HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Software Business and Engineering Institute

Matti Kannala

Escape Route Analysis Based on Building Information Models: Design and Implementation

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Helsinki, December 7, 2005

Supervisor: Professor Reijo Sulonen
Instructor: M.Sc. (Tech.) Pasi Paasiala

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Matti Kannala
Title of the thesis:	Escape Route Analysis Based on Building Information Models: Design and Implementation
Date:	December 7, 2005 **Number of Pages:** 52
Department:	Department of Computer Science and Engineering
Professorship:	T-76 Development of Digital Products
Supervisor:	Prof. Reijo Sulonen
Instructor:	M.Sc. (Tech.) Pasi Paasiala

Current development in building design is leading to a situation where buildings are designed with three-dimensional intelligent objects. Since these designs can also be shared using open standards, the possibility has emerged of using the designs to automate tasks that have traditionally been manual. One such task is escape route analysis, which is performed against building fire codes. This thesis defines the set of information that is needed to automatically perform such analysis, the methods for performing the analysis, and the means for presenting the results of the analysis.

Within this thesis, an escape route analysis module is implemented and included in the Solibri Model Checker software product. The work consists of the fundamental activities of software process: study, definition, design, implementation, and testing. The definition is based on studies of building information modeling (BIM) and escape route analysis. The module was designed using the UML modeling language. The quality of the implementation was assured by continuous automatic testing and by tools that analyze the source code. Finally the module was evaluated against the defined requirements.

The work shows that current building information models have adequate information to automatically check escape routes against the most essential regulations of the building fire codes. Currently there remains a lot of variation in the information content in the building information models. Much of the information is implicit. It appears only in the geometry of building components. In order to process geometric information the thesis introduces several geometry algorithms. The three most notable of these are: an algorithm for decreasing or increasing polygons in size, an algorithm for merging spaces, and an algorithm for creating compartments.

Keywords: algorithm, automatic testing, building information modeling, escape route, fire code, geometry

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Matti Kannala	
Työn nimi:	Rakennusten tuotemalleihin perustuvan poistumistieanalyysin suunnittelu ja toteutus	
Päivämäärä:	27.4.20065	**Sivuja:** 52
Osasto:	Tietotekniikan osasto	
Professuuri:	T-76 Digitaalisten tuotteiden kehittäminen	
Työn valvoja:	Prof. Reijo Sulonen	
Työn ohjaaja:	DI Pasi Paasiala	

Rakennusten suunnittelu on kehittymässä suuntaan, jossa rakennukset suunnitellaan kolmiulotteisilla älykkäillä objekteilla. Koska nämä suunnitelmat voidaan jakaa myös avointen standardien avulla, on suunnitelmien avulla mahdollista automatisoida tehtäviä, jotka on perinteisesti tehty käsin. Yksi sellainen tehtävä on rakennusmääräysten mukainen poistumistieanalyysi. Tämä opinnäytetyö määrittelee informaation, jota tarvitaan analyysin suorittamiseen, menetelmät analyysin tekemiseen ja tavat analyysin tulosten esittämiseen.

Opinnäytetyössä on toteutettu poistumistieanalyysimoduuli Solibri Model Checker-tuotteeseen. Työ koostuu tavanomaisista ohjelmistoprosessin toiminnoista: tutkiminen, määrittely, suunnittelu, toteutus ja testaus. Työssä tutkittiin tuotemallinnusta ja poistumistieanalyysia, minkä perusteella tehtiin määrittely. Moduuli suunniteltiin UML-kuvauskielellä. Toteutuksen laatu varmistettiin jatkuvalla automaattitestauksella ja lähdekoodin analysointityökaluilla. Lopulta moduuli arvioitiin määriteltyjen vaatimusten avulla.

Työ osoittaa, että nykyiset rakennusten tuotemallit sisältävät riittävän informaation automaattiseen tärkeimpien paloturvallisuusmääräysten mukaiseen poistumisteiden tarkastamiseen. Koska rakennusten tuotemallien informaation sisältö vaihtelee paljon, on suuri osa informaatiosta pääteltävä rakennusosien geometrian pohjalta. Geometrisen informaation käsittelyä varten työssä esitellään useita geometria-algoritmeja. Kolme merkittävintä niistä ovat algoritmi monikulmion pienentämiseen tai suurentamiseen, algoritmi tilojen yhdistämiseen ja algoritmi osastojen luomiseen.

Avainsanat: algoritmi, automaattitestaus, geometria, palomääräykset, poistumistie, tuotemallinnus

Acknowledgements

This work has been carried out at Solibri Oy. I would like to thank my employer Solibri Oy for providing me with an interesting topic for my thesis. I also wish to thank all the people at Solibri Oy who have helped during this thesis.

My gratitude goes to Professor Reijo Sulonen and my instructor Pasi Paasiala for guidance during this work. Both of them have provided helpful comments and suggestions during the writing process.

I wish to thank all those people who have somehow contributed to this thesis.

Finally, I would like to thank my wife Minna who encouraged and helped me in completing this work.

Helsinki, 27 April 2006

Matti Kannala

Contents

List of Abbreviations

AEC	Architecture, Engineering and Construction
AIA	American Institute of Architects
API	Application Program Interface
CAD	Computer-Aided Design/Drafting
CAG	Constructive Area Geometry
CCCC	C and C++ Code Counter
CSG	Constructive Solid Geometry
CSM	Constraint Set Manager
BIM	Building Information Modeling
BSP	Binary Space-Partitioning
FM	Facilities Management
GSA	General Services Administration
IAI	International Alliance of Interoperability
IDE	Integrated Development Environment
IFC	Industry Foundation Classes
NFPA	National Fire Protection Association
NURBS	Non-Uniform Rational B-Splines
OOCAD	Object-Oriented CAD
PC	Personal Computer
PDF	Portable Document Format
RE	Requirements Engineering
RTF	Rich Text Format
SAE	Solibri Application Engine
SAF	Solibri Application Framework
SIL	Solibri Issue Locator
SMC	Solibri Model Checker
STEP	Standard for The Exchange of Product model data
UI	User Interface
UML	Unified Modeling Language
XML	eXtensible Markup Language
XOR	Exclusive-Or

List of Symbols

$A(P)$	Area of a polygon
$A(p, p, p)$	Area of a triangle
C	Number of building components in the model
E	Number of edges in a polygon
S	Number of segments
N	Number of nodes in the octree
O(z)	Big O Notation
p	Point
P	Polygon
t	Average time
v	Vertex
V	Number of vertices in a polygon
x	X-coordinate
y	Y-coordinate

1 Introduction

1.1 Background

Current development in building design is leading to a situation where buildings are designed with three-dimensional intelligent objects. Since these designs can also be shared using open standards, the possibility has emerged of using the designs to automate tasks that have traditionally been manual. One such task is to ensure that a building has a safe escape route from each room. In this thesis this task is called escape route analysis. The attributes that affect the analysis are given in building fire codes, which are set and enforced by local authorities.

The aim of this thesis is to define the set of information that is needed to automatically perform the escape route analysis, the methods for performing the analysis, and the means for presenting the results of the analysis. From the user point of view the aim is to save time by providing an easy way to check building escape routes against the building fire codes.

This thesis has been researched at Solibri Oy. The results of this work are included as part of the Solibri Model Checker software product. They include help material that is available in the Solibri Model Checker online help.

1.2 Objectives

The main objective of this work was to implement an escape route analysis module, which satisfies most of the current users' and customers' needs and is easily expandable to fulfill future needs.

1.3 Methods

Many software processes comprise the basic activities of the waterfall model: study, definition, design, implementation, and testing. In the software process of this project these activities were practiced in parallel. The methods used in the process are described below.

The study was started at the beginning of the project and continued throughout the work. It is based on literature and interviews.

The definition began with collecting requirements through consulting with stakeholders, studying documents, and brainstorming. The requirements were documented and managed using a spreadsheet program. The priorities for the requirements were calculated simply by multiplying the business value by the estimated implementation effort.

The design is based on these collated requirements. The architecture was designed and documented using the Unified Modeling Language (UML) and design patterns. The high level architecture was designed according to the Solibri Model Checker architecture.

The module was implemented using the Java programming language and the latest development tools. A substantial part of the source code was programmatically generated from the UML diagrams. Performance critical algorithms were optimized using Java code profiler software. The quality of source code was assured using a source code analysis tool.

Testing had an important role in the software process. Unit tests were written for non-trivial code. All unit tests were automatically run and reported nightly. Automatic module testing was used to test the module against the requirements. The implemented module was evaluated against the requirements using two real life building information models obtained from customers. Non-functional requirements (e.g. performance and robustness) were tested using several large building information models.

1.4 Scope

This thesis can be divided roughly into five parts: study, requirement engineering, design, implementation, and testing. The main emphasis is on design and implementation. The study part describes building information modeling (BIM) and escape route analysis. The requirement engineering part presents the requirements and the methods used on a general level. The design and implementation part includes high-level descriptions of the essential algorithms. The testing part introduces different methods used in testing and quality assurance. This thesis excludes data related to requirement prioritization, details of the architecture of the Solibri Model Checker (SMC), source code for the module, and most of the testing results.

1.5 Outline of the Thesis

This section describes the structure and contents of this thesis. The thesis comprises eight chapters and an appendix.

Chapter 1 introduces briefly the project background, objectives, methods, scope, and outline.

Chapter 2 studies building information modeling.

Chapter 3 introduces the Solibri Model Checker product, the platform for the implemented module.

Chapter 4 studies building escape route analysis from evacuation model and building code viewpoints. The emphasis is on the latter. The chapter also introduces definitions and user studies.

Chapter 5 introduces the collected requirements and the methods used in requirements engineering. The requirements are listed in a short form in tables.

Chapter 6 describes the design and the implementation of the escape route analysis module. The high-level architecture of the Solibri Model Checker is also introduced.

Chapter 7 explains how testing and quality assurance were implemented. It also presents some results of the acceptance testing.

Chapter 8 is the conclusion of this work. Future work is also discussed in this chapter.

Appendix contains the UML class diagrams of the implemented modules.

2 Building Information Modeling

2.1 Introduction

The term Building Information Modeling (BIM) was first introduced in 2002 by Autodesk to describe the new technology of the Revit CAD solution. Later on other CAD vendors such as Graphisoft and Bentley started to use the same term to describe their solution as well. Below are two definitions for BIM:

> "Building information modeling (BIM) is a building design and documentation methodology characterized by the creation and use of coordinated, internally consistent computable information about a building project in design and construction." (Autodesk 2005)
>
> "A computable representation of the physical and functional characteristics of a facility and its related project/life-cycle information using open industry standards to inform business decision-making for realizing better value.
>
> BIM can integrate all the relevant aspects into a coherent organization of data that computer applications can access, modify and/or add to, if authorized to do so." (Facilities Information Council 2004)

Building Information Modeling is an approach that offers access to building information through the three major phases of the building life cycle: design, construction, and management. While the adaptation of BIM is challenging, it offers clear advantages for each of these phases and creates a possibility for new services such as the automated escape route analysis implemented in this work. BIM is not a technology, but it can be achieved using a set of interoperable technologies. Different technologies utilize the reliable, high quality, and well-coordinated information stored in the integrated data model. The building model is revised throughout the whole building process resulting a thorough representation of a new building.

The data model of a BIM solution consists of building components and their relationships with each other. The building components have information about their material, purpose (e.g. load bearing and fire protecting), and physical quantities (e.g. length and width). Physical components such as walls, slabs, beams and columns have also 3D

representations. In the best case the model has sufficient information for a range of purposes: 2D documentation, 3D visualization, structural analysis, cost estimation, facilities planning, asset management, and so on. Theoretically, a building information model provides a single, logical, and consistent source for all information associated with the building (Howell & Batcheler 2005).

2.2 Evolution of Design Practices

About 30 years ago nearly all drawings were done with ink or pencil on paper. Major changes meant recreating the drawing from scratch. CAD (Computer-Aided Design/Drafting) automated the task of drafting. Turing award winner Ivan Sutherland produced the innovative program Sketchpad in 1963. It is considered the first step of CAD. CAD applications improved productivity. Drawings were still nothing but dummy 2D graphics: lines, circles, ellipses, hatches, and text. Building elements such as walls were represented as 2D lines. It was possible to separate lines of walls to their own drawing layer but nothing more. The information was in such a form that its meaning could only be interpreted by human.

Charles Lang's team (including Donald Welbourn and A. R. Forrest) began research into 3D CAD software in 1965. The commercial benefits of 3D CAD began to appear in 1970s. The significant development in 3D modeling was the introduction of constructive solid geometry (CSG) (Requicha 1977). The CSG model consists of a set of Boolean operations applied to half-spaces. Another important step in 3D modeling was the introduction of non-uniform rational B-splines (NURBS) invented by Ken Versprille. The first NURBS modeler for PC (personal computer) NöRBS developed by CAS Berlin was available in 1993. The emergence of 3D CAD initially focused almost entirely on creating geometry to support visualization, and subsequent advances concentrated on creating realistic rendering and lighting effects (Howell & Batcheler 2005).

There was also a need for non-graphic information. Object-oriented CAD (OOCAD) systems replaced graphic objects with building elements. The building elements have 3D geometry and a capability to store non-graphic information. Relations between building elements and abstract objects such as spaces made the system more intelligent. A change to an element could now automatically have an effect on other elements through relationships. This was impossible in previous CAD systems. This kind of parametric building modeling technology is separated from OOCAD in Autodesk white paper (Autodesk 2003), but in general terms it is OOCAD with very sophisticated relations between building objects.

The earlier development steps were separated by technology (ink to CAD, 2D to 3D), whereas the newest step – BIM – is an overall concept that emphasizes information sharing between the different software that are used during the building life cycle, from initial design to demolition. The term was introduced by Autodesk in 2002. Figure 2-1 shows the described design practices on a time line. The old practices are still in use and the popularity of the new paradigms is growing.

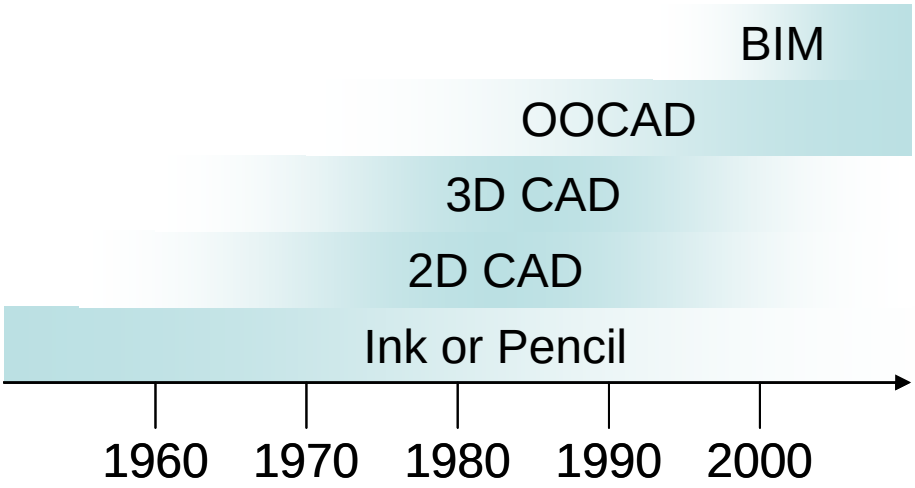

Figure 2-1 Evolution of Design Practices

Currently BIM hotspots are the Nordic countries and Singapore, but BIM is also gaining popularity elsewhere, in Western Europe and the USA. The U.S. General Services administration (GSA) has stated that all AEC firms dealing with GSA will have to include a building information model as part of their work proposal. This year the American institute of architects (AIA) honored three projects with the first annual BIM award.

One of the most recent surveys is conducted by GeoPraxis in 2004 (GeoPraxis 2004). The survey investigated the use of CAD systems in the construction industry. Most of the 687 respondents were architects, designers or CAD drafters from the USA. It was encouraging that 50.9% of the respondents used a BIM application. The top three BIM applications were Graphisoft ArchiCAD (20.8%), Autodesk Architectural Desktop (13.0%), and Autodesk Revit (13.0%). However, 14.4% of respondents did not produce 3D models or building information models at all.

2.3 Use Cases

This section introduces shortly four common use cases of building information modeling. The escape route analysis module implemented in this thesis is useful in two use cases: building design and automated building code checking. For each of the four use cases an example is given.

Building design is the first use of BIM. Intelligent components can automate simple tasks and design work flows easily between different design disciplines, since the information is stored in a non-proprietary format. A good example is the Eureka Tower (located in Melbourne), which is one of the largest projects designed using the principles, methodology, and processes of BIM (Khemlani 2004). In the project, automated documentation allowed everyone to concentrate on the design, which enhances cooperation between employees.

Facilities management (FM) is the management of buildings and services. It deals mostly with building spaces (rooms) and their related equipment. The expenses of managing a building during its lifetime are greater than its design and construction costs. A BIM-based FM solution helps in maintaining information about the building. For example, Frankfurt airport is managed by a Bentley Facilities solution. The BIM solution

allows access to the information on spaces, piping, and cabling. The system helps to keep expensive downtime to minimum.

Quantity take-off is one of the most frequently implemented BIM use cases. It is the process of automatically creating the list of quantities in a building design. By combining quantity information with building component unit costs, it is possible to create a fast and reliable cost estimate for the building. In traditional quantity take-off, a substantial part of the process is manual and laborious. The current trend is to automate the process as much as possible. Graphisoft's Virtual Constructor is a good example of a product where quantity take-off and cost estimation are essential part of modeling software.

