Technical Space");
rdbtnTechnicalSpace_1.setBounds(23, 180, 137, 23);
panel.add(rdbtnTechnicalSpace_1);

JPanel panel_1 = new JPanel();
panel_1.setLayout(null);
panel_1.setBounds(154, 78, 162, 288);
frmRe.getContentPane().add(panel_1);
}
}
```

### Window 5

```
package FrontEnd;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JTextPane;
import javax.swing.JButton;
import javax.swing.JEditorPane;
import javax.swing.JTextField;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import java.awt.event.ActionEvent;
import java.awt.Color;
import javax.swing.UIManager;

import backend_constraints.Backend;

public class ValueInput {
```



```

private JFrame frmRequirementsChecker;
private JTextField txtInputValue;
private JTextField txtRequirementStatement;
public static Backend backend;
static String textField;

/**
 * Launch the application.
 */
public static void NewScreen(String textfield) {
    textField = textfield;
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                ValueInput window = new ValueInput();
                window.frmRequirementsChecker.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/**
 * Create the application.
 */
public ValueInput() {
    initialize();
}

/**
 * Initialize the contents of the frame.
 */
private void initialize() {
    frmRequirementsChecker = new JFrame();
    frmRequirementsChecker.getContentPane().setBackground(Color.WHITE);
    frmRequirementsChecker.setTitle("Requirements Checker");
    frmRequirementsChecker.setBounds(600, 300, 585, 503);
    frmRequirementsChecker.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frmRequirementsChecker.getContentPane().setLayout(null);

    JTextPane txtpnInThisMenu = new JTextPane();
    txtpnInThisMenu.setText("In this menu the values of the templates must be
selected per requirement Check");
    txtpnInThisMenu.setBounds(10, 11, 500, 40);
    txtpnInThisMenu.setEditable(false);
    frmRequirementsChecker.getContentPane().add(txtpnInThisMenu);

    JTextPane txtSelectedRequirement = new JTextPane();
    txtSelectedRequirement.setText("The following requirement is built up with
the templates and with the selected elements ");
    txtSelectedRequirement.setBounds(10, 45, 550, 40);
    txtSelectedRequirement.setEditable(false);
    frmRequirementsChecker.getContentPane().add(txtSelectedRequirement);

    JButton button = new JButton("Previous");
    button.setBounds(339, 430, 89, 23);
    frmRequirementsChecker.getContentPane().add(button);

    JButton btnStartCheck = new JButton("Start Check");
    btnStartCheck.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            ReportOutcome nw = new ReportOutcome();
            frmRequirementsChecker.dispose();
        }
    });
}

```

```

JFrame window2 = new JFrame();

window2.setVisible(true);
window2.getContentPane().setBackground(Color.WHITE);
window2.setTitle("Requirements Checker");
window2.setBounds(100, 100, 300, 200);
window2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
window2.getContentPane().setLayout(null);

JTextPane txtpnExecution = new JTextPane();
txtpnExecution.setText("Please wait while checks are executed");

txtpnExecution.setBounds(100, 100, 300, 200);
window2.getContentPane().add(txtpnExecution);

List<String> strings=new ArrayList<String>();
strings.add("Comfort");
backend = Backend.init();
backend.readModel(textField);
backend.selectConstraints(strings);
backend.execute();

window2.dispose();

nw.NewScreen();
    }
});
btnStartCheck.setBounds(440, 430, 118, 23);
frmRequirementsChecker.getContentPane().add(btnStartCheck);

JEditorPane editorPane = new JEditorPane();
editorPane.setBackground(UIManager.getColor("Button.background"));
editorPane.setBounds(10, 153, 169, 46);
frmRequirementsChecker.getContentPane().add(editorPane);

txtInputValue = new JTextField();
txtInputValue.setText("Input value for X");
txtInputValue.setBounds(187, 153, 169, 46);
frmRequirementsChecker.getContentPane().add(txtInputValue);
txtInputValue.setColumns(10);

txtRequirementStatement = new JTextField();
txtRequirementStatement.setText("An Internal Wall between a user and a
movement space must have an acoustic rating of at least X");
txtRequirementStatement.setBounds(10, 92, 400, 50);
frmRequirementsChecker.getContentPane().add(txtRequirementStatement);
txtRequirementStatement.setColumns(10);
    }
}

```

### Window 6

