Instytut Podstawowych Problemów Techniki
Polskiej Akademii Nauk

Projektowanie koncepcyjne wspomagane komputerowo z użyciem grafowych struktur danych i formuł logicznych

mgr Szymon Gajek

Rozprawa doktorska napisana pod kierunkiem
Prof. dr hab. Ewy Grabskiej

Kraków 2016

Streszczenie

Faza szczegółowa projektowania architektonicznego jest obecnie bardzo dobrze wspierana przez narzędzia komputerowe. W ostatnich latach otwarła się możliwość prowadzenia badań nad automatyzacją procesu sprawdzania wymagań projektowych. Wymagania stawiane współcześnie przed projektami należą do takich kategorii jak normy budowlane, wymogi bezpieczeństwa, prawo pracy, ograniczenia ekonomiczne, czy ekologiczne. Istotne są również ustalone praktyki projektowania oraz wymagania zleceniodawcy stawiane przed konkretnym projektem. Proces ręcznego sprawdzania tych wymagań, pochodzących z wielu różnych dziedzin, jest narażony na błędy, a jego automatyzacja ma znaczny potencjał ograniczenia kosztów jak i czasu trwania projektów.

Cześć wymagań projektowych może być zweryfikowana już na etapie fazy koncepcyjnej projektowania. Współczesne narzędzia wspomagające projektowanie nie są jednak przystosowane do wspierania fazy koncepcyjnej, nie istnieją również narzędzia dedykowane dla tej fazy, a architekci nie posługują się w tej fazie systemami CAD tylko tradycyjnymi szkicami.

W ramach niniejszej pracy zostało opracowanie i zaimplementowane prototypowe narzędzie wspomagania fazy koncepcyjnej projektowania architektonicznego umożliwiające automatyczne pozyskiwanie wiedzy projektowej oraz sprawdzanie zgodności projektu z wymaganiami. Projektant porozumiewa się z systemem za pomocą języka wizualnego diagramów stosowanego w architekturze. Wewnętrzna reprezentacja projektu jest definiowana w postaci hierarchicznego atrybutowanego hipergrafu.

Proponowane narzędzie jest systemem opartym na wiedzy. Rolę bazy wiedzy pełnią testy zgodności, zapisane w ogólnym, otwartym języku testów bazującym na logice pierwszego rzędu. Język ten został zastosowany do zapisu szeregu wymagań projektowych, w tym złożonych norm polskiego prawa budowlanego dotyczących ograniczeń przeciwpożarowych. Częścią systemu jest ewaluator formuł logiki pierwszego rzędu posiadający parser języka testów. Dane projektu, zawarte w hipergrafie hierarchicznym, są na potrzeby wnioskowania przekazywane do struktury relacyjnej. Struktura ta umożliwia wartościowanie formuł języka testów w kontekście konkretnego projektu. Moduły walidacji projektu z założenia działają w tle, ostrzegając tylko projektanta, jeśli projekt jest niezgodny z aktualnie wybranymi spośród dostępnych zestawów testów ograniczeniami.

Ponieważ żaden z opisanych w literaturze systemów nie podejmuje problemu analizy diagramów tworzonych w fazie koncepcyjnej projektowania architektonicznego, niniejsze badania można postrzegać jako nowatorską próbę zastosowania języków wizualnych do efektywnego pozyskania wiedzy o projekcie oraz wnioskowania o nim za pomocą formuł języka logiki pierwszego rzędu.

Abstract

Nowadays, detailed design phase of the architectural design process is well supported by computer tools. In recent years, a new research area in automated design constraints checking has emerged. Contemporary designs have to meet criteria such as building code, safety requirements, labor code, economical or ecological constraints. Best practices of design and client requirements for particular project also play an important role. Manual project review, especially against criteria ranging from different domains, is error prone. Automation of this process has the potential to save both time and cost of a whole project.

Some of requirements may be already verified during conceptual design phase. However, contemporary CAD tools still lack satisfactory support for conceptual design and there are no tools strictly dedicated to this phase. Architects still prefer hand-drawn sketches over CAD tools.