Automated building code checking is a promising use case**.** Construction has long been controlled by legislation. Ensuring that a building design follows all building codes is a major undertaking for an architect. Once a BIM model contains all the information required for code checking, it is possible to computerize the checking, which can save a lot of work. Time savings can be particularly substantial if a code violation is found before the design is sent to the building authorities. Building authorities, too, can benefit from automation. Automated building code checking is becoming a reality. The Singapore government has recently unveiled an automated national building code compliance system (NIBS 2005). The Singapore system allows building designers to check their plans against the building codes and submit them to the building authorities for approval.

2.4 Benefits

There is a consensus on the importance of BIM and its potential benefits. There is not yet much scientific research on BIM, but a number of successful BIM projects in the building industry show that BIM has obvious advantages when compared to traditional methods. Most of the recent writings of the AEC field experts and whitepapers of CAD vendors give a positive picture of BIM. The following list of the benefits is collected and refined from BIM project success stories, the writings of experts, and CAD vendor whitepapers: higher productivity, higher quality, better coordination, central information, consistency, accessibility, versatility, and new services.

2.5 Challenges

Adoption of BIM is very challenging because building industry firms are relatively small and contracts are often made for single projects. The agreements between different parties state clearly the responsibilities of each party. Where BIM propagates for sharing of information, the agreements inhibit sharing of responsibility. Quite often price is the most important selection criteria when choosing subcontractors. This often leads to sub-optimally performing project teams.

A panel of CAD vendors and end users at the Technology for Construction Executive Forum held in 2004 brought out the following obstacles to BIM implementation: globalization, corporate culture, data storage costs, lack of standards, and interoperability (Ferris 2005). David Becker holds that the fragmentation of the building industry and current building processes are also challenges for BIM (Becker 2004).

Globalization is challenge, because many firms have offshore business, often in regions that do not have the technology to support BIM.

Corporate culture is one of the biggest challenges. Project participants such as estimators and draftsmen are used to doing their jobs in a particular way. They may not want to change their way of working or trust the data derived from a building information model. The employees may also fear losing their positions as experts when their skills become obsolete.

Data storage costs increases as the amount of information grows. The increased collaboration and regulatory requirements to save data also serve to grow the costs. The most used IFC exchange format is uneconomic. The size of a small building model without details is normally several megabytes. Currently the IFC standard is extending, which means that models become even bigger.

Lack of standards is one of the reasons for the variable quality of building models. There are no agreements as to what information must be included in the models. Different parties are creating guidelines to solve the problem, but they are not yet widely adopted. In Finland the ProIt project recently published guidelines for creating better building models (ProIt 2004, ProIt 2005).

Interoperability is a key issue because each building project requires the use of many software applications. One solution is a standardized format for building models. Currently the IFC exchange format studied in the next section (2.6) is the most promising. More applications should support IFC to make interoperability a reality.

Fragmentation of building industry has led to technological inconsistency across different segments. Leading-edge architects may produce 3D models, but they have to deliver 2D paper drawings to city planning departments. A lot of information is lost in conversions.

Current building processes are full of old rules that define the responsibilities of employees and organizations. Current organizational structures have a lot of inefficiency. BIM is requiring architects and designers to perform tasks that were typically left to other stakeholders, but changing the rules is hard. Quantity surveyors are afraid of losing their work when quantities are automatically listed by software.

In this project the biggest challenge was the varying quality of building information models. It may be a consequence of poor modeling standards or knowledge. In most of the tested real life building models there was decent modeling of the geometry of building elements but the relations between elements were occasionally missing. Some of the building models had also building elements that were of the wrong type e.g. doors were modeled as windows. Because of these kinds of modeling errors the escape route analysis module has to use geometry as a primary information source. This raises the complexity of the analysis significantly.

2.6 IFC

Each major CAD solution has its own BIM formats and a set of compatible building-related applications. Smaller CAD vendors lack resources and they tend to integrate third party applications. So there is a need for standard BIM format that could support interoperability across individual, discipline-specific applications. (Khemlani 2003)

Industry Foundation Classes (IFC) is an open international standard managed by the International Alliance of Interoperability (IAI). IAI is an alliance of organizations within the construction and facilities management industries dedicated to improving processes within the industry through defining the use and sharing of information. Organizations within the alliance include architects, engineers, contractors, building owners, facility managers, manufacturers, software vendors, information providers, government agencies, research laboratories, universities and more. (IAI International 2004)

The IFC model is expressed in EXPRESS language, which is the data modeling language of STEP (Standard for the Exchange of Product model data, ISO 10303) and standardized as ISO 10303-11. EXPRESS consists of language elements which allow unambiguous data definition and specification of constraints on the data defined and by which aspects of product data can be specified. It deals with data types and constraints on instances of the data types. (ISO 10303-11 1994)

The IFC high level architecture diagram is illustrated in Figure 2-2. The IFC model is divided into four layers: domain, interoperability, core, and resource layer. Each layer contains categories which define sets of entities. There are 623 entity definitions in the IFC2x2 model. Entities on layers can only be related to or reference an entity at the same or lower layer, but not one at a higher layer. It means that a boiler in the HVAC category on domain layer can be related to a wall in building element category on interoperability layer, but the wall cannot be related to the boiler.

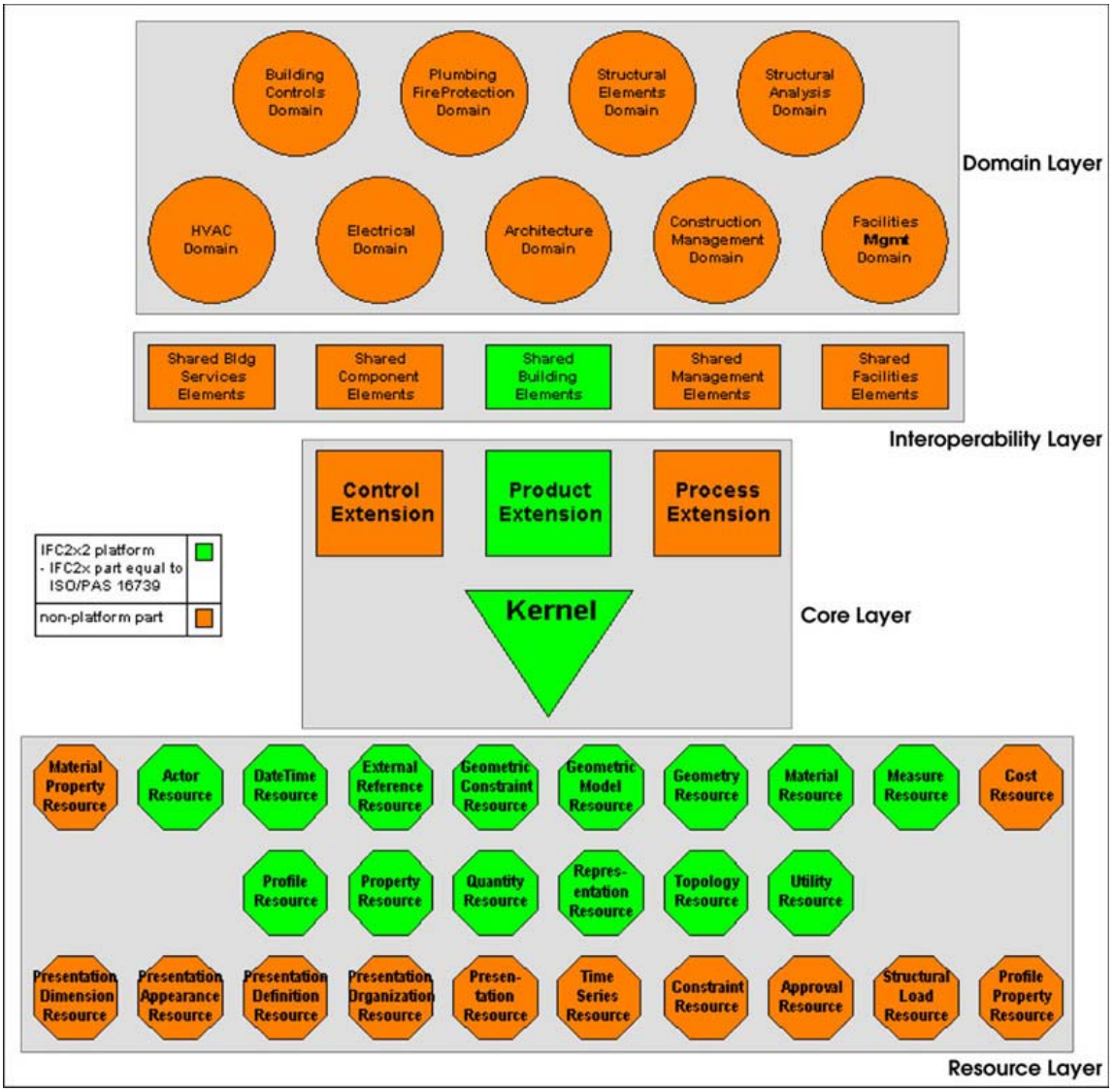

Figure 2-2 The overall architecture of the IFC model (IAI International 2003)

The analysis of this thesis uses entities on the interoperability layer and on lower layers. Essential entities for the escape route analysis of this thesis are space, wall, door, opening, and stair. Spaces are the most central entities for the analysis. The location and geometric representation of a space define the location for the routes. The space identifiers: type, name, and number are used in defining occupancies and reporting results. The location and geometry of walls are mainly used in compartmentation and combining partial spaces. The doors and openings are used in connecting routes between spaces through walls. The stairs are used in vertical connection of spaces. Use of relations between entities is avoided in the analysis because they are often lacking or faulty. Entities from the plumbing fire protection domain are not used, because currently there is only one fire protection entity specified. There are some projects in IAI that are related to escape routes. A project named Escape Route Planning (AR-4 1998) was started in 1998 and is currently on hold and looking at resources to start again. Another project named Code Compliance Support (CS-4) was completed in 2003, and fire and personal safety was one of the areas that had a particular focus. The capabilities of some existing entities have been extended and support for alarms, controls, drainage etc. has been added to the domain layer.

3 Solibri Model Checker

3.1 General

Solibri Model Checker (SMC) is a commercial design spell-checking software product for the Architecture Engineering Construction (AEC) and Facilities Management (FM) industry. SMC can check building information models in the IFC format or models imported directly from ArchiCAD. SMC is a stand-alone application and is compatible with Microsoft Windows and Apple Mac OS X. SMC is localized into English and Finnish languages and metric and imperial units.

Solibri Model Checker adds value throughout the life cycle of the building. It is a valuable tool for architects, construction companies, and building owners. The users can check design cost-effectively, deliver high quality building information models and obtain reliable cost estimates and key factors.

In this thesis Solibri Model Checker provides the software environment in which the escape route module operates. SMC provides an internal object-oriented representation of the building information model that is used as source information for the escape analysis.

3.2 User Interface

The user interface (UI) of SMC is shown in Figure 3-1. The functionality of the SMC is arranged in different views. The UI of each view consists of a toolbar and a panel. The user can resize, change location, close, and open them. The user interface includes two main perspectives: SMC, which is the default perspective, and CSM (Constraint Set Manager). Perspectives are visual containers for a set of views. The UI of CSM is not introduced in this thesis because it is not relevant to this work. The perspectives and view are controlled from the Window menu.

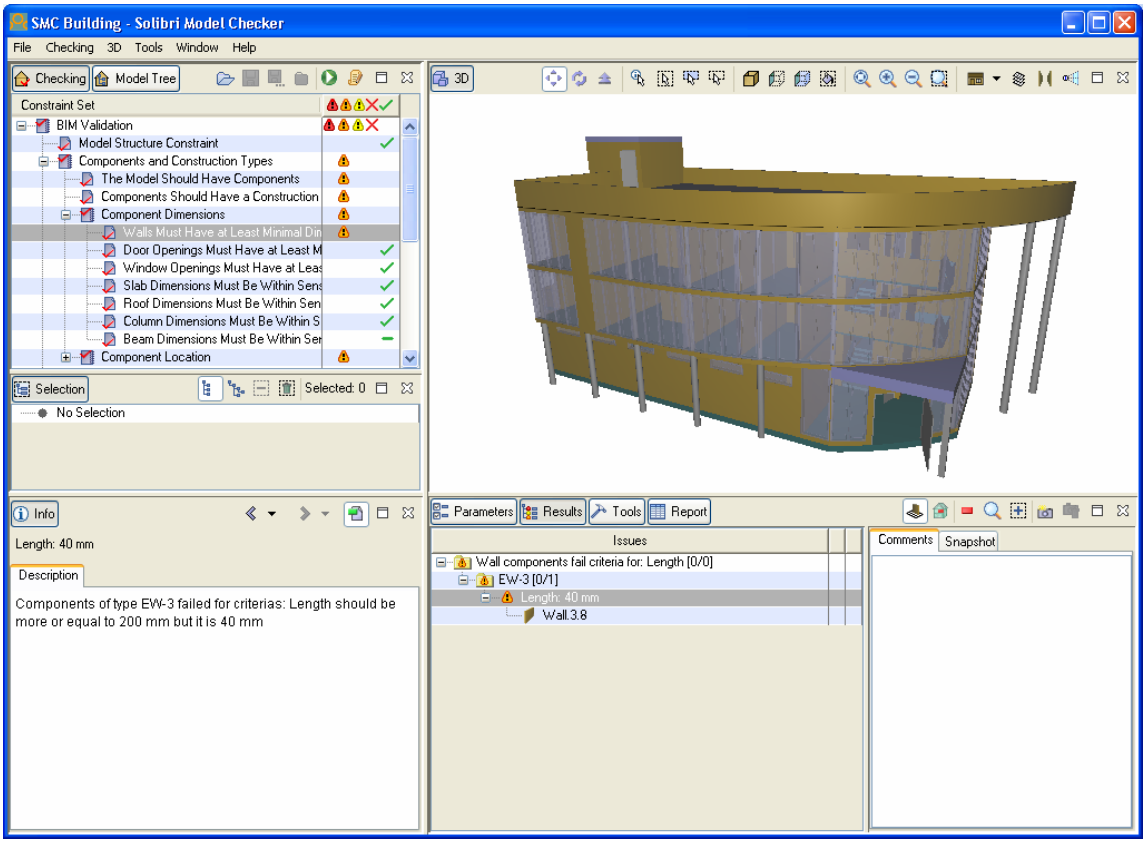

Figure 3-1 The user interface of Solibri Model Checker 3.0

The building information model is visualized in the 3D view, where the model can be easily navigated and visualized in many different ways. Checking of the model is based on rules defined by constraints which are shown in the checking view in top left corner. The constraints are introduced in more detail in the next section 3.3. The lower right corner has four views: parameters, results, tools, and report. Each of these views contains specific information about the constraint selected in the checking view. Information on the currently selected building element, constraint, constraint result, and other objects is shown in the info view in the bottom left corner. The default UI layout includes the model tree view for browsing the model and the selection view for handling selections. The filter view and compartmentation view are not visible by default. The compartmentation module is a part of this work and is discussed in section 6.3.5.

3.3 Constraints

SMC checks and analyses the product model using constraints. Constraints are compact software modules that contain checking parameters and logic. Constraints are configured based on the user requirements. For example, the maximum allowed escape route length is adjusted based on local requirements. Several constraints are combined into constraint sets. Constraints sets can also nest other constraint sets. Constraint sets are stored in files that can be created and modified in the CSM perspective. In the SMC perspective the constraint sets are located in the checking view (Figure 3-2).

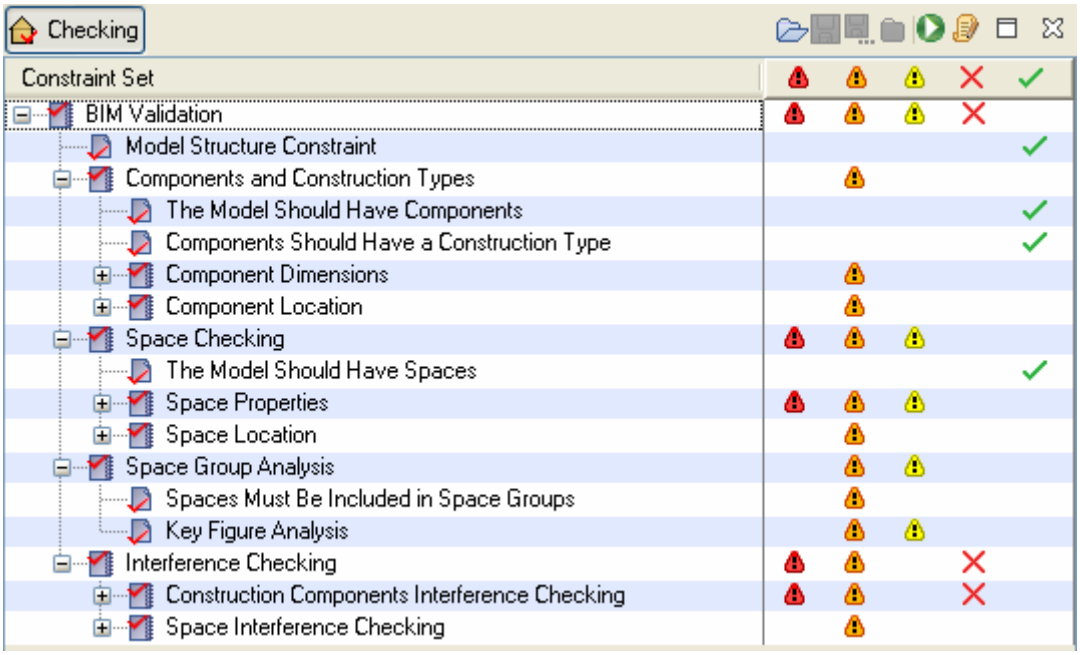

Figure 3-2 BIM Validation constraint set in checking view

The result icons in the right side indicate that the constraint set is checked. The colored triangles with exclamation marks reveal how severe the problems found under the branch are. The red 'x' means that some of the problems are marked rejected. The green check mark means that no problems have been found or that all the underlying problems have been accepted by the user.