```

package FrontEnd;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JTextPane;
import java.awt.Font;

public class ReportOutcome {

    private JFrame frmRequirementsChecker;

```

```

/**
 * Launch the application.
 */
public static void NewScreen() {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                ReportOutcome window = new ReportOutcome();
                window.frmRequirementsChecker.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/**
 * Create the application.
 */
public ReportOutcome() {
    initialize();
}

/**
 * Initialize the contents of the frame.
 */
private void initialize() {
    frmRequirementsChecker = new JFrame();
    frmRequirementsChecker.setTitle("Requirements Checker");
    frmRequirementsChecker.setBounds(600, 300, 400, 250);
    frmRequirementsChecker.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frmRequirementsChecker.getContentPane().setLayout(null);

    JTextPane txtpnQuerySuccesfull = new JTextPane();
    txtpnQuerySuccesfull.setEditable(false);
    txtpnQuerySuccesfull.setFont(new Font("Tahoma", Font.BOLD, 16));
    txtpnQuerySuccesfull.setText("Check succesfull!");
    txtpnQuerySuccesfull.setBounds(103, 45, 151, 25);
    frmRequirementsChecker.getContentPane().add(txtpnQuerySuccesfull);

    JTextPane txtpnOutputGeneratedAt = new JTextPane();
    txtpnOutputGeneratedAt.setEditable(false);
    txtpnOutputGeneratedAt.setText("Output generated and saved to desktop");
    txtpnOutputGeneratedAt.setBounds(16, 102, 351, 25);
    frmRequirementsChecker.getContentPane().add(txtpnOutputGeneratedAt);