This thesis presents the idea and implementation of a prototype computer tool supporting conceptual design that automatically extracts design knowledge and performs conformity checks. The visual language build of diagrams popular among architects is used as system input. The attributed hierarchical hypergraphs are used as internal project representation.

The proposed tool is a knowledge-based system. Compliance tests, formulated in an abstract, open, based on first-order logic language act as a knowledge base. A number of design requirements have been encoded in this language, including complex fire safety constraints from the Polish building code. A first-order logic formulae evaluator containing parser of the tests language is part of the system. Project data, stored in the hierarchical hypergraph are passed to the relational structure for purposes of reasoning. This structure enables evaluation of tests language formulae in context of a particular design. System's validation modules are designed to operate in background and warn a designer only if a design does not conform to criteria chosen from available tests sets.

As none of the systems described in literature deals with analysis of diagrams of preliminary phase of architectural design, this research can be seen as an innovatory attempt of applying visual languages to effective design knowledge extraction and reasoning with use of language of first-order logic formulae.

Spis treści

1. Wstęp

1.1. Tematyka badawcza i teza pracy

Współczesne narzędzia komputerowego wspomagania projektowania (ang. Computer-Aided Design, CAD) weszły w fazę rozwoju daleko wykraczającą poza komputerową wersję deski kreślarskiej. Projektowanie szczegółowe jest obecnie dobrze wspomagane systemami komputerowymi, które w niektórych obszarach całkowicie wyparły tradycyjne techniki [1]. Klasyczne systemy CAD, przechowujące projekty jako zestaw dwuwymiarowych rzutów, są wypierane przez narzędzia oparte na paradygmacie modelowania informacji o budynku(ang. Building Information Modelling, BIM), które używają bazy danych komponentów 3D jako głównego sposobu zapisu projektów.

W fazie wstępnej, zwanej również fazą koncepcyjną, większość architektów nie posługuje się systemami CAD. Jej produktami są raczej tradycyjne szkice wykonywane bez użycia komputera [1] [2] [3] [4] [5]. W dedykowanych programach powstają jedynie wizualizacje wstępnych projektów na potrzeby prezentacji dla klientów. Narzędzia CAD nie są bowiem przystosowane do wspierania fazy koncepcyjnej [3]. Paradygmat BIM również nie znalazł zastosowania w tej fazie [6].Okazuje się, że złożoność narzędzi dedykowanych dla fazy szczegółowej jest przeszkodą w kreatywnej pracy nad nowymi ideami i koncepcjami. Rezultatem tego jest poświęcanie zbyt dużo czasu interakcji z oprogramowaniem w stosunku do czasu poświęconego na projektowanie samo w sobie [2]. Tymczasem faza koncepcyjna jest kluczowa dla całego procesu projektowania [7] [8], a intensywne szkicowanie w tej fazie idzie w parze z lepszymi rezultatami projektu [9]. Znamienny jest fakt, że błędy popełniane w tej fazie są szczególnie kosztowne, ponieważ zostają powielane w kolejnych fazach [3].

Coraz większego znaczenia nabiera zgodność projektów z wymaganiami. Standardy oraz przepisy budowlane są definiowane prze wiele niezależnych instytucji z różnych perspektyw: od norm budowlanych po prawo pracy, a także na różnych poziomach: od ponadnarodowych poprzez branżowe, po wymagania klienta określone dla konkretnego projektu. Mnogość tych perspektyw i poziomów znacznie komplikuje proces walidacji projektu. Proces ten wymaga nieraz analizy złożonych teksów technicznych lub prawnych, połączonych dodatkowo siecią zależności i poddawanych późniejszym poprawkom. Analiza niektórych z nich pozostawia pewną swobodę interpretacji podczas stosowania. Wiele ograniczeń to także niesformalizowana wiedza ekspercka, nazywana wiedzą ukrytą (ang. tacit knowledge), która obejmuje zasady projektowania, pewne powszechnie znane dobre praktyki, a także sam proces niesformalizowanego sprawdzania projektu [10].