3.4 Workflow

This section gives a high level picture of the normal workflow with Solibri Model Checker. The advanced features of SMC are not described. The main points of the process are shown in Figure 3-3. The process can be repeated several times during the building information model life-cycle.

SMC can check building information models in the IFC format or models imported directly from ArchiCAD. Currently almost every CAD vendor supports the IFC format. Some CAD applications include IFC support as part of the product while others provide a separate IFC module. Importing the model directly from ArchiCAD is done by a module which is integrated to the ArchiCAD during the installation of SMC. When the building is fully loaded into SMC it appears in the 3D view.

After a model is opened a suitable constraint set file is opened into the checking view. The SMC installation contains constraint sets designed for various purposes. The escape route constraint, whose development is described in this thesis, is included in the Security Check constraint set. Usually constraints are preconfigured to meet the requirements of the organization and the checking can started right away. Configuration can be done in the constraint parameter view or in the CSM perspective. Checking is a fully automatic process and it normally lasts from a couple of seconds to several minutes, depending on the size of model and the number and complexity of constraints.

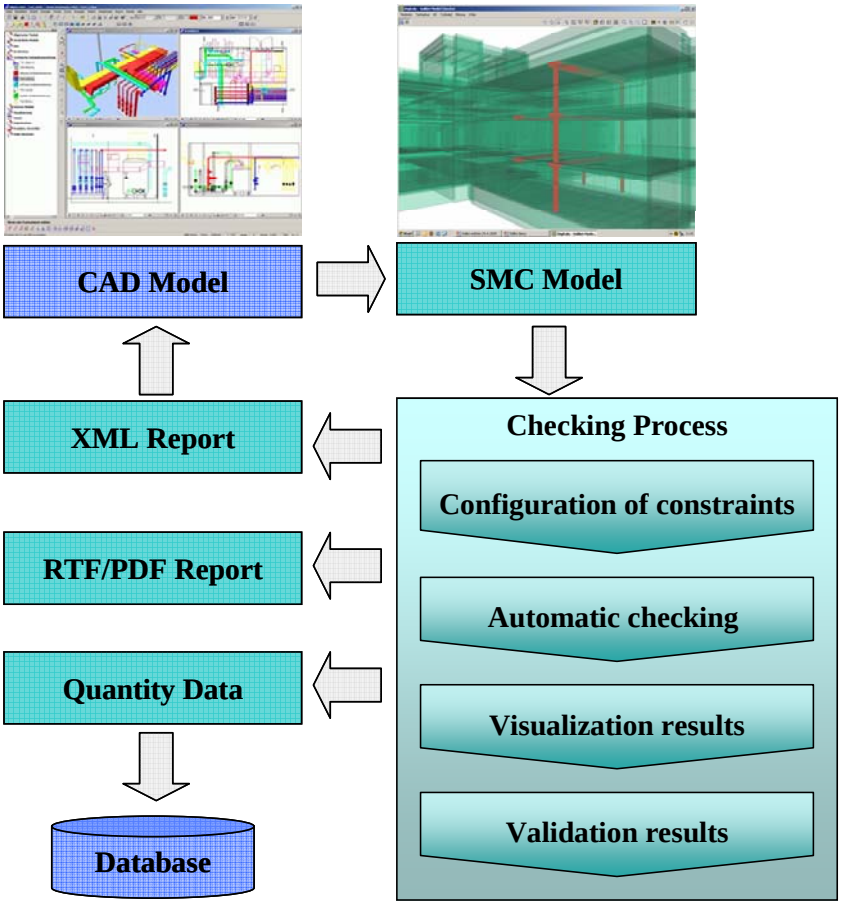

Figure 3-3 Solibri Model Checker workflow

The results for each checked constraint are immediately available for closer examination even if the whole checking process is not yet finished. The results are represented in the constraint result view and the generated report tables are located in the constraint report view. The checking results consist of categorized issues. Every issue has a textual description of the problem and a list of building elements related to the problem. In the result view the user can mark which issues are real problems and which are not. The user can also attach textual comments and snapshots from the 3D view to the issues. To help decision making, SMC visualizes the selected problems automatically in the 3D view. The report tables contain usually quantity and key figure data. The escape route constraint does not generate any report data.

When the user has gone through all the issues, it is time to generate the report document. The supported report formats are RTF (Rich Text Format), PDF (Portable Document Format), and XML (eXtensible Markup Language) (W3C 2003). The report contains all issues with decisions, comments and snapshots added by the user. The RTF format is the best format for later word processing. The XML report can be imported to the ArchiCAD and Autodesk ADT using Solibri Issue Locator (SIL). The SIL helps the fixing of problems by locating the problematic building elements in the CAD application.

4 Escape Route Analysis

4.1 Introduction

Traditional building plan checking and approval process using the manual approach is inefficient and time-consuming. It needs a huge amount of educated manpower and the time spent delays the whole project. Automated checking of building codes offers significant benefits. Possible design flaws can be found in minutes instead of hours. Well designed automated code checking does not miss flaws as humans sometimes do. Automated code checking has been studied for about ten years. Existing work includes (Han et al. 1997), (Han et al. 2002), (Rong et al. 2004).

This chapter studies escape route analysis from two viewpoints: evacuation models and fire codes. The aim of this study is to find the essential escape route regulations and properties compliance checking for which is possible using the current building information models. The main emphasis is on fire codes, because the analysis of this thesis is based on them. Evacuation models are introduced shortly to bring out another viewpoint. The last section introduces results of interviews of architects and public authorities. The current processes and problems of escape route analysis are also discussed in the section.

4.2 Definitions

Table 4-1 introduces central terms with definitions that are related to escape route analysis and used in this thesis. The definitions are mainly collected from the building code documents.

Table 4-1 Escape route definitions used in the thesis

Term	Definition
Escape Route	The entire path of travel, measured from an escape door to the furthest point in any room in a building.
Fire Compartment	An enclosed space in a building that is separated from all other parts of the building by enclosing construction that provides a fire separation having a required fire-resistance rating. (Ontario Fire Code 1997)
Fire Compartmentation	The process of defining fire compartments.
Exit	A part of a means of egress, including doorways, that leads from the floor area it serves to a separate building, an open public thoroughfare or an exterior open space protected from fire exposure from the building and having access to an open public thoroughfare. (Ontario Fire Code 1997)
Exit Passageway	An enclosed passageway that leads from the compartment exit to the final exit.
Firewall	A fire separation of noncombustible construction that subdivides a building or separates adjoining buildings to resist the spread of fire that has a fire-resistance rating as prescribed in the building code and that has structural stability to remain intact under fire conditions for the required fire-rated time. (Ontario Fire Code 1997)
Fire Use	A member of a fire use classification. (AR-4 1998)
Fire Use Classification	A classification listing off all the possible uses of a building or space for the purposes of fire compartmentation. (AR-4 1998)
Occupancy Load	The number of persons for which a building or part thereof is designed. (Ontario Fire Code 1997)
Storey	A portion of a building that is situated between the top of any floor and the top of the floor next above it, and where there is no floor above it, that portion between the top of the floor and the ceiling above it. (Ontario Fire Code 1997)
Travel Distance	The distance from any point in a floor area to an exit measured along the path of exit travel, except that when floor areas are subdivided into rooms used singly or into suites of rooms and served by public corridors or exterior passageways, the distance shall be measured from the door of the rooms or suites to the nearest exit. (Ontario Fire Code 1997)

4.3 Evacuation Models

Models of human behavior in fire evacuation have been researched since the late 1970s. Currently evacuation models are increasingly used in assessing the fire safety of buildings. The potential applications are assisting building design, development of performance-based building codes, emergency planning, and crowd planning and management. If the models are used early enough in the design phase, models can help in identifying possible design problems.

The two main categories of models are conceptual models and computer models. The conceptual models are more abstract and theoretical than computer models. The conceptual models try to explain the decision making process, stress, and behavioral responses of occupants in an emergency. The computer models simulate human movement and behavior during fire emergencies. The main objective of the computer models is to predict evacuation times. Visualizations are very important for the computer models. Figure 4-1 shows two different computer model visualizations from the same location of a building.

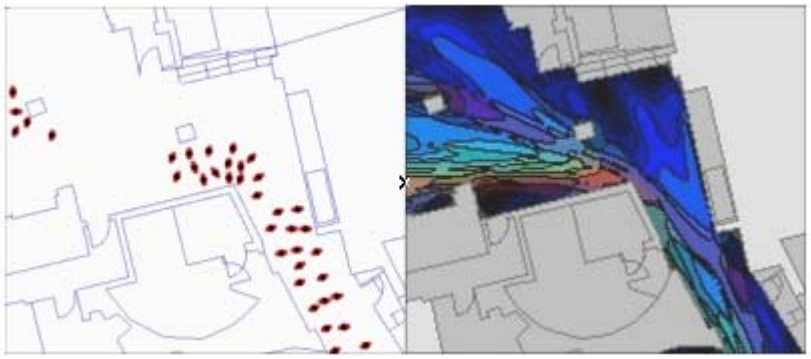

Figure 4-1 Visualization of Simulex model and Myriad analysis (Crowd Dynamics 2005)

Currently the models are still hard to use and the results between different models vary a lot. The review of 28 egress models shows that only 9 of them allow the user to import buildings from a CAD application (Kuligowski 2004). Often the drawings imported from the CAD need to be modified and extra information added before the model can be used. In the comparison of two egress models the evacuation times differed by as much as 40% (Kuligowski & Milke 2004). A report by the Society of Fire Protection Engineers notes that the evacuation model tends to rely heavily on assumptions and it is not possible to gauge with confidence their predictive accuracy (SFPE 2002).

4.4 Building Fire Codes

4.4.1 Overview

Building codes are used everywhere to control quality of building and engineering provision. The codes differ from one place to another, but the building information needed for code compliance is consistent. Codes are generally considered the minimum acceptable level of safety for a new building. Usually building codes include the following

parts: structural safety, fire safety, health requirements, and accessibility. The fire safety codes are essential for this thesis.

Fire codes are that portion of the building code that relates to fire safety requirements, and standards. The earliest public fire regulations in the US were adopted by New York City in 1860 (NFPA 1983). One of the first model regulations promoted by the National Fire Protection Association (NFPA) was the 1927 Building Exits Code (Bukowski & Kuligowski 2004). The first Finnish fire regulations were established by the law L 26/1920 (Finland's environmental administration 2003). The current Finnish fire code E1 was introduced in 1976 and has since been updated in 1981, 1997, and 2002.

4.4.2 Relevant Properties

In the thesis several fire codes used in countries in North America, Europe and Asia were used as reference material. The purpose was to identify the most essential escape route regulations and find similarities between different codes. The method used in the study of codes was simply reading and comparing. The structure and similarity of government regulations has been researched using more scientific methods in the Regnet & Regbase project (Lau et al. 2003).

The basic principles in the codes were similar, but their presentation differed slightly. Figure 4-2 and Figure 4-3 are examples of two regulation tables. These tables are relatively complex compared to the usual presentation of regulations, although the accompanying long lists of exceptions are not shown.

Occupancy Group of Building or Space	Group Desig-nation	Maximum Travel Distance (ft.) [a]		Capacity Number of Persons per Unit of Width				Corridors[i]	
				Doors Openings[m]		Stairs, Escalators[k]	Ramps,[b] Corridors, Exit Passage-ways,[j] Horizontal Exits	Min. Width (in.)	Max. Dead End[h] (length in ft.)
		Unsprinklered	Sprinklered	To Outdoors at Grade	All Other Exit and Corridor Doors				
High Hazard	A	N.P.	150	50	40	30	50	36	N.P.
Storage	B-1[c]	100	150	75	60	45	75	36	50
	B-2[c]	125	175	75	60	45	75	36	N.R.
Mercantile	C	150	200	100	80	60	100	36	50
Industrial	D-1	125	175	100	80	60	100	44	50
	D-2	150	200	100	80	60	100	44	50
Business	E	200	300	100	80	60	100	44	50
Assembly*	F	150	200	100	80	60	100	44	30
Educational	G	150	200	100	80	60	100	66[e]	30[d]
Institutional	H-1	125	175	50	40	30	50	36	40
	H-2	125	175	30	30	15	30	96[f]	30[g]
Residential	J-1	150	200	50	40	30	50	36	40
	J-2	150	200	50	40	30	50	36	40
	J-3	N.R.	N.R.	N.R.	N.R.	N.R.	N.R.	N.R.	N.R.

Figure 4-2 Determination of exit and access requirements (Building Code of the City of New York 2004)

Capacity of room or storey	Min. No. of exit doors (from room) or exit routes (from storey)	Min. Total Width of		Min. Width of each	
		exit doors	exit routes	exit door	exit route
4 - 30	1			750 mm	1050 mm
31 - 200	2	1750 mm	2100 mm	850 mm	1050 mm
201 - 300	2	2500 mm	2500 mm	1050 mm	1050 mm
301 - 500	2	3000 mm	3000 mm	1050 mm	1050 mm
501 - 750	3	4500 mm	4500 mm	1200 mm	1200 mm
751 - 1000	4	6000 mm	6000 mm	1200 mm	1200 mm
1001 - 1250	5	7500 mm	7500 mm	1350 mm	1350 mm
1251 - 1500	6	9000 mm	9000 mm	1350 mm	1350 mm
over 1500	7 or such greater number as the Building Authority may require	to be calculated at the rate of 300mm per 50 persons		1500 mm	1500 mm

Figure 4-3 Minimum number of exit doors from a room, or exit routes from a storey, and required minimum width thereof (Hong Kong Buildings Department 1996)

The basic idea of the fire codes is to ensure that it is possible to exit safely from the building in case of fire or other emergency. The building must have a sufficient number of suitably located exit passageways that have sufficient capacity, so that exit time is not dangerously long. In the study of different fire codes five essential properties were found:

Number of occupants defines the number of persons that is used in definition of minimums for capacity and number of escape routes. It can be specified using an actual number for whom the spaces are designed or using appropriate occupant-area ratios. The actual number is used for spaces that have, for example, fixed seating. The occupant-area ratio can depend on the usage of the space. Sometimes occupant load is used instead of the term number of occupants and fire use instead of the term usage.

Minimum number of escape routes defines how many separate routes must be leading from a space to a safe place. Usually the minimum is two routes, but in residential buildings only one route is often allowed. The minimum number of escape routes can depend on the usage and number of occupants of the space (Figure 4-3).

Maximum travel distance defines the maximum allowed escape route length inside a fire compartment measured from a space to a safe place. The route starts from the furthest corner space or the center of a door and ends to the center of an exit door. The start point can depend on the usage of the space. The exit door leads out from a fire compartment or directly out from the building. The measurement method is often loosely defined e.g. “Travel distance shall be measured along a natural and unobstructed path of travel”

(Building Code of the City of New York 2004). The maximum travel distance can depend on the number of different escape routes (Figure 4-2).

Minimum width of escape route defines the minimum capacity for spaces and door. The widths can depend on the occupant load and the number of escape routes (Figure 4-3). The occupant load is a cumulative sum of occupants that uses the door or space in case of escape.

Minimum height of escape route defines the minimum clear height of spaces and doors. Usually it is a simple value without any exceptions, as in the fire code of Finland: “Exit passageways shall have a clear height of 2100mm” (Finland's environmental administration 2002).

The codes also comprise a large number of other properties that are important, but very difficult to check because of insufficient information on current building information models. Checking of them needs information about objects that are not yet included in the IFC exchange format. The five selected properties deal with basic building elements and their compliance can analyzed with most building models. As the IFC evolves the analysis module can be developed to check new properties.

4.5 User Studies

Three user studies have been done in the early stages of the work. The objective of the interviews was to gain a greater understanding of current processes relating to escape route planning in Finland. All the people interviewed were chosen from different sectors: the Ministry of the Environment, an architect’s office, and the Helsinki building inspectorate department. The Ministry of the Environment is responsible for development of building codes. Architects plan and design buildings according to the codes. The building inspectorate office approves and stores building plans.

Before the questioning, this project and the Solibri Model Checker product were introduced to the interviewees. The questions were planned before each meeting. The main emphasis was in stakeholders, responsibilities, current practices, and building fire codes.

The Ministry of the Environment is responsible for the national building code of Finland. The focus in the discussion was on the fire codes (Finland's environmental administration 2002). It seemed that the regulations for escape route length and capacity were the most important parts of the code. The route length depends much on the method of measurement. Environment guide number 39 (Finland's environmental administration 2003) is a useful handbook that helps users understand the fire codes. (Lilja 2004)

The visit to an architect’s office gave a good picture of current escape route planning process. A chief architect is responsible for the plan. Normally, a project architect checks the escape routes manually without any special software. In some complex cases a fire safety consultant may be used. The time spent in checking of routes varies from minutes to days depending on the size and complexity of building. Sometimes more time is spent because of alterations to the design of the building and escape routes have to be checked again. The escape routes are documented in the master drawings. The documentation

contains as a minimum the borders of fire compartments, the widths of passageways and doors, and the number of occupants. (Isoaho 2004)

The building inspectorate department approves the plans and grants a building license. Usually the process takes between two weeks and a few months. The master drawings are one of many documents that have to be delivered. Escape routes are marked on these drawings. The drawings are still delivered and filed on paper. Sometimes CAD software is used in order to help the approval process. There have been discussions about using building information models in the process, but it seems that the filing of the digital models is the most problematic issue. It is difficult to ensure that the models will be usable during the whole life-cycle of a building, which can be over a hundred years. (Miller et al. 2004)

The architects were the most promising users for the escape route analysis module implemented in this work. Many architects are skilled CAD users and they are more and more moving towards building information modeling. The building inspectors were the second most promising user group that could take advantage of the module, if building information models become part of the deliverables. It is hard to see how the module might be valuable for the Ministry of the Environment.