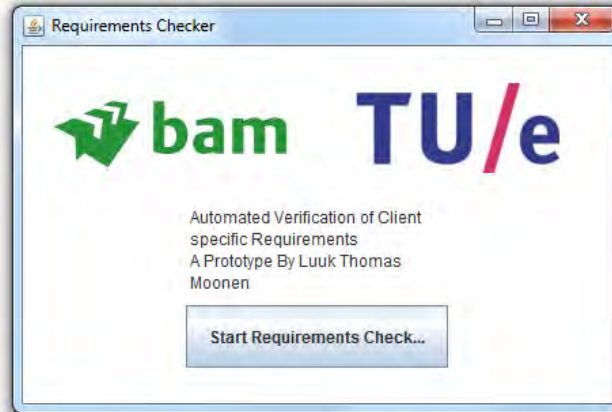
    JTextPane txtpnInTotal = new JTextPane();
    txtpnInTotal.setEditable(false);
    txtpnInTotal.setFont(new Font("Tahoma", Font.BOLD, 11));
    txtpnInTotal.setText("In total 4 bam:InternalWalls are non-complying with
the requirement. See output for more information.");
    txtpnInTotal.setBounds(16, 138, 351, 34);
    frmRequirementsChecker.getContentPane().add(txtpnInTotal);
}
}

```

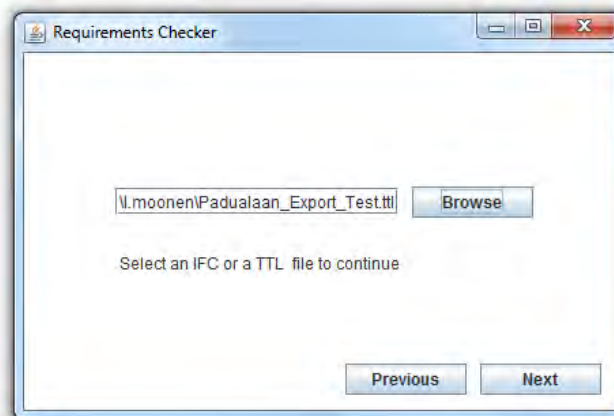
## Annex L – Application windows

In this annex the application windows of the interface of the requirements checker can be seen.

### Window 1 – Starting the application



### Window 2 – Selecting the to be checked file



### Window 3 – Selecting the need and the accompanying requirement type

In this menu the requirement types must be chosen to define which requirements will be checked automatically

<input checked="" type="checkbox"/> Comfort	Requirements
<input type="checkbox"/> Accessibility	Requirements
<input type="checkbox"/> Functionality	Requirements
<input type="checkbox"/> Availability	Requirements
<input type="checkbox"/> Safety	Requirements
<input type="checkbox"/> Security	Requirements
<input type="checkbox"/> Quality	Requirements
<input type="checkbox"/> Spatiality	Requirements
<input type="checkbox"/> Strength	Requirements
<input type="checkbox"/> Costs	Requirements

Comfort requirements

- ☐ Heating Requirements
- ☐ Cooling Requirements
- ☐ Air Ventilation Require...
- ☒ Audiologic Requirements
- ☐ Lighting Requirements
- ☐ Moisture Requirements
- ☐ Reflection Requirements

Previous Next

### Window 4 – Selecting the elements of the to be checked requirement

In this menu the object types must be chosen for the selected requirements.

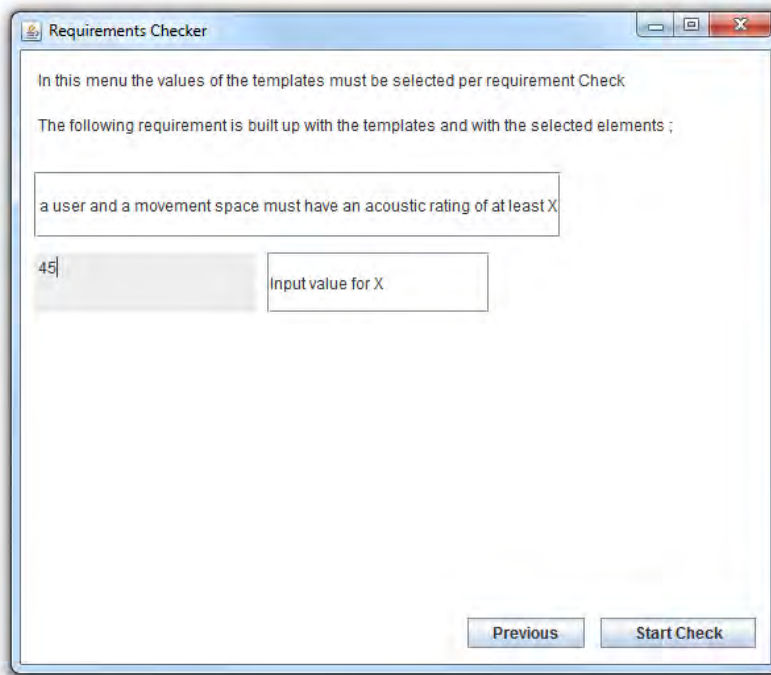
<input type="checkbox"/> External Walls
<input checked="" type="checkbox"/> Internal Walls
<input type="checkbox"/> Floors
<input type="checkbox"/> Stairs
<input type="checkbox"/> Roofs
<input type="checkbox"/> Construction
<input type="checkbox"/> Openings
<input type="checkbox"/> Mechanical Faciliti...
<input type="checkbox"/> Electronic Facilities
<input type="checkbox"/> Foundation Facilities

In Space

- ☒ User Space
- ☐ Functional Space
- ☐ Sanitary Space
- ☒ Movement Space
- ☐ Undefined Space
- ☐ Technical Space

Previous Next

Window 5 – Statement of the selected requirement and input of required value



Requirements Checker

In this menu the values of the templates must be selected per requirement Check

The following requirement is built up with the templates and with the selected elements ;

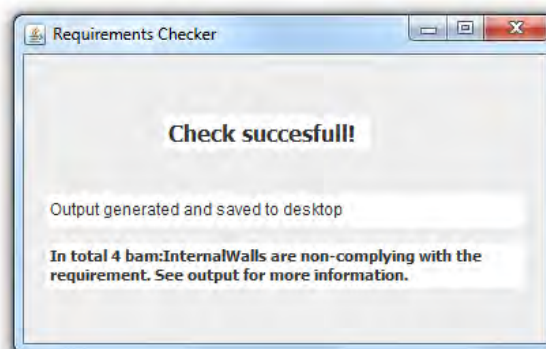
a user and a movement space must have an acoustic rating of at least X

45

Input value for X

Previous Start Check

Window 6 – outcome of the check



Requirements Checker

**Check succesfull!**

Output generated and saved to desktop

**In total 4 bam:InternalWalls are non-complying with the requirement. See output for more information.**

## Annex M – Flowchart Template selection