Do niedawna jedynym sposobem radzenia sobie z mnogością wymagań projektowych było poleganie na wiedzy architektów oraz wewnętrzny proces sprawdzania i recenzji projektów w biurach projektowych [11]. W ostatnim czasie, pojawiają się jednak systemy

CAD wspierające podejmowanie decyzji. Należą do nich systemy CAD z bazą wiedzy. Ich zadaniem jest zaawansowane wspieranie projektanta na podstawie technik inżynierii wiedzy. Stanowią one istotną kategorię systemów z bazą wiedzy. Systemy z bazą wiedzy są złożone. Do ich działania konieczna jest formalizacja specjalistycznej wiedzy z pewnej dziedziny oraz zapis jej w formie mogącej służyć do automatycznego przetwarzania. To zadanie wymaga zaangażowania inżynierów wiedzy (ang. knowledge engineer).

Pomimo że projektowanie oparte na wiedzy (ang. knowledge-based engineering) jest jedna z pożądanych cech systemów wspierających projektowanie[27], systemy tego rodzaju nie są szeroko stosowane w przemyśle.

W dziedzinie zgodności projektów daje się zauważyć rozwój badań związany z wprowadzeniem BIM. Pomimo tego, jedynym systemem sprawdzającym projekty oraz mogącym być użytym w przemyśle jest system CORENET, opracowany w Singapurze na potrzeby weryfikacji tamtejszych norm budowlanych. Według autorów przeglądu systemów sprawdzających projekty architektoniczne [11], kompleksowe sprawdzanie ograniczeń projektowych jest dużym wyzwaniem dla branży, a jego realizacja zajmie wiele lat. Dotychczasowe próby podejmowane w tej dziedzinie, realizowały weryfikacje projektów wyłącznie pod kątem warunków hermetycznie zaszytych w kodzie programów, z ewentualną z możliwością parametryzacji. Tymczasem istnieje potrzeba stworzenia otwartego języka wymagań projektowych, możliwie przystępnego oraz o sile wyrazu mogącej objąć jak największą liczbę wymagań.

Pomimo rezygnacji z wysokiego poziomu szczegółowości w fazie wstępnej, wiele wymagań projektowych może być brane pod uwagę już na tym etapie [7]. Narzędzia CAD mogłyby pełnić rolę asystentów projektanta przypominających mu o wymaganiach oraz ograniczeniach projektowych, a nawet sugerować pewne rozwiązania. W przeciwieństwie do analizy projektów post factum, taka analiza on-line pozwala na natychmiastową identyfikację problemu, oraz uwzględnienie modyfikacji na dalszym etapie procesu projektowego. Formalizacja wiedzy projektowej stwarzająca możliwość wielokrotnego jej wykorzystywanie jest wartościową techniką w projektowaniu. Tymczasem faza projektowania koncepcyjnego ciągle nie doczekała się efektywnych narzędzi ją wspierających, a większość architektów posługuje się w tej fazie odręcznymi szkicami [2]. Żaden z systemów CAD nie oferuje analizy spójności szkiców tworzonych przez architektów [7].

Istnieje wyraźna potrzeba stworzenia systemów CAD za pomocą których można szybko tworzyć, oceniać, modyfikować i wnioskować o poprawności wstępnych rozwiązań projektowych. Skłania to do postawienia następującej tezy:

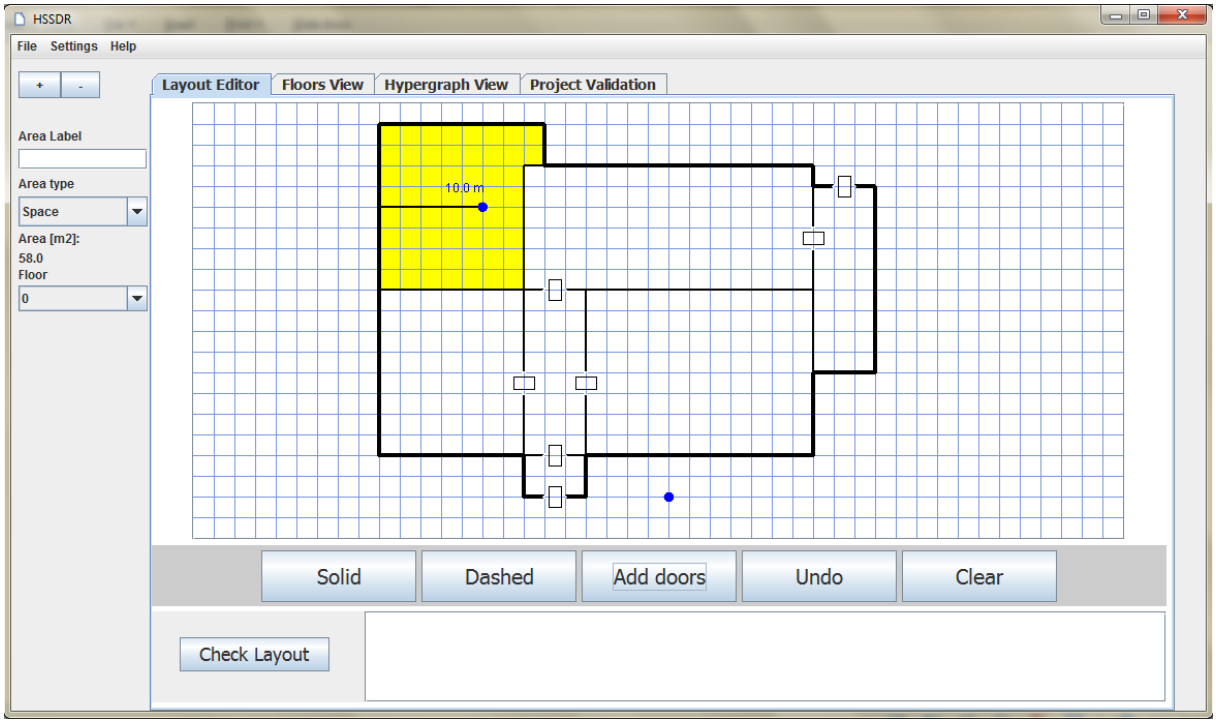
Istnieje możliwość wspomagania projektowania architektonicznego w fazie wstępnej poprzez automatyczne pozyskiwanie wiedzy projektowej oraz sprawdzanie zgodności projektu z wymaganiami.

Celem niniejszej pracy jest opracowanie i implementacja prototypu narzędzia CAD dla fazy wstępnej projektowania architektonicznego, które umożliwi zarówno automatyczne pozyskiwanie wiedzy projektowej, jak i wspomaganie projektowania w fazie wstępnej poprzez sprawdzanie zgodności projektu z wymaganiami.