5 Requirements Engineering

5.1 Introduction

This chapter introduces the requirements engineering (RE) practices used and the requirements for the module. RE covers all of the activities involved in discovering, documenting, and maintaining a set of requirements for a system. The term engineering implies that systematic and repeatable techniques should be used to ensure that system requirements are complete, consistent, relevant etc (Sommerville & Sawyer 1997). The RE can be divided to three activities: requirements definition, requirements management, and acceptance testing. The activities and their relations are shown in Figure 5-1. This chapter focuses on the requirements definition and management. The acceptance testing, which validates the system against the requirements, is described in section 7.3.

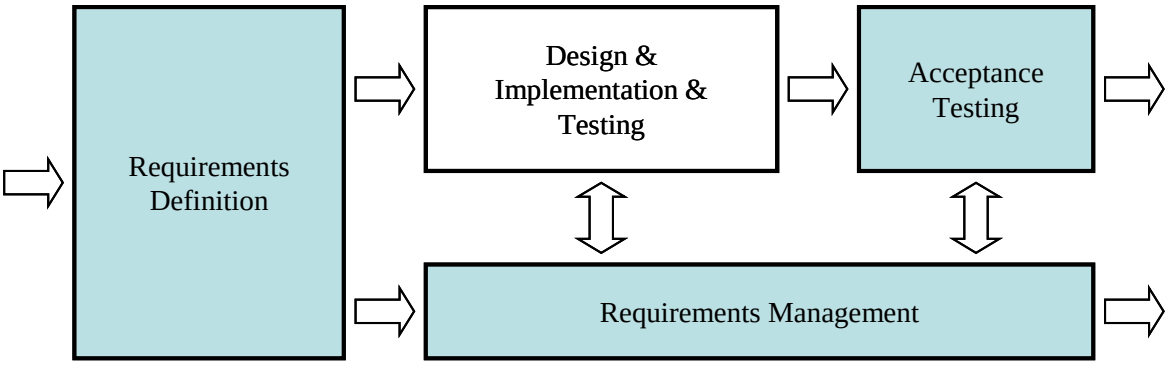

Figure 5-1 Requirements engineering process

In this thesis the requirements are divided into functional and non-functional requirements. The functional requirements specify functions or services that the system must be capable of performing from a user's point of view. The non-functional requirements describe the properties of the system, including usability, reliability and performance. All the requirements are represented later in this chapter. The requirements provided the basis for the design, implementation, and testing. They were also used in effort estimation and task planning.

5.2 Requirements Definition

The purpose of the requirements definition process is to refine business requirements into a requirements document. The process consists of four phases: elicitation, analysis, representation, and validation.

In the elicitation phase the requirements were discovered by interviewing architects and public authorities, studying building fire codes, and brainstorming. The interviews and fire code study are represented in Chapter 4. Some requirements were also reused from the existing constraint modules.

In the analysis phase the collected requirements were refined into the initial set of requirements. Each requirement was given estimates of business value and implementation effort. These estimates were used in prioritization of requirements. The priority for a requirement was calculated simply by multiplying the business value by implementation effort.

The requirements were documented using a spreadsheet application, which helped in the priority calculation. The uses cases were not used because the effort of writing was too great for a one-man project. The requirement tables are represented in this chapter in short form without information relating to prioritization.

5.3 Requirements Management

Requirements management is the process of managing changes to a system's requirements (Kotonya & Sommerville 1998). Normally requirements management needs a remarkable effort. Because of the size of this project the effort spent on requirements management was quite small.

In this project the requirements were managed by updating the requirements document. The document was updated when new requirements were discovered or the estimates of existing requirements were clarified. Improved understanding was the main reason for the changes. The document acted also as a priority list for the design and implementation.

5.4 Functional Requirements

The user requirements are listed in Table 5-1. Each of these requirements describes one function that the system must provide to users. The requirements do not describe user interface or non-functional properties.

Table 5-1 User Requirements

ID	Short Description
R1	The user must be able to add exits (doors, windows, and openings).
R2	The user must be able to remove exits (doors, windows, and openings).
R3	The user must be able to visualize the current exits (doors, windows, and openings).
R4	The user must be able to add fire compartments.
R5	The user must be able to remove fire compartments.
R6	The user must be able to visualize the fire compartments.
R7	The user must be able to specify the minimal acceptable length of escape routes.
R8	The user must be able to specify the minimal acceptable height of escape routes.
R9	The user must be able to specify the measurement method of the escape route (direct linear measurement, indirect linear measurement, and wall aligned linear measurement).
R10	The user must be able to specify the spaces where the escape route starts from the door or from the furthest corner of the space.
R11	The user must be able to specify fixed occupancy numbers for the spaces.
R12	The user must be able to specify occupancy number per area unit for the spaces.
R13	The user must be able to specify a person number when the doors must open in the direction of escape.
R14	The user must be able to specify the minimal acceptable number of different escape routes the spaces must have with different occupancy numbers.
R15	The user must be able to specify the minimal acceptable escape route width with different person numbers.
R16	The user must be able to visualize the escape routes for the selected space.
R17	The user must be able to visualize the calculations of the analysis.
R18	The user must be able to produce a report of the calculations.
R19	The user must be able to get a list of reasons why the model cannot be analyzed.
R20	The user must be able to get a list of spaces that do not have an escape route.
R21	The user must be able to get a list of spaces that do not have an acceptable escape route.
R22	The user must be able to get a list of spaces that do not have an acceptable number of different escape routes.

The system requirements are listed in Table 5-2. These requirements are similar to the user requirements, except that the system plays role of the user.

Table 5-2 System Requirements

ID	Short Description
R23	The system must automatically detect fire compartments.
R24	The system must automatically detect exterior exits (door, windows, and openings).

5.5 Non-functional Requirements

The non-functional requirements are listed in Table 5-3. Each of these requirements describes one property that the system must provide. The main emphasis in the definition of these requirements was verifiability.

Table 5-3 Non-Functional Requirements

ID	Short Description
R25	The user interface must have English and Finnish localizations.
R26	The parameters must have different default values for the imperial and the metric localizations.
R27	The system with recommended hardware must be able to analyze a building with 500 spaces in 5 minutes.
R28	The system must be able to analyze all building models available in Solibri without program errors.

6 Design and Implementation

6.1 Introduction

This chapter describes the design and the implementation of the escape route analysis module. Design was done simultaneously with implementation. The objective was to implement an escape route analysis module, but in addition four general modules were implemented. In terms of SMC the escape route analysis module is a constraint which checks escape routes. The functionalities of route analysis and compartment creation were separated from the constraint module to their own modules, because there are other constraints planned that need the same functionalities. Two additional modules were implemented to ensure performance and low memory consumption.

The design of the modules relied heavily on use of UML diagrams. Use case and class diagrams were modeled using the MagicDraw application (No Magic 2005). It supports the addition of extra information relating to object oriented programming languages and source code documentation. This feature – with the possibility of generating source code from class diagrams – helped implementation a lot. The class diagrams are represented in the appendix of this document.

The whole implementation was carried out using the Java programming language and Eclipse integrated development environment (IDE) (Eclipse 2005). Testing and quality assurance performed during the implementation are introduced in Chapter 7. A substantial effort in addition to coding was spent in designing algorithms to solve geometric problems. The varying quality of building information models was the biggest challenge for the algorithms. The most notable are an algorithm for resizing areas, an algorithm for merging interconnected spaces, and an algorithm for creating compartments. All these algorithms are described in the later sections of this chapter.

6.2 High Level Architecture

The current architecture of the Solibri Model Checker is divided into three clearly separated layers. The lowest layer is Solibri Application Engine (SAE). The next layer is Solibri Application Framework (SAF) and the topmost layer is the Application Layer. (Solibri 2001)

The high level architecture of the escape route analysis module follows the architecture of the Solibri Model Checker. It consists of one constraint module and a few plug-in modules. The constraint module serves as the user interface and performs the actual checking of the building information model. The plug-in modules offer services in selected functional areas and any constraint module can use these services. The plug-in modules are placed in the SAF layer and the constraint module is placed in the application layer. Figure 6-1 shows the modules and their relations in a layer diagram.

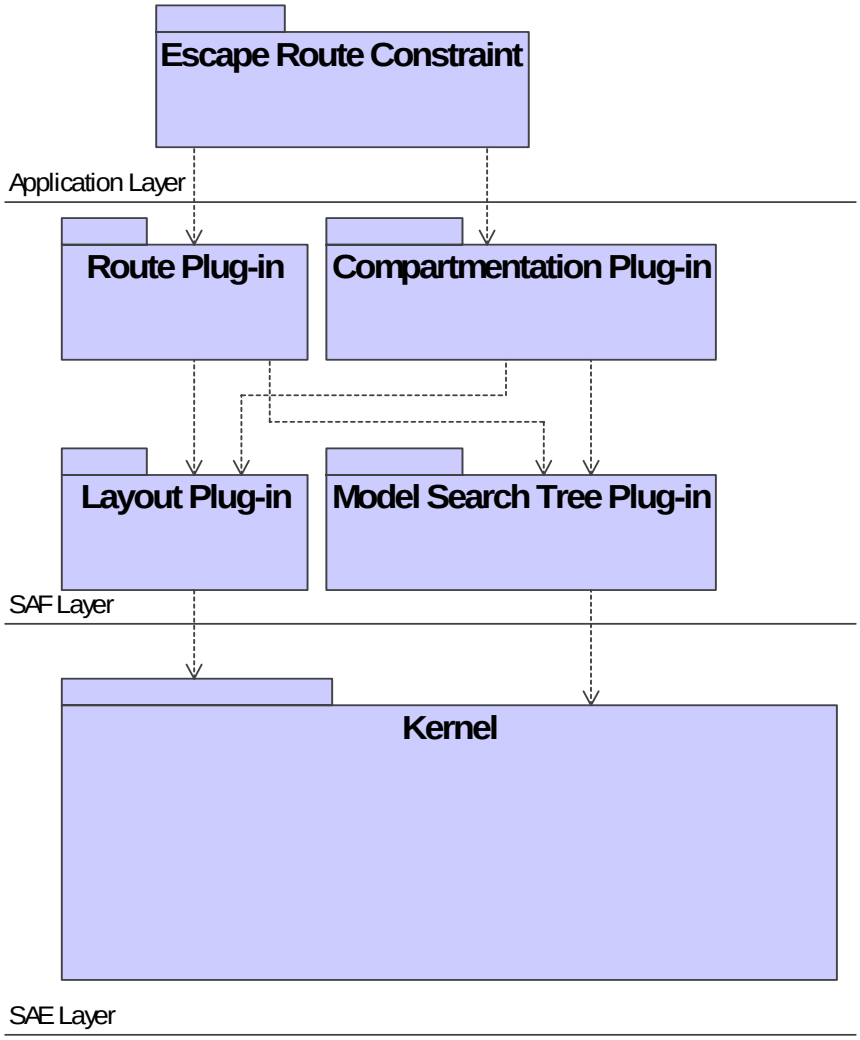

Figure 6-1 High Level Architecture

6.3 Plug-in Modules

6.3.1 Overview

In the architecture of SMC plug-ins there are program modules that provide functionalities and optionally UI for the application. The implemented plug-ins are located in the SAF layer and they implement the singleton design pattern, which ensures that the class has only one instance and it is globally accessible (Gamma et al. 1995). This ensures reusability of the modules in the coming projects. The class diagrams of the plug-ins are represented in the appendix of this document.

6.3.2 Model Search Tree Plug-in

The model search tree plug-in offers fast geometric building component searches. Geometric searches are needed because the building components are often missing relations to other components. It is often the case that a space object does not have information about its bounding walls. The most common use case is to find all components that are near a component, e.g. walls that are near a space. The model search tree is used in route and compartmentation plug-ins.

There are two commonly used tree structures that can be used for three-dimensional searches – octree and a binary space-partitioning (BSP) tree. The octree structure was chosen for this plug-in. The BSP trees have more efficient partitioning of space and search times are shorter (Hearn & Baker 1994). The octree was chosen instead of the BSP tree because its implementation is easier and construction of the octree is faster, and search times are short enough. In the current implementation the tree construction of a huge model takes less than 500ms, which is acceptable.

The octree structure is based on a node with eight children. Each node corresponds to a cubic region of three-dimensional space (Figure 6-2). Building components are linked to the lowest node in the tree structure that entirely contains the three-dimensional shape of the component. Large components are normally in the upper nodes and smaller components in the lower nodes. A search of the building components starts from the root node and continues recursively to the child nodes. The search continues only to these nodes whose region intersects with the region under search.

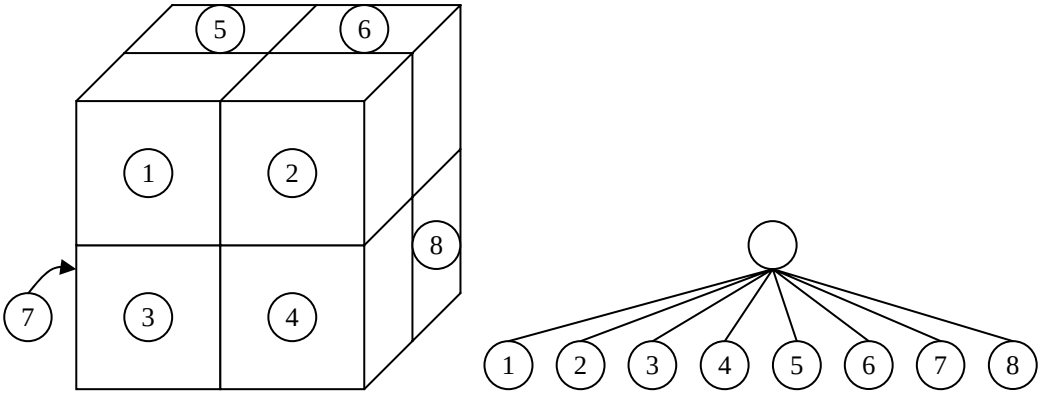

Figure 6-2 Three-dimensional space divided to octants and the corresponding octree node

The theoretical complexity of the octree search is O (log N) where N is the number of nodes in the octree. Performance of the implementation was tested with three building models. Building components of each model were uniformly distributed in a cubic formation. The formation of the biggest model had 10648 (22^3) components in total. Test results in Table 6-1 show that search time t grows almost linearly when the number of components C grows exponentially. Average search time for the each model is average time of million searches.

Table 6-1 Average octree search times

Number of Building Components, *C*	**Average Search Time, *t* [µs]**
125 (5^3)	17.6
1000 (10^3)	24.9
10648 (22^3)	56.8

6.3.3 Layout Plug-in

The layout plug-in is a module that calculates footprints and area objects of the building components and stores them for future use. A footprint is a closed polyline that represents the 2D geometry of a building component. An area object is an area generated from the footprint. Footprints and area objects are often needed in two-dimensional geometric analyses of route and compartmentation plug-ins. This is one of reasons why the handling of these objects is centralized to a separate plug-in. The other reason is to store created objects into a single cache, which improves performance and reduces memory consumption.

The layout plug-in has also services for handling area objects. The handling of area objects is based on Boolean operations of constructive area geometry (CAG). The CAG method creates a new area object by applying the binary union, difference, intersection, or exclusive-or (XOR) operation to two area objects. The Boolean operations used in CAG are demonstrated in Figure 6-3. The CAG is a part of the Java 2D API (Application Program Interface) and it can be used in the plug-in implementation without any third party library. The CAG operations are implemented in the Area class (J2SE 2005).

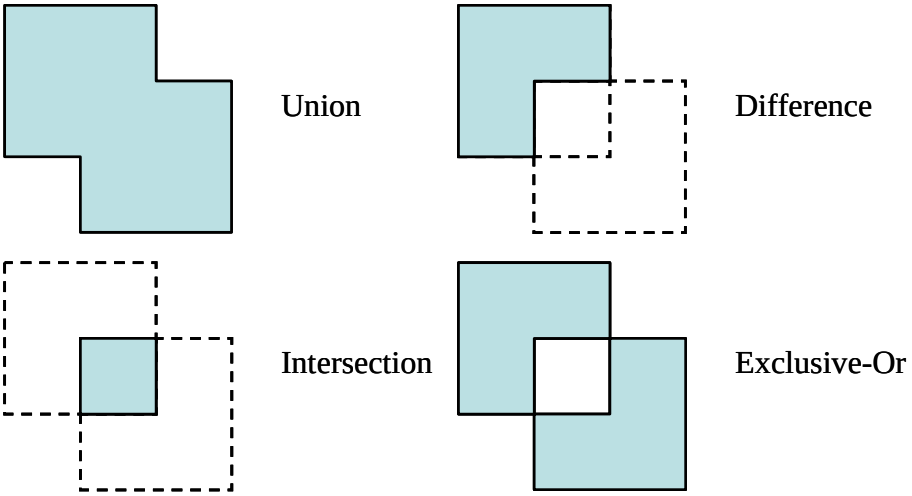

Figure 6-3 Boolean operations in CAG

The documentation of the Java 2D API does not give an indication of the complexity of a single CAG operation, but it should be some where between $O(E \log E)$ and $O(E^2)$, where E is the number of edges in operands. The average times of the empirical complexity test shown in Table 6-2 confirms the hypothesis. The test performed 10000 CAG operations for each operand pair. The number of edges E in polygonal operands was increased exponentially from 4 to 1024. The average times increased only slightly faster than the number of edges, which means that the complexity for Java 2D CAG operations is probably $O(E \log E)$.