1.2. System HSSDR

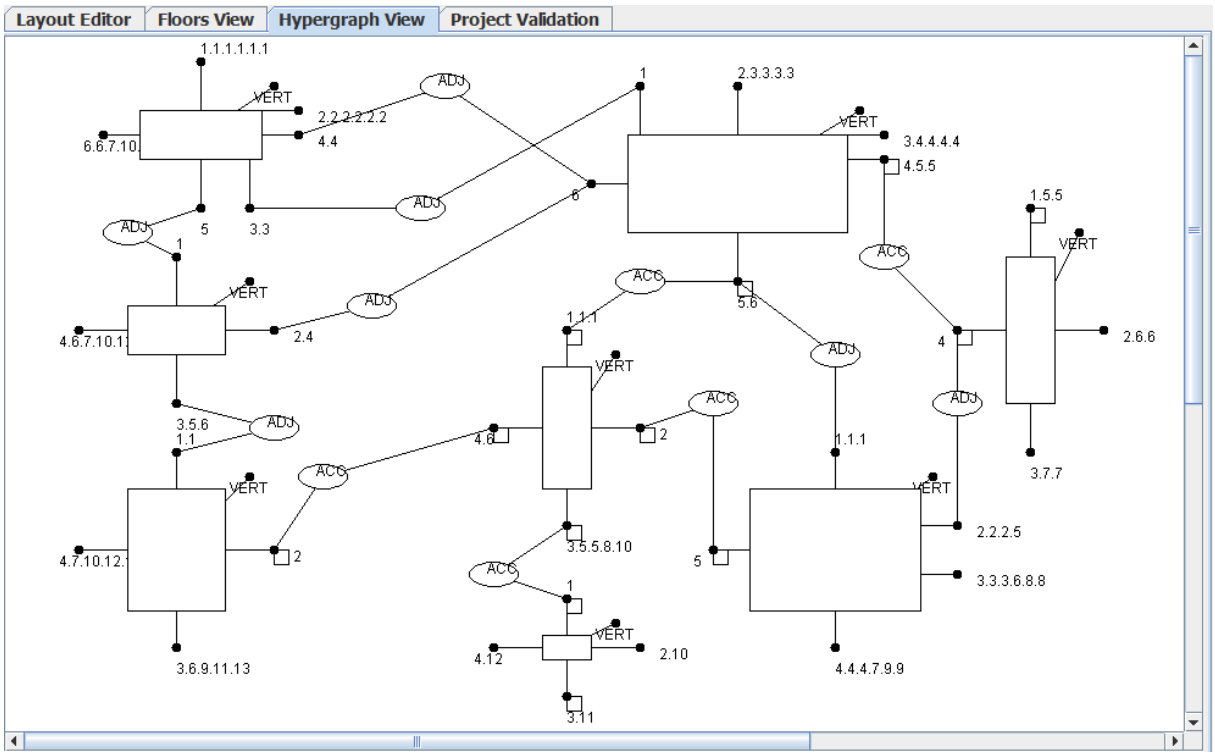
W niniejszej pracy przedstawione jest prototypowe narzędzie wspomagania projektowania architektonicznego stworzone z myślą o projektowaniu koncepcyjnym. Pełna nazwa systemu określanego w dalszej części pracy akronimem HSSDR brzmi: Hipergrafowy System Wspierający Projektowanie i Wnioskowanie (ang. Hypergraph System Supporting Design and Reasoning). Założeniem systemu jest aby dialog między projektantem a systemem komputerowym odbywał się za pomocą języka wizualnego. Diagramy systemu HSSDR są uproszczeniem diagramów powszechnie stosowanych w architekturze. System wymusza na użytkowniku projektowanie metodą od ogółu do szczegółu.

Poniższy zrzut ekranu (Rysunek 1.1) przedstawia główne okno systemu w trakcie procesu projektowania rozkładu pomieszczeń.



Rysunek 1.1 Projektowanie rozkładu pomieszczeń w systemie HSSDR

System HSSDR jest systemem opartym na wiedzy. Rolę bazy wiedzy pełnią testy zgodności, zapisane w języku testów HSSDR, bazującym na logice pierwszego rzędu. Moduły walidacji projektu systemu HSSDR zostały zaprojektowane tak, aby pełniły rolę inteligentnego asystenta projektanta. Z założenia działają one w tle, sugerując tylko projektantowi działania jeśli projekt nie jest zgodnym z aktualnie wybranymi wymaganiami. Cechą charakterystyczną fazy wstępnej jest posługiwanie się uproszczoną reprezentacją projektu w postaci szkiców, które z kolei nie nadają się do przetwarzania komputerowego. W celu wspomagania tej fazy istnieje zatem potrzeba stworzenia wewnętrznej reprezentacji projektu na potrzeby jego analizy. W proponowanym systemie ta reprezentacja oparta jest na strukturach grafowych. Konkretnie do zapisu projektu użyty jest *hierarchiczny atrybutowany hipergraf*, w skrócie hipergraf (określenia hipergraf planu, hipergraf, graf będą używane zamiennie). Struktura ta umożliwia zapis informacji o składowych projektu, relacjach przestrzennych między składowymi oraz o ich właściwościach. Struktura hipergrafowa jest automatycznie uaktualniana przez algorytmy systemu po każdej akcji projektanta. Na rysunku Rysunek 1.2 przedstawiony jest hipergraf wygenerowany przez HSSDR dla rozkładu pomieszczeń przedstawionego na rysunku Rysunek 1.1. Widok ten, dostępny na jednej z zakładek okna programu, został stworzony na potrzeby prezentacji oraz monitorowania programu, nie ma on służyć jako część procesu projektowania koncepcyjnego.