Table 6-2 Average times for CAG operations of the Java 2D API

Edges, *E*	**Union, *t* [µs]**	**Difference, *t* [µs]**	**Intersection, *t* [µs]**	**Xor, *t* [µs]**
4	15,6	11,0	22.2	21,1
8	29,9	29,9	31.1	21,7
16	64,1	34,6	38.8	32,5
32	91,9	57,5	94.3	61,0
64	203,2	120,7	195.5	98,8
128	356,7	244,8	356.5	231,1
256	777,6	517,5	693.3	504,7
512	1610,2	963,0	1688.5	975,0
1024	2910,5	1958,0	3127.9	2407,6

The most essential functionalities where the CAG is used are increasing and decreasing the size of an area or a two-dimensional polygon. There are some trivial and efficient algorithms with $O(V)$ complexity, where V is the number of vertices in the polygon. These algorithms use vector mathematics and produce good results when the polygon is convex (a convex polygon contains all the line segments connecting any pair of its points), but

low quality results when the polygon has sharp corners, thin regions, or holes. A sharp corner should not expand to infinite as its angle approaches zero when the polygon is increased in size. A thin region should divide the polygon into two polygons when the polygon is decreased in size. Small holes should disappear when the polygon is increased in size. Implementing resizing with CAG is not as efficient, but the results are very good regardless of the geometry of polygon.

The algorithm implemented in this thesis is displayed in Figure 6-4. It can be divided to four phases:

1. An area object is created from the polygon.
2. The line segments of the polygon are iterated and area objects are created for each segment and sharp corner. The size of each area object depends on the required increment of the resize.
3. The area objects created in the previous phase are added (union) or subtracted (difference) from the original area object.
4. A new polygon is created from the resulting area object.

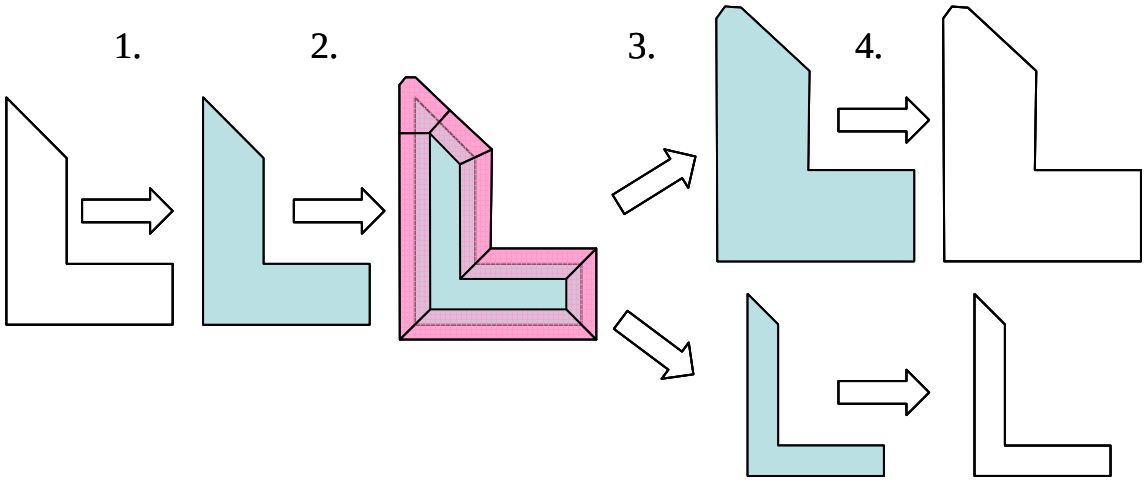

Figure 6-4 Method for increasing and decreasing the size of a polygon using CAG

Phases 1 and 4 are simple conversions between area and polygon objects. The complexity of one conversion is $O(E)$. Phase 2 has one conversion per each edge and sharp corner. The complexity of each conversion is $O(1)$, because there are not more than 6 edges per created polygon. The phase 3 has one CAG operation per each area object created in phase 2. If the complexity of CAG operation is $O(E \log E)$, then the complexity of the algorithm is

$$O(E) + O(E)O(1) + O(E)O(E \log E) + O(E) = O(E^2 \log E). \quad \textbf{6.1}$$

The footprints of the building components are often rectangular. The algorithm is optimized to handle rectangular polygons with a simple algorithm. The algorithm just moves the four corners inwards or outwards using vector mathematics. The complexity of this kind of algorithm is $O(1)$.

The layout plug-in also offers functionality for calculating the area value of an area object. It is needed in the compartmentation plug-in when the sizes of area objects are compared.

The Java 2D API does not include this functionality. The area values are calculated using the method for polygon area presented in the Computational Geometry in C (O'Rourke 1998). Let a polygon P have vertices v_0, v_1,…,v_{n-1} and let p be any point in the plane. Then the area is

$$A(P) = A(p, v_0, v_1) + A(p, v_1, v_2) + \cdots + A(p, v_{n-2}, v_{n-1}) + A(p, v_{n-1}, v_0). \quad \textbf{6.2}$$

If $v_i = (x_i, y_i)$, this expression is equivalent to the equation

$$2A(P) = \sum_{i=0}^{n-1} (x_i y_{i+1} - y_i x_{i+1}) = \sum_{i=0}^{n-1} (x_i + x_{i+1})(y_{i+1} - y_i). \quad \textbf{6.3}$$

The complexity of this algorithm is $O(E)$, where E is the number of edges in the polygon.

6.3.4 Route Plug-in

The route plug-in analyses the building information model and creates a route graph. This is the most complex and time consuming phase in the escape route analysis. It is done only once per building model, because the plug-in stores the graph into the building model when the user saves the model and loads it when the user opens the saved model. The route graph is used in route queries, which are done by the constraint module. The plug-in calculates the shortest route using the well known Dijkstra's shortest path algorithm (Dijkstra 1959). The route graph is created by analyzing the locations and geometries of the building components. This can be divided into five phases:

1. Finding building elements that connect spaces
All doors, openings, and stairs that lead to spaces are found. This is done by searching doors and stairs near the spaces using the model search tree plug-in. The locations of these candidates are still inspected closer before choosing because (for instance) a door can be near the space without being in direct contact with the space.

2. Finding interconnected spaces
Interconnected spaces are connected directly to each other without any walls between them (Figure 6-5). First spaces and walls near the spaces are searched using the model search tree plug-in. Then the segments of the space boundary that are not covered by walls are compared to corresponding segments of the spaces close to them. If common segments are found between the spaces, they are interconnected.

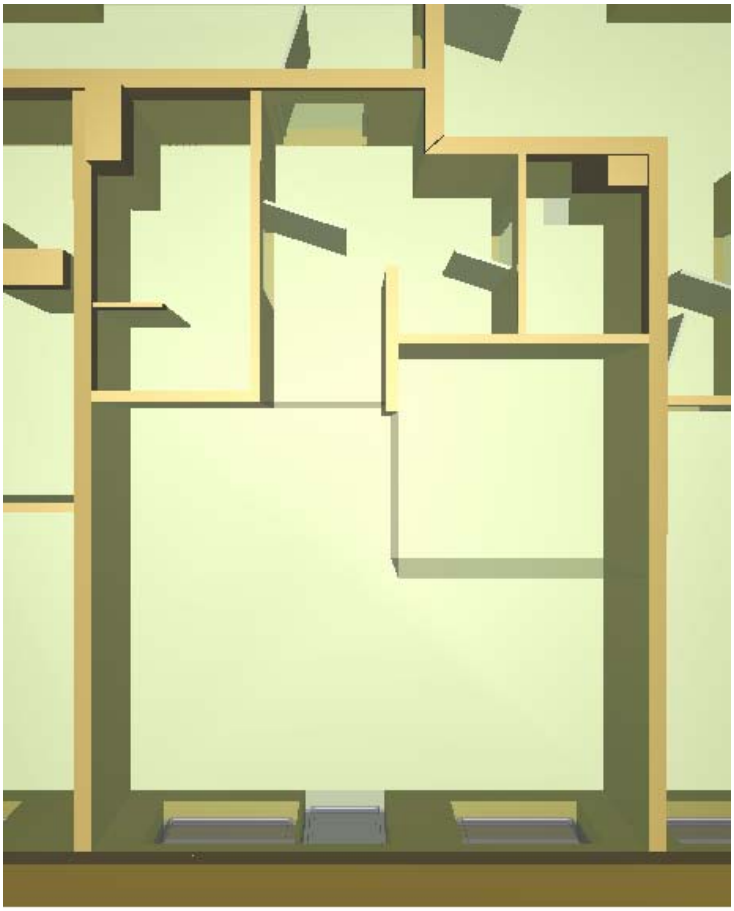

Figure 6-5 Three interconnected spaces, Piparminttu model, Skanska Oy

3. Merging interconnected spaces

Interconnected spaces are merged before a route graph of the spaces can be created. This phase is skipped if the space is not interconnected. Merging is done using CAG by combining the space area objects with an area created from the common segments (Figure 6-6). The common segments are found in the previous phase. This algorithm produces good results even when the spaces have narrow gabs between them or spaces are intersecting.

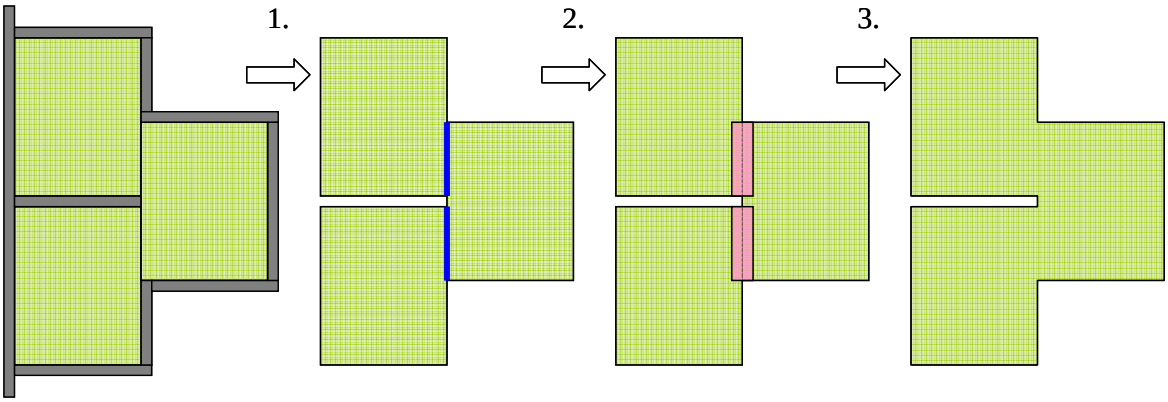

Figure 6-6 Method for merging interconnected spaces using CAG

4. Creating a graph for spaces

For this operation a space area object, connected doors, and stairs are needed as the source information. The idea is to create a graph of the area object that contains all routes between any door and stairs. At first the space area is converted to polygons (an outer boundary and possible holes), which are decomposed to a list of directed line-segments. The line-segments are used for creation of route segments inside the space area. In total, five different types of line segments are created. This is done using vector mathematics. All segments are split so that the resulting segments do not cross any other segment. The segments are split using a "brute force" algorithm, which tests each segment pair for intersection. The complexity of the algorithm is $O(S^2)$, where S is the number of segments, since there are

$$(S-1)+(S-2)+\ldots+1=S(S-1)/2 \qquad \textbf{6.4}$$

different segment pairs. There is a faster and well-known "Bentley-Ottmann Algorithm" (Bentley & Ottmann 1979), which computes intersections in $O(S \log S)$ time. It is not needed because the bottleneck is currently in algorithms that use CAG. The split segments that are located outside the area are discarded. The remaining segments form the graph. The graph edges and nodes are seen in the graph testing window in Figure 6-7.

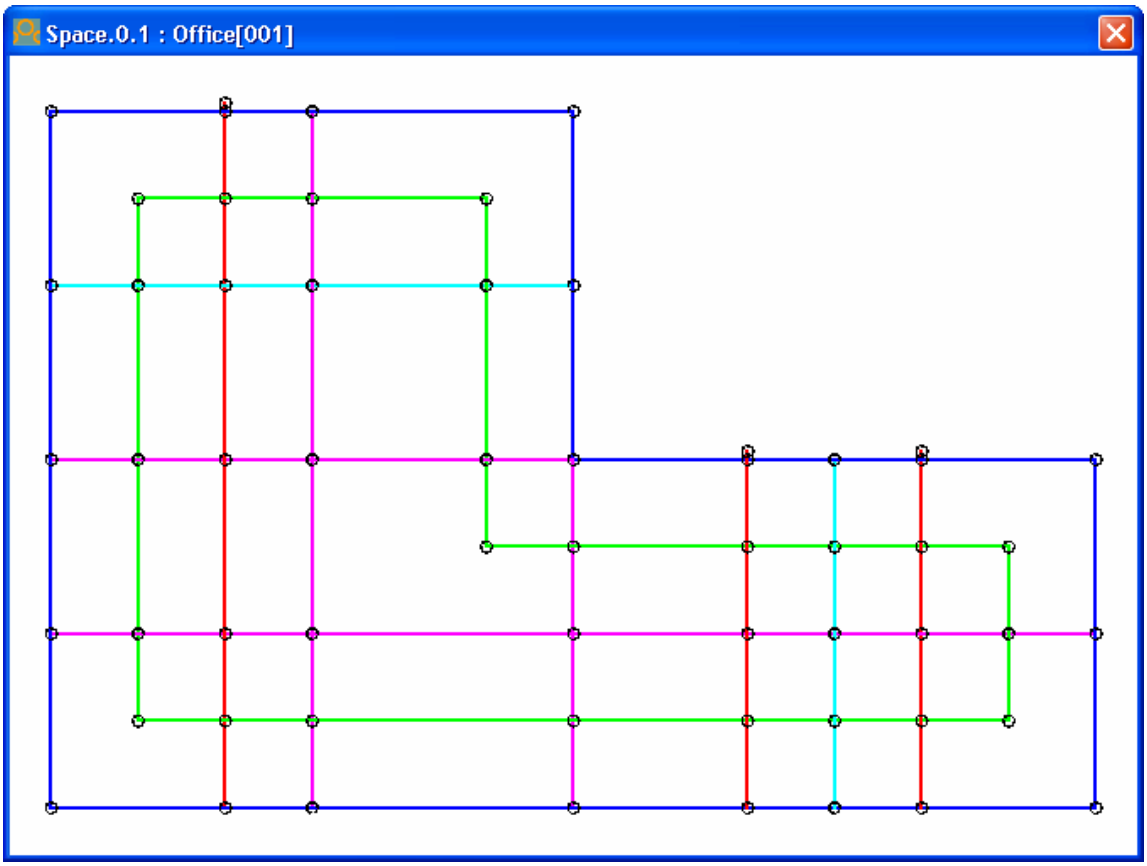

Figure 6-7 Route edges in ascending priority order in testing window: red, magenta, green, cyan, and blue

In the testing window the edges are colored by cost. The edges with higher cost have a penalty in route calculation. This way routes use edges that have smaller cost when possible. The costs are designed so that the route goes along the centre line of the space between the center points of the doors. This way the routes are acceptable for fire code

compliance studied in paragraph 4.4. Normally the route goes from a door to the magenta centre line using the red segments. The red edges lead to the doors and have the lowest priority. The magenta edges are located in the centre of the walls and have the next lowest priority. The green edges are parallel and cyan edges perpendicular to the walls. The green edges have lower priority than cyan. The blue edges have the highest priority and are located in the boundary of the area. The blue edges are seldom used by routes.

5. Building the final route graph
The route graph is created by combining the space graphs into a single master graph. To assure small memory consumption and fast route queries, only the edges of the routes between doors and stairs are added. These edges are searched by Dijkstra's shortest path algorithm (Dijkstra 1959). The other edges are discarded.

6.3.5 Compartmentation Plug-in

Buildings are often divided into compartments that serve different purposes. Compartments are two-dimensional wall bounded areas. The compartmentation Plug-in is used to create and modify compartments for different purposes. This is needed because current building information models seldom have information about compartments. The plug-in can handle three different types of compartments: fire compartments, gross area compartments, and secure compartments. Escape route analysis uses gross area and fire compartments. Gross area compartments are bounded by exterior walls and fire compartments are bounded by exterior walls and fire walls.

The compartmentation plug-in has a user interface for creating and handling compartments (Figure 6-8). The UI has tool buttons for creating and editing compartments and a tree control for browsing the compartments and bounding walls. The creation of compartment is a semiautomatic process. The user chooses the compartment type and the method for selecting walls. The plug-in analyses the selected walls and creates the compartments bounded by the walls. The compartments selected by the user are visualized in 3D. The user can modify the compartments by adding or removing bounding walls. After these operations the plug-in automatically updates the affected compartments.