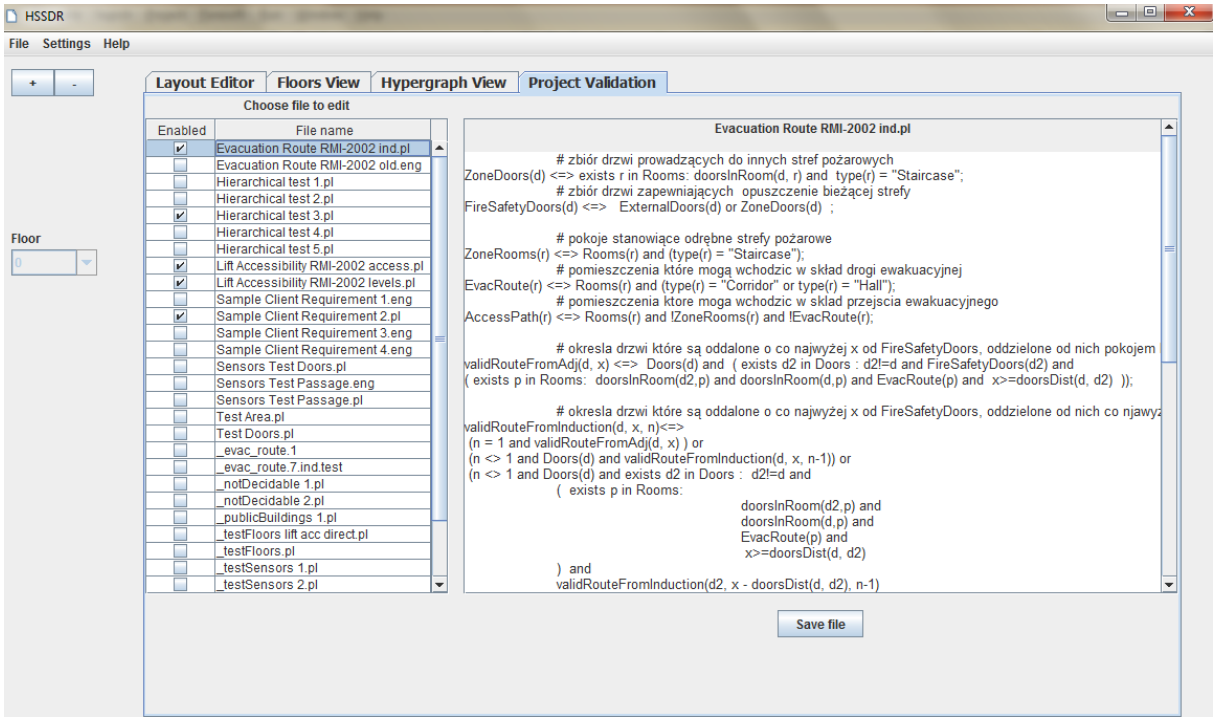


Rysunek 1.2 Fragment hipergrafu wygenerowany przez HSSDR dla projektu z rysunku Rysunek 1.1

Informacje o topologii projektu oraz o właściwościach jego składowych zapisane w strukturze hipergrafowej są dalej udostępniane na potrzeby automatycznego wnioskowania

o projekcie. Zasilana jest nimi *struktura relacyjna hipergrafu*, która z kolei umożliwia wartościowanie formuł *języka testów HSSDR*.

Język testów HSSDR jest ogólnym, otwartym językiem wyższego poziomu pozwalającym formułować tezy o projekcie. Jego formuły odnoszą się do obiektów istniejących w analizowanym projekcie, a więc do drzwi, pomieszczeń, a także do relacji i właściwości charakteryzujących obiekty, zarówno bezpośrednich jak i obliczonych przez system. Za pomocą tego języka można tworzyć *zestawy testów zgodności*. Zawarta w nich wiedza projektowa z wiedzy ukrytej, staje się sformalizowaną wiedzą wyrażoną explicite w postaci reguł. Na jednej z zakładek, użytkownicy systemu wybierają które spośród plików zawierających testy mają być wykonywane dla bieżącego projektu (Rysunek 1.3).