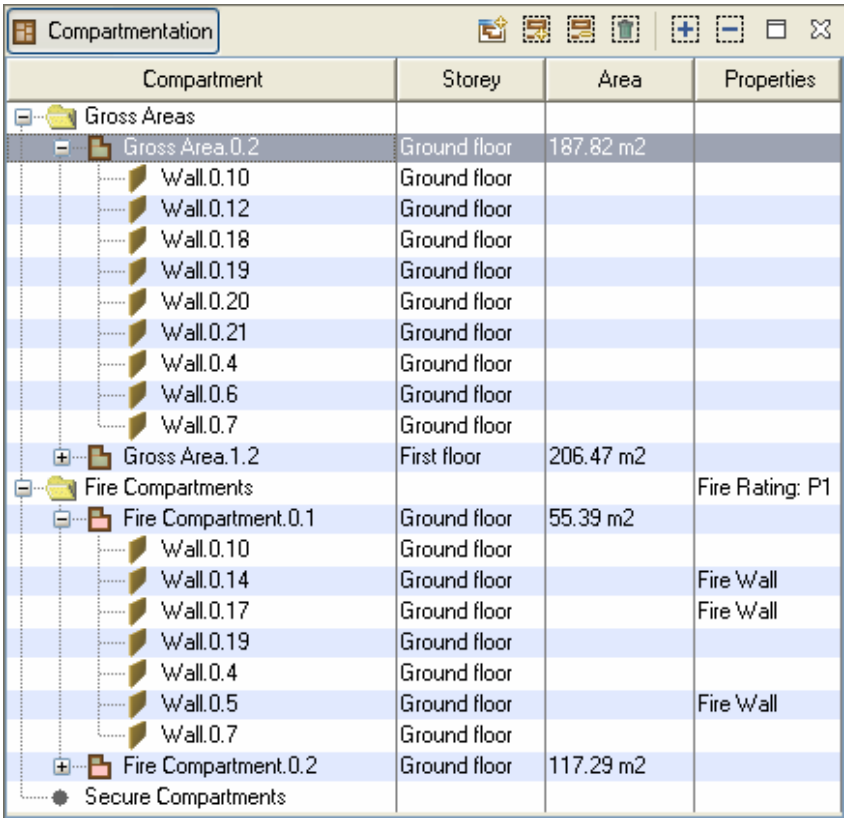

Figure 6-8 Compartmentation View

Compartment creation from the set of walls is challenging task. The geometries of walls are inconsistent, there can be gaps between walls, and the walls sometimes intersect each other. Constructive area geometry (CAG) can be used to solve these problems. The algorithm is presented in Figure 6-9 is following:

1. The bottom area objects of the walls are increased in size using the method of the layout plug-in described in Section 6.3.3.
2. The increased area objects are combined into one area object using the CAG union operation.
3. a) Outer compartments (gross areas) are extracted from the outer boundary of the combined area that is decreased in size.
   b) The inner compartments (fire compartments) are extracted from the holes of the combined area that are increased in size.

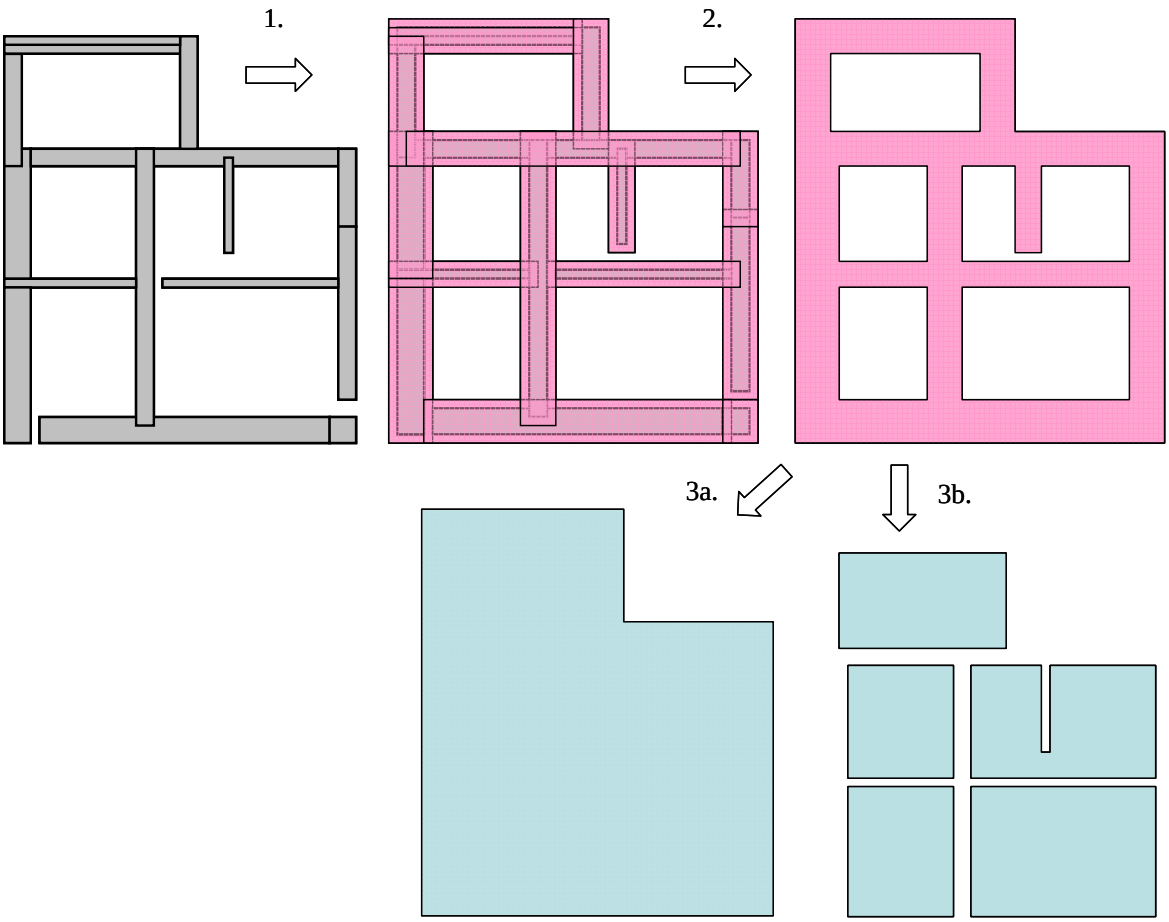

Figure 6-9 Method for creating compartments from a set of walls using CAG

The bounding walls of the compartment are the walls whose enlarged bottom area intersects with the compartment area. The intersection is tested with the intersection operation of CAG. If the result of intersection is an empty area there is no intersection and if the result area is smaller than the area of the increased bottom area there is a partial intersection. The sizes of area objects can be compared by comparing their calculated area values obtained from the layout plug-in.

6.4 Constraint Module

6.4.1 Overview

Constraints are small program modules that can check a building information model from one or more aspects. Constraints are parametric and they can be configured to check the model against different requirements. The user can configure the parameters in the constraint parameter view (Figure 6-11). Problems found in checking are displayed in a tree of the constraint results view (Figure 6-12) where they are organized by category. The constraint may also have tools, which are like small applications that support the constraints. Because of the nature of constraint tools the appearance and functionality varies considerably between the different constraints. Typically constraint tools are used for advanced visualization of the checking results. Tools are located in the constraint tools

view (Figure 6-14). In addition to checking the model, constraints have the ability to report information from the model. (Solibri 2004)

6.4.2 Checking

The escape route constraint module is responsible for checking the model against the escape route requirements and representing the checking results. Since most of the functionality needed for the checking is located in plug-ins, the constraint module can focus on analyzing and representing the results.

Model checking has three phases: pre-check, check, and post-check. In the pre-check the existence of fire compartments and exit doors are checked. Check method is called once for every space component in the building. Figure 6-10 shows the flow of the check method. The last phase contains only the checking of route widths. The widths cannot be checked earlier because the information for all the routes is needed. The information is collected during the check phase.

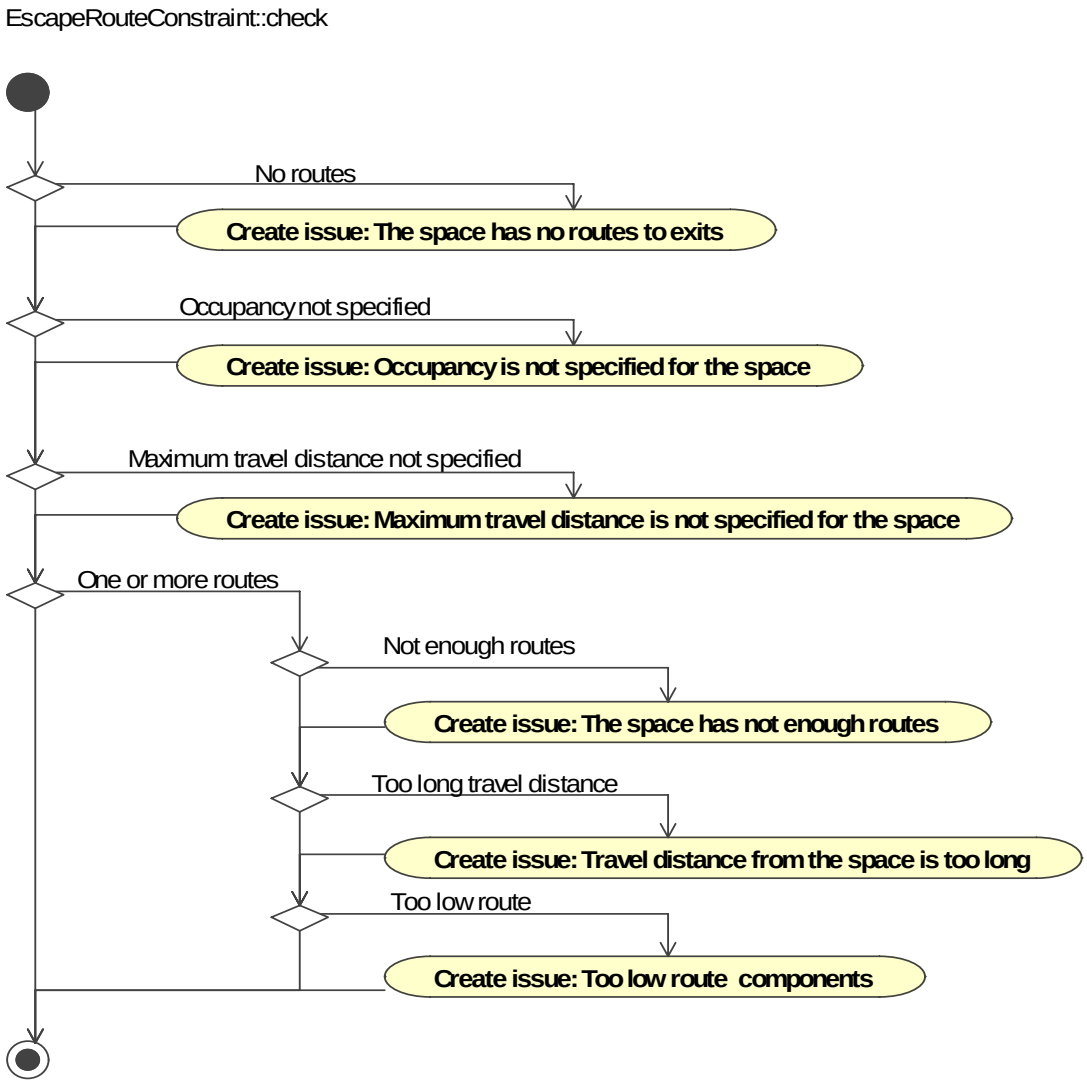

Figure 6-10 Flow chart of the check method

6.4.3 Constraint Parameter View

The constraint module must be configurable because fire codes differ from code to code. All the fire codes examined have at least the following requirements:

- Minimum number of escape routes
- Formulas for determining the occupant counts of spaces
- Maximum travel distance (usually depends on the number of routes)
- Minimum width of escape route (usually depends on the number of occupants)
- Minimum height of escape route

The parameter view is displayed in Figure 6-11. It has one list parameter, three table parameters, and one numeric parameter. The list parameter Exit List contains exits at the termination of an escape route from a building. The list is building-specific and the user must always fill the list with the exits of the current building. The first table parameter is General Requirements. Each row in the table contains data for one usage type. The usage type of the row is defined in the first cell and the data in the rest. The second table parameter Space Requirements defines the usage types of the spaces. One row in the table can define the usage type of a single space or a set of spaces. The matching spaces are identified by type, name, and number. It is also possible to set a fixed occupant count for a space or a set of spaces by filling the Occupant Count of the row. This is designed for spaces where the maximum number of occupants is known. The third table parameter, Minimum Exit Passageway Width, defines the minimum widths for escape routes. The first cell in the row tells for what number of occupants the requirements are. The minimum total widths and single widths are in the rest of the cells. The value of the last parameter Minimum Exit Passageway Height is the minimum height of the escape routes.

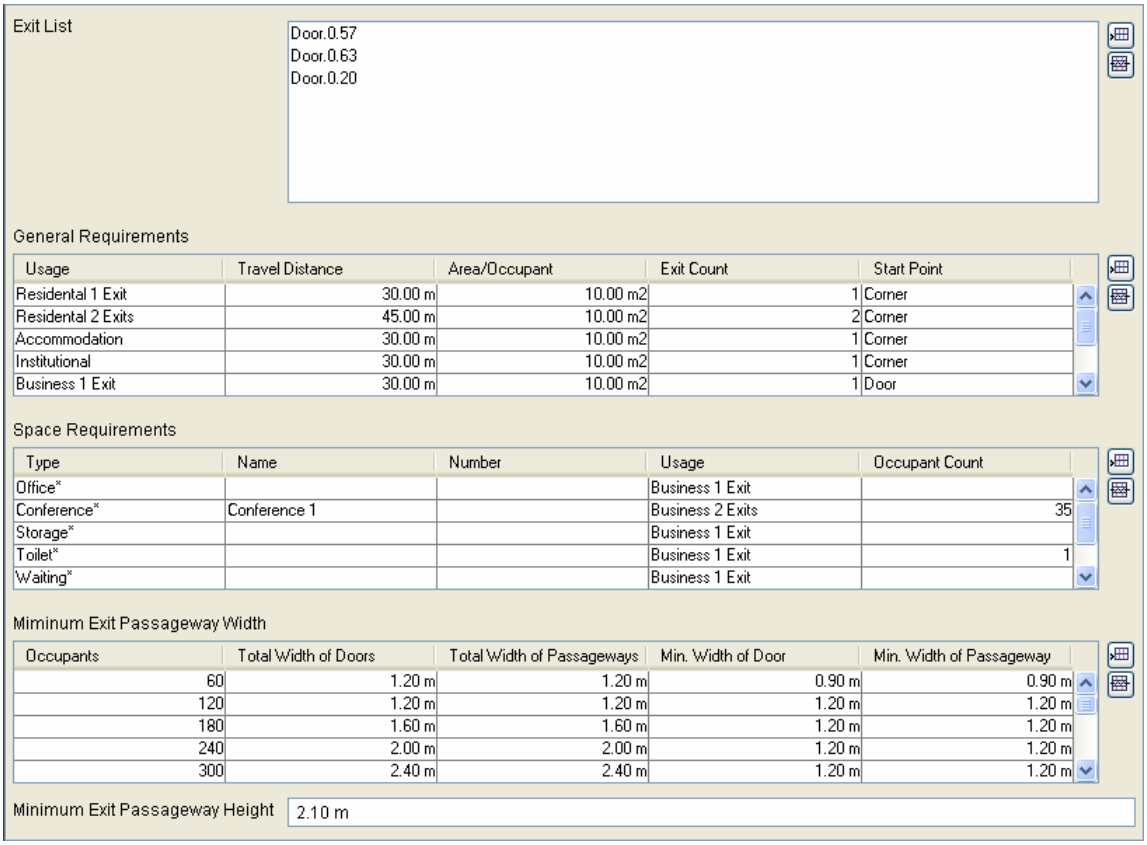

Figure 6-11 Constraint parameter view

6.4.4 Constraint Results View

Constraint results are represented to the user in a tree component in the constraint results view. The tree consists of category, issue, and building element nodes. The purpose of a category is to group a set of similar issues. An issue describes a problem which is related to a set of building elements. The different types of nodes are seen in Figure 6-12.

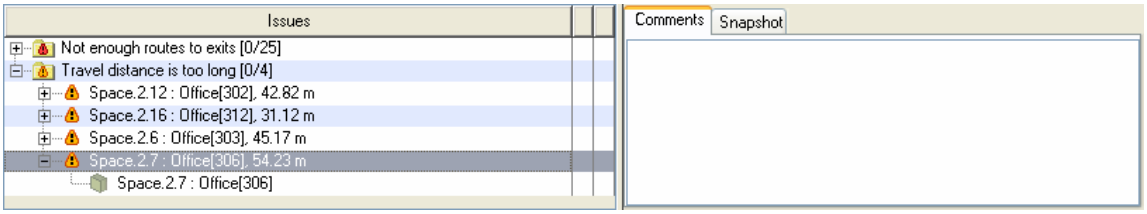

Figure 6-12 Constraint results view

Table 6-3 contains all the categories and issues that the result tree of the escape route constraint can have. The varying parts of the issue descriptions are marked with "<" and ">" marks.

Table 6-3 Categories and issues of the escape route constraint

Category name	Issue description
Missing information	Exits are not specified. Exits are specified in constraint parameters.
	Occupancy is not specified for the space <space identifier>. Occupancies are specified in constraint parameters.
	Maximum travel distance is not specified for the space <space identifier>. Travel distances are specified in constraint parameters.
	The model does not have any fire compartments. They can be added to the model from Window menu (Window -> Views -> Compartmentation).
No routes to exits	Space <space identifier> has no routes to exits. Check that all exits are added to the constraint parameters.
Not enough routes to exits	Space <space identifier> has <number> escape routes. There should be at least <minimum number> routes.
Travel distance is too long	Travel distance from space <space identifier> to safe place is <length>. The maximum travel distance is <maximum length>.
Exit passageway is too low	Height of space <space identifier> is <height>. The minimum exit passageway height is <required height>.
Exit passageway is too narrow	Width of door <door identifier> is <width>. The minimum door width for <number> occupants is <required width>.
	Width of space <space identifier>. The minimum passageway width for <number> occupants is <width>.
	Total door width is <width>. The minimum total door width for <number> occupants is <required width>.
	Total passageway width is <width>. The minimum total passageway width for <number> occupants is <required width>.

6.4.5 Constraint Tools View

The constraint results view shows the problems found in checking. Only routes that have something wrong are shown. The constraint tools panel of the escape route constraint is designed so that the user can see any route, fire compartment or occupant count visualized in the 3D view (Figure 6-13).

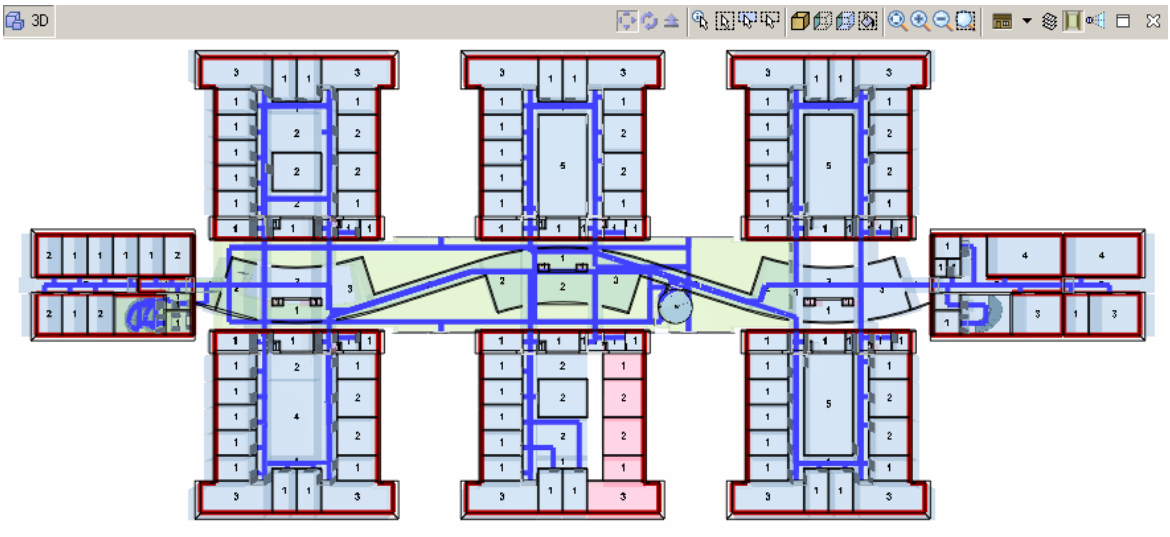

Figure 6-13 Visualization of the escape route analysis results

The tools panel that controls the visualization consists of checkboxes and a tree (Figure 6-14). The checkboxes in the user interface are used to specify the visualizations that are shown when a node in the tree is selected. The storeys are root nodes of the tree. The storeys have spaces as child nodes and the spaces have routes as child nodes. The user can see all the routes on one storey in the 3D view (Figure 6-13) by selecting a storey node in the tree. The visualization of other kind of nodes works in a similar way.

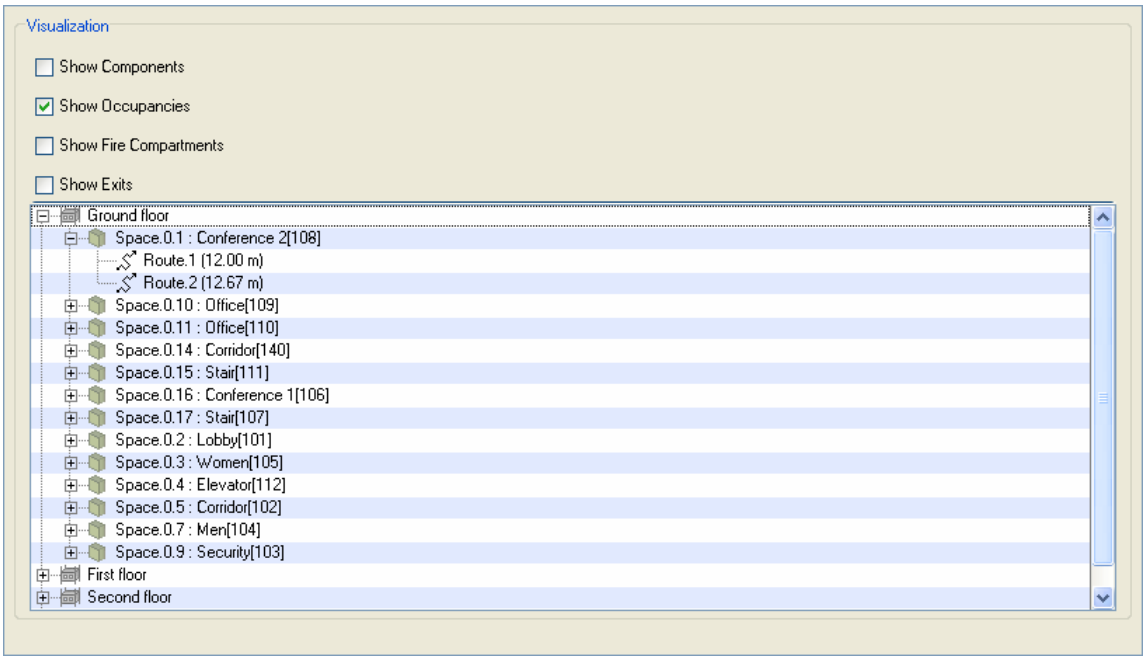

Figure 6-14 Constraint tools view

7 Testing and Quality Assurance

7.1 Introduction

This chapter concentrates on the testing and quality assurance of the module. The module was tested using automatic unit and module testing. The module was evaluated against the requirements using manual acceptance testing. The main emphasis was in automatic testing. Automatic testing proved to be very valuable during the implementation of this project and is now used in all projects. Several times it helped to find and locate critical errors. It also turned to be very useful in refactoring and debugging. Currently automatic testing is an important part of software process in Solibri. Automatic testing has also increased the number of tests written by coders and changed the attitude to testing.

7.2 Automatic Testing

Unit and module tests are run automatically in a separate testing environment. The testing environment consists of a PC with the testing framework installed. The PC must be connected to the intranet because all the testing material is there. The testing framework can be installed on the following operating systems: Microsoft Windows 2000/XP and Apple Mac OS X. The central software of the framework is Apache Ant, which is a Java-based build tool that can also run JUnit tests (Apache Ant 2005). The Ant reads and executes XML-based (EXtensible Markup Language) configuration files.

The test is normally run once a day. The test script checks out the latest source codes and test case codes from the version control system and compiles them. After the tests are run, the results are collected and published on the Solibri intranet. Ant runs also a small Java application which writes an automatic testing summary table. The summary table shown in Figure 7-1 contains links to all test reports and numeric data such as the total number of test cases, number of passed and failed test cases, and the total testing time in seconds. It can be seen in Figure 7-1 that on 6th July source code that committed to the version control was reason to a failure. The source code was fixed the next day.

Report	Date	Time Report	Tests	Failures	Errors	Success rate	Time
Latest	02.08.2005 03:10	Latest	217	0	0	100.00%	12681.728
Old 1	20.07.2005 03:11	Old 1	217	0	0	100.00%	12770.954
Old 2	19.07.2005 03:12	Old 2	217	0	0	100.00%	12804.172
Old 3	18.07.2005 13:52	Old 3	217	0	0	100.00%	13067.986
Old 4	12.07.2005 03:11	Old 4	217	0	0	100.00%	12784.872
Old 5	09.07.2005 03:13	Old 5	217	0	0	100.00%	12897.732
Old 6	08.07.2005 03:18	Old 6	217	0	0	100.00%	13224.235
Old 7	**07.07.2005 01:04**	**Old 7**	**217**	**1**	**0**	**99.54%**	**5203.540**
Old 8	06.07.2005 03:12	Old 8	217	0	0	100.00%	12832.718
Old 9	05.07.2005 03:11	Old 9	217	0	0	100.00%	12752.538

Figure 7-1 Ten topmost rows of the automatic testing summary table (August 2, 2005)

The unit tests are normal JUnit test cases that test classes of the modules. The module tests are extended JUnit test cases. The test cases have helper methods, which make them easy and fast to write. The helper methods are designed to decrease the bad smells that are specific for test code (van Deursen et al 2001). The following simple module test assures that the escape route module does not find any problems with a faultless building model.

```
public void testRoutes28() {
    String modelName = "179/routes28.ifc";
    String csetName = "179/EscapeRoutes.cset";
    init(modelName, csetName);

    Smodel model = (SModel)ModelChecker.getModelHandlingPlugin().getCurrentModel();
    EscapeRouteConstraint cons = (EscapeRouteConstraint) findConstraint(0);

    // Create fire compartments
    CompartmentationPlugin.getInstance().createCompartments(
    SFireCompartment.class, CompartmentationPlugin.ALL_WALLS_METHOD);

    // Set exits
    addExit(model, cons, "2ZkPW8$Cf50ubLED_ygE$Y");

    // Run constraint set
    runCset();

    // Check that there are no problem found in checking
    Collection cats = cons.getCategories();
    assertNotNull("Categories are null", cats);
    assertEquals("Wrong number of categories", 0, cats.size());
}
```

7.3 Acceptance Testing

Acceptance testing is conducted to determine whether a system satisfies its acceptance criteria. It normally is normally done by actual users. In this thesis the software is validated by the author against the requirements. The system is accepted if it satisfies all requirements with the highest priority and half of the requirements with medium priority. Two real life building models were used in the validation. The first building was an office building designed with Autodesk ADT (Figure 7-2) and the second was a housing block designed with Graphisoft ArchiCAD (Figure 7-3).

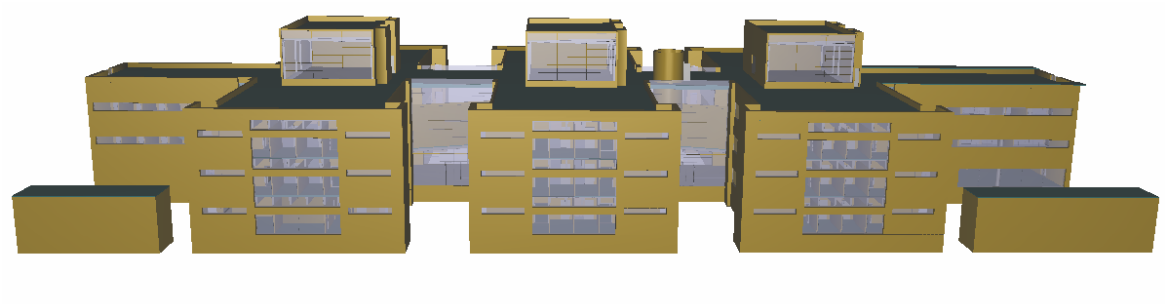

Figure 7-2 Digitalo model in SMC, Senaatti-kiinteistöt

Figure 7-3 Piparminttu model in SMC, Skanska Oy

As the result of acceptance testing the system satisfies all but six medium or low priority requirements. These six requirements are listed in Table 7-1. This means that the system satisfies its acceptance criteria.

Table 7-1 Requirements that the system does not satisfy at the moment

ID	Short Description
R9	The user must be able to specify the measurement method of the escape route (direct linear measurement, indirect linear measurement, and wall aligned linear measurement).
R13	The user must be able to specify a person number when the doors must open to the direction of escape.
R14	The user must be able to specify the minimal acceptable number of different escape routes the spaces must have with different occupancy numbers.
R18	The user must be able to produce a report of the calculations.
R23	The system must automatically detect fire compartments.
R24	The system must automatically detect exterior exits (door, windows, and openings).

7.4 Quality Assurance

The quality of the source code was measured using CCCC tool (C and C++ Code Counter). Despite the name, CCCC also analyzes Java code. It generates a report on various metrics of the code. The supported metrics include lines of code, lines of code per line of comment, McCabe's cyclomatic complexity, and metrics related to object oriented design.

McCabe’s cyclomatic complexity is the most widely used software complexity metrics. It measures the number of linearly-independent paths through a program module. In this thesis all source code was refactored so that the maximum complexity number for each Java class was 20 and 10 for each method in the classes. The complexity number was also useful for designing unit testing so that the focus is on the methods with higher risk.

The other metrics measured the classic characteristics of object oriented design: visibility, inheritance, reuse, and coupling. The visibility metric revealed the classes that had too many methods and fields accessible to other objects. The inheritance metric showed the depth of inheritance tree. The reuse metric counted the number of direct inherited objects. The coupling metric was the most important of these metrics. It helped to spot classes that had too high coupling.

8 Conclusions and Future Work

8.1 Overview

This thesis has been carried out at Solibri Oy. The aim of this thesis was to define the set of information that is needed to automatically perform escape route analysis, the methods for performing the analysis, and the means of presenting the results of the analysis. The thesis has of two parts: study and work.

The study of the thesis concentrates on building information modeling (BIM) and escape route analysis. The study is mainly based on literature and interviews. BIM is becoming approach that is widely adopted in the AEC industry. BIM offers many benefits and totally new and powerful ways of working over the building life cycle. Implementing BIM is challenging, since it requires a major paradigm shift from drawing-based to model-based operations. One of the new use cases is automated building code checking. The thesis studies regulations of fire codes that concern escape routes.

The work consists of the fundamental activities of software processes: definition, design, implementation, and testing. The activities were carried out simultaneously. The definition consisted of common activities of requirements engineering. The focus in the work was on the design and implementation. Automatic testing had an important role in testing. Finally, the system was acceptance tested against the requirements.

8.2 Results

As the result of this work an escape route analysis module was implemented and included into the Solibri Model Checker product. The work shows that current building information models have adequate information for automatic validation of escape routes against the most essential regulations of building fire codes. High similarity between the regulations in different geographic areas made it possible to create a common user interface that allows the use of different fire codes as input for the analysis. The analysis results in a list of problems that violate the regulations. The users can browse this list and visualize problematic routes in the 3D view.

The escape route analysis module has already been used in some ongoing building projects. Typically the module has revealed some modeling errors such as missing doors or spaces. These kinds of errors are equally problematic for almost any other use of the building model.

One of the biggest challenges in the analysis was the varying quality of the building information models. The problem was solved by analyzing the geometry and location of building elements instead of relying on relations between them. This required a set of efficient geometry algorithms that give reliable and high quality results. The three most notable algorithms developed are an algorithm for increasing and decreasing polygons in size, an algorithm for merging interconnected spaces, and an algorithm for creating compartments. Constructive area geometry (CAG) was the key technology in these algorithms in order to obtain high quality results.

The acceptance testing carried out at end of the project showed that the module meets the requirements set. The module has still small problems in certain areas. These problems can be solved when the module is developed further.

An automatic testing system developed to test the modules and algorithms proved to be valuable. It is run every night and reports possible problems that changes to source code have caused. The testing system is now important part of the software process in Solibri Oy.

8.3 Future Work

The results of the thesis and feedback from the users have brought out some interesting ideas for further development in escape route analysis. More ideas can be generated by systematically collecting feedback from users or by organizing a field study.

Accessibility is one of the potential areas into which the module could be expanded. Building codes often have separate regulations for accessibility. The accessibility codes contain (for instance) regulations that concern usability of a wheel chair in a building.

In addition to these new ideas, there are some requirements defined that were not implemented. The development of the module continues by implementing the most valuable of the remaining requirements and by fixing possible problems.

References

Apache Ant, Apache Ant www-pages, Apache Software Foundation, Referred 2005-8-10, Available at http://ant.apache.org, 2005.

AR-4, Process Definitions Means of Escape – In Case of Fire (Draft 2), IFC R3.0 Domain Project Documentation, 1998.

Autodesk, Autodesk www-pages, Autodesk, Inc., Referred 2005-8-15, Available at http://www.autodesk.com, 2005.

Autodesk, Building Information Modeling in Practice, White paper, Autodesk, Inc., 2003.

Becker, D., Bye-bye, Blueprint: 3D Modeling Catches on, ZDNet News, 2004.

Bentley, J. L., and Ottmann, T. A. Algorithms for Reporting and Counting Geometric Intersections, IEEE Transaction on Computers C 28, pp. 643-647, 1979.

Building Code of the City of New York, Department of Citywide Administrative Services, 2004.

Bukowski, R. W., and Kuligowski, E. D., The Basis for Egress Provisions in U.S. Building Codes, NIST Building and Fire Research Laboratory, Interflam 2004 (Interflam '04), International Interflam Conference, 10th Proceedings, Volume 1, July 5-7, 2004.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design Patters, Addison-Wesley, 1995.

Crowd Dynamics, Crowd Dynamics Limited www-pages, Crowd Dynamics Limited, Referred 2005-7-12, Available at http://www.crowddynamics.com, 2005.

Dijkstra, E. W., A Note on Two Problems in Connexion With Graphs, Numerische Mathematik 1, 269–271, 1959.

Eclipse, Foundation, Eclipse www-pages, Eclipse Foundation, Referred 2005-6-10, Available at http://www.eclipse.org, 2005.

Facilities Information Council, Working definition by Facilities Information Council, November, 2004

Ferris, S., To BIM or not to BIM?, CADalyst Magazine, February, 2005.

Finland's environmental administration, Rakennusten paloturvallisuus & Paloturvallisuus korjausrakentamisessa, Ympäristöministeriö, 2003.

Finland's environmental administration, Suomen rakentamismääräyskokoelma E1, Ympäristöministeriö, 2002.

GeoPraxis, AEC Design Practice Study 2004 - Final Report, GeoPraxis, 2004.

Han, C., Kunz, C. J., and Law, K. H., Computer Models & Methods for a Disabled Access Analysis Design Environment, CIFE Technical Report #123, July, 2000

Han, C., Kunz, C. J., and Law, K. H., Making Automated Building Code Checking a Reality, Facility Management Journal, September/October, pp. 22-28., 1997.

Hearn, D., and Baker, M. P., Computer Graphics, C Version, 2nd edition, Prentice Hall, 1994.

Hong Kong Buildings Department, The Provision of Means of Escape in Case of Fire, Hong Kong Buildings Department, 1996.

Howell, I., and Batcheler, B., Building Information Modeling Two Years Later – Huge Potential, Some Success and Several Limitations, White paper, Newforma, January, 2005.

IAI International, IFC2x2 Final Online Documentation, International Alliance for Interoperability, Available at http://www.iai-international.org/Model/documentation/R2x2_Final/Online_Documents/index.htm, 2003

IAI International, International Alliance for Interoperability www-pages, International Alliance for Interoperability, Referred 2004-10-7, Available at http://www.iai-international.org/iai_international, 2004.

ISO 10303-11, Industrial automation systems and integration - Product data representation and exchange - Part 11: The EXPRESS Language Reference Manual, ISO, Geneva, Switzerland, 1994.

Isoaho, J., A-Konsultit Oy, Interview, Juni, 2004.

J2SE, Java 2 Platform SE 5.0 Documentation, Sun Microsystems, Available at http://java.sun.com/j2se/1.5.0, 2005.

Java3D, Java3D 1.3.2 Documentation, Available at https://java3d.dev.java.net, 2005.

Khemlani, L., Should We BIM? Pushing the State of the Art in AEC, CADENCE Magazine, June, 2003.

Khemlani, L., The Euraka Tower: A Case Study of Advanced BIM implementation, AECbytes, June, 2004.

Kotonya, G., and Sommerville, I., Requirements Engineering - Processes and Techniques, John Wiley & Sons, New York, 1998.

Kuligowski, E. D., Review of 28 Egress Models, NIST SP 1032, January 2005, Workshop on Building Occupant Movement During Fire Emergencies, Proceedings, Session 4.4. June 10-11, 2004.

Kuligowski, E. D., and Milke, J. A., Performance-Based Design of a Hotel Building Using Two Egress Models: A Comparison of the Results, Human Behavior in Fire: Public Fire Safety - Professionals in Partnership, International Symposium, 3rd. Proceedings. September 1-3, 2004, Belfast, N. Ireland, Interscience Communications Ltd., London, England, 399-410 pp, 2004.

Lau, G. T. , Kerrigan, S., and Law., K. H., An Information Infrastructure for Government Regulations, Proceedings of the 13th Workshop on Information Technology and Systems (WITS'03), pp. 37-42, Seattle, WA, Dec 13-14, 2003.

Lilja, O., Finland Ministry of the Environment, Interview, September, 2004.

Miller, K., Rämä, M., and Karvinen, J., Helsinki Building Inspectorate Department, Interview, November, 2004.

NFPA, 1984 Fire almanac, National Fire Protection Association, Quincy, 1983.

NIBS, Building Sciences, The National Institute of Building Sciences, Vol. 29, 2005.

No Magic, MagicDraw www-pages, No Magic Inc., Referred 2005-8-21, Available at http://www.magicdraw.com, 2005.

Ontario Fire Code, Canadian Government Publishing, 1997.

O'Rourke, J., Computational Geometry in C, 2nd edition, Cambridge University Press, 1998.

ProIt, Arkkitehdin tuotemallisuunnittelu, Innovarch Oy, 2005.

ProIt, Tuotemallinnus rakennesuunnittelussa, perusteet ja ohjeita I, Finnmap Consulting Oy, Rakennusteollisuus RT ry, 2004.

Requicha, A. A. G., Mathematical models of rigid solids, Tech. Memo. No. 28, Production Automation Project, University of Rochester, 1977.

Rong, X., Wawan, S., and Zhiyong, H., Code Checking and Visualization of an Architecture Design, 15th IEEE Visualization, 2004.

SFPE, Engineering Guide to Human Behavior in Fire, Society of Fire Protection Engineers, Technical Report, 2002.

Solibri, Overview of the Solibri Application Framework Architecture, Solibri Oy, 2001

Solibri, Solibri Online Help, Solibri Oy, 2004.

Sommerville, I., and Sawyer, P., Requirements Engineering: A Good Practice Guide, John Wiley & Sons, New York, 1997.

van Deursen A., Moonen L., van den Bergh A. and Kok G., Refactoring Test Code. In Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001), pp. 92-95. University of Cagliary, 2001.

W3C, Extensible Markup Language (XML) 1.0 3rd edition, World Wide Web Consortium, February, 2004.

Appendix

This appendix contains the UML class diagrams of the plug-in modules. The diagrams are generated from the source codes using MagicDraw UML tool (No Magic 2005). All methods and attributes related to user interface are hidden manually. Constructors and private methods are not show in the diagrams. There are many external types used in classes that are not defined in the diagrams. Most of them are defined in Java 3D 1.3.2 (Java3D 2005) or J2SE 5.0 (J2SE 2005) documentations. The rest types such as DefaultPlugin and IComponent are part of Solibri Application Engine or Solibri Application Framework.

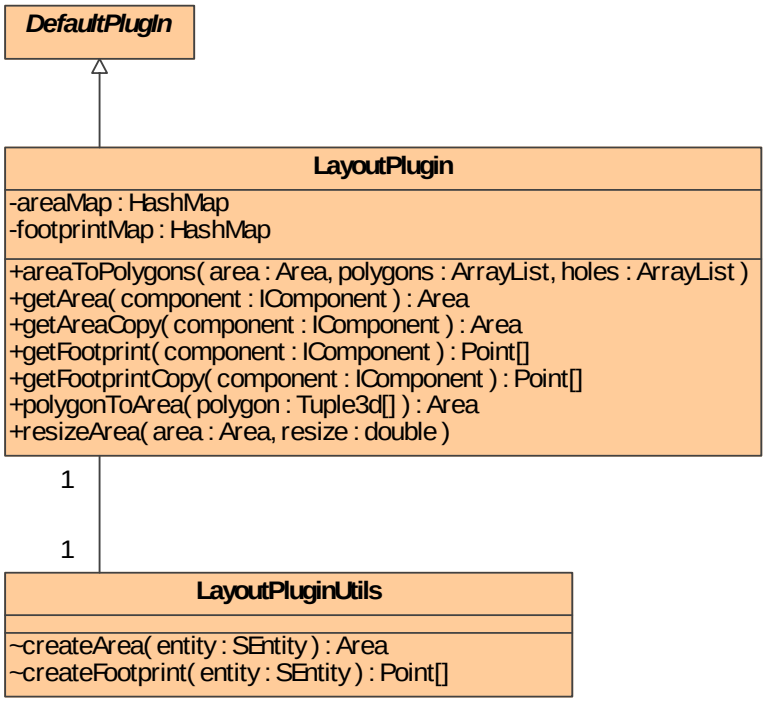

Diagram 1 Class diagram of the layout plug-in

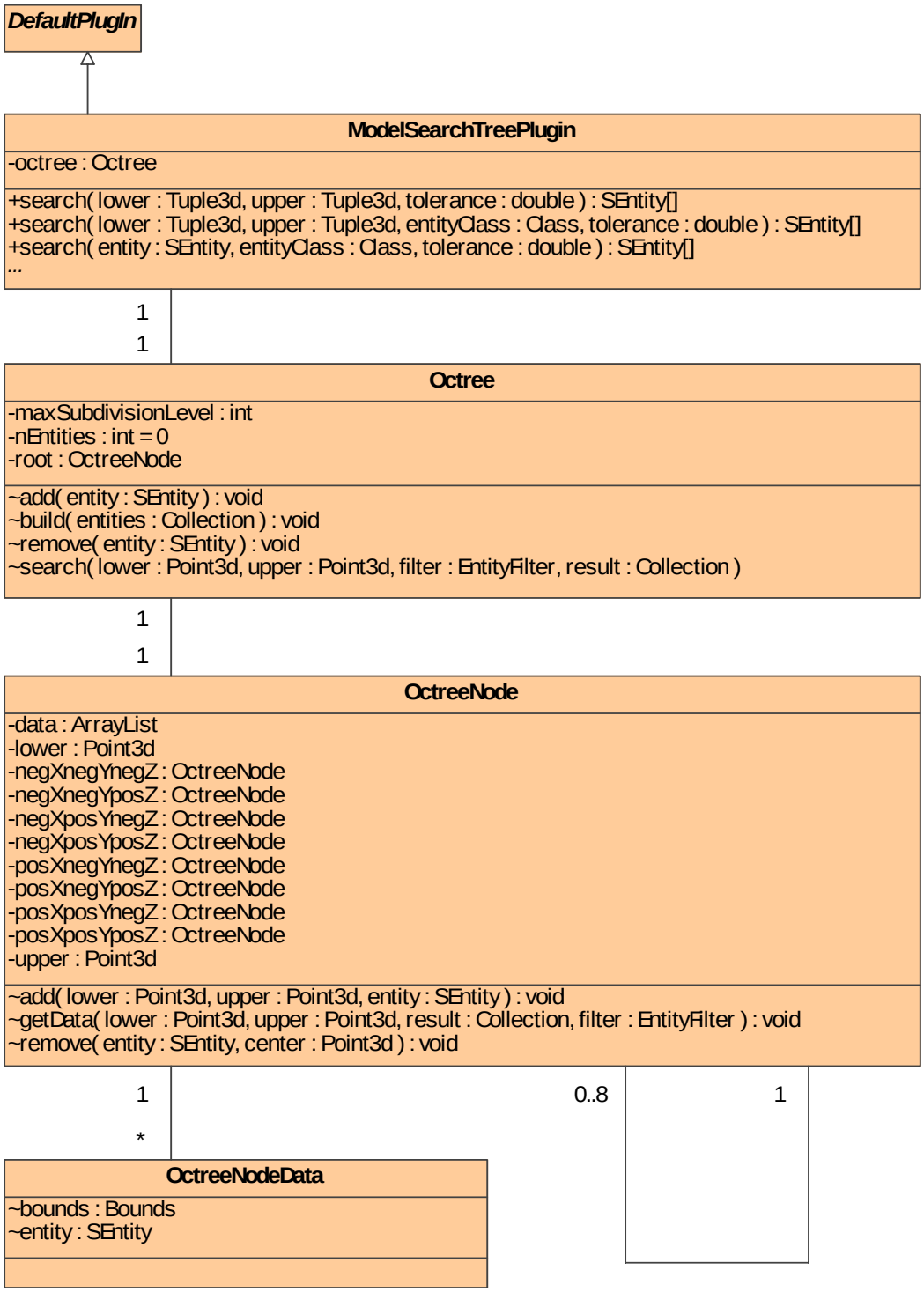

Diagram 2 Class diagram of the model search tree plug-in

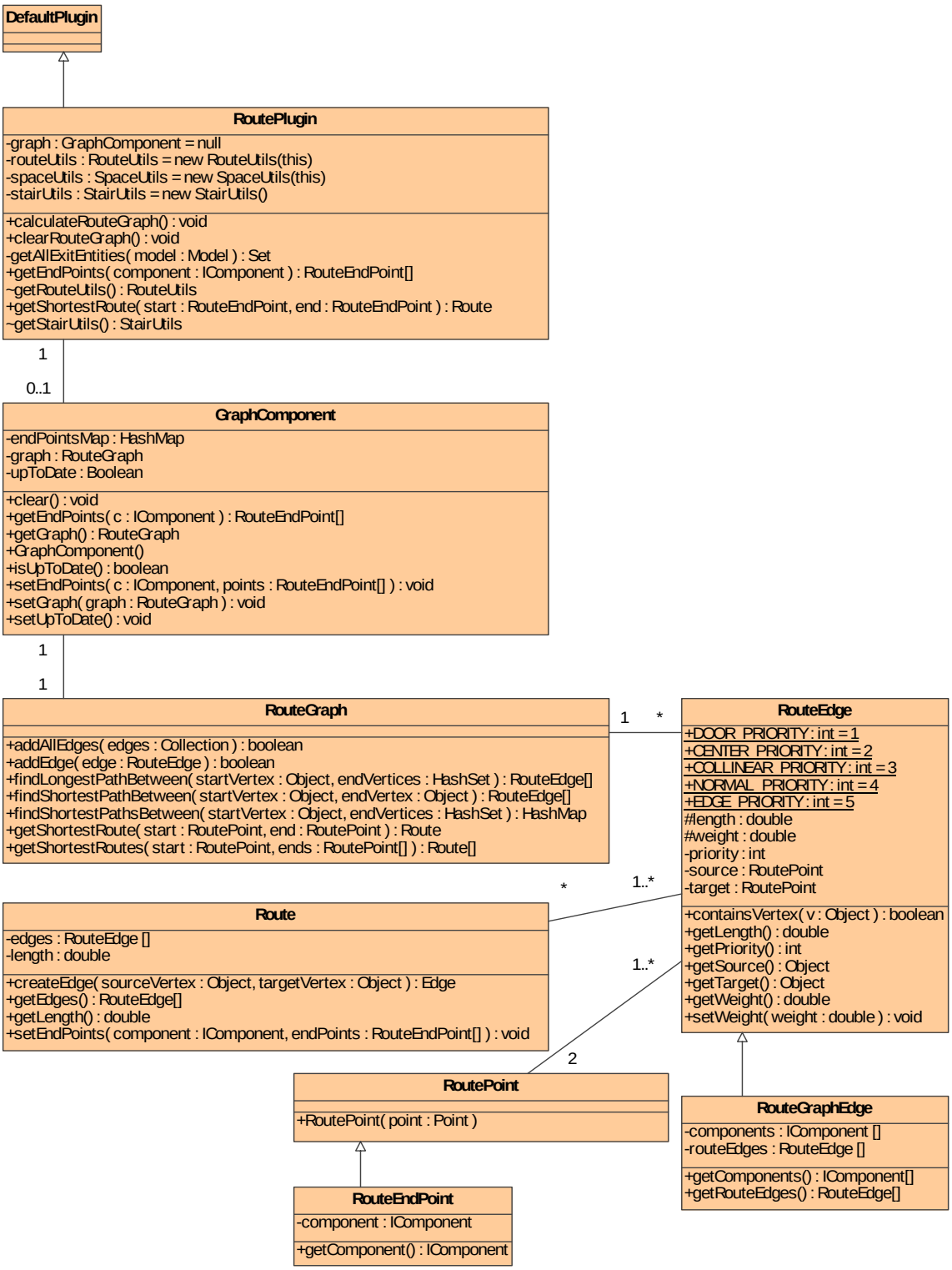

Diagram 3 Class diagram of the route plug-in

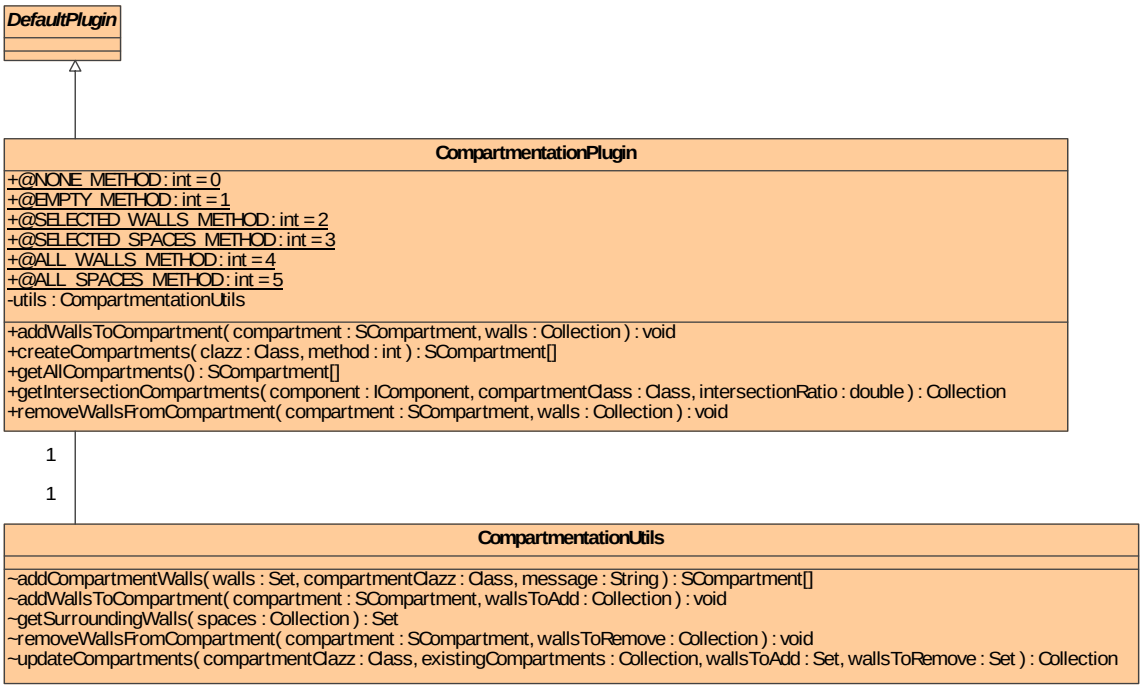

Diagram 4 Class diagram of the compartmentation plug-in