Rysunek 1.3 Edycja oraz wybór zestawów testów zgodności

Wewnętrzna reprezentacja projektu w postaci hipergrafów hierarchicznych jest aktualizowana automatycznie po każdej zmianie projektu. Rezultaty testów, wykonywanych w tle, są prezentowane projektantowi. Dokładny opis sposobu interakcji użytkownika z systemem znajduje się w Dodatku. W przypadku wykrycia naruszenia warunków zapisanych w testach, użytkownik jest o tym powiadamiany poprzez informacje tekstową w dolnej konsoli lub poprzez wyróżnienie kolorem niespełniających wymagań fragmentów projektu w głównym oknie programu.

1.3. Zawartość rozprawy

Niniejsza rozprawa składa się z pięciu rozdziałów, bibliografii i jednego dodatku.

W rozdziale drugim przedstawione są aktualne zagadnienia związane z projektowaniem architektonicznym wspieranym komputerowo, omówiony jest proces projektowania ze szczególnym uwzględnieniem fazy koncepcyjnej (podrozdział 2.1) oraz jego komputerowe wsparcie przez współczesne systemy CAD, z uwzględnieniem paradygmatu BIM (podrozdział 2.2). Następnie omówiona została problematyka zgodności projektów z wymaganiami (podrozdział 2.3). W podrozdziale 2.4 przedstawione są zagadnienia związane z systemami z bazą wiedzy. Podrozdział 2.5 zawiera przegląd literatury związanej ze wspomaganiem projektowania.

Rozdział trzeci przedstawia szczegółowo stworzony w ramach niniejszej pracy system wspomagania projektowania oraz powiązane zagadnienia teoretyczne. Podrozdział 3.1 przedstawia założenia systemu HSSDR oraz jego architekturę. Podrozdział 3.2 opisuje diagramy projektowe. Podrozdział 3.3 wprowadza struktury hipergrafowe używane do zapisu projektu. W podrozdziale 3.4 opisany jest algorytm automatycznego tworzenia hipergrafu na podstawie diagramów projektowych. Podrozdział 3.5 wprowadza język logiki pierwszego rzędu oraz język testów HSSDR. Podrozdział 3.6 definiuje pojęcie struktury relacyjnej oraz opisuje proces walidacji projektu. W rozdziale 3.7 przedstawione są wybrane testy zgodności zapisane w języku testów HSSDR.

Rozdział czwarty przedstawia rozszerzenie proponowanego systemu wprowadzające możliwość projektowania oraz sprawdzania reguł dla budynków wielopiętrowych. Zawiera on również przykłady ograniczeń projektowych dotyczące budynków wielopiętrowych.

Rozdział piąty omawia koncepcje udostępnienia mechanizmów walidacji projektu HSSDR dla innych narzędzi CAD. W tym podejściu biblioteka testów przechowywana jest na serwerze, a usługa sprawdzania ograniczeń projektowych jest udostępniana przez sieć wielu klientom.

Ostatni, szósty rozdział, zawiera podsumowanie oraz wnioski wyciągnięte z przeprowadzonych prac.

2. Projektowanie architektoniczne wspomagane komputerowo

2.1. Proces projektowania

Projektowanie architektoniczne, ze względu na złożoność końcowych produktów oraz liczbę różnych dziedzin, które musi objąć, jest złożonym procesem. Architekci muszą z jednej strony być kreatywni, ponieważ od projektów na ogół oczekuje się innowacyjności i spełniania wymagań estetyki. Z drugiej strony, muszą oni brać pod uwagę szereg wymagań technicznych, prawnych i ekonomicznych [3]. Istnieją różne fazy projektowania, w których na różnych poziomach abstrakcji podejmowane są różne aspekty problemu. Ze względu na poziom abstrakcji można wyróżnić dwie fazy główne projektowania: projektowanie koncepcyjne (ang. conceptual design) oraz projektowanie szczegółowe (ang. detailed design).

Krajowa Rady Izby Architektów w uchwale „Standardy wykonywania zawodu i zakres usług architekta" [12] wyróżnia następujące fazy dokumentacji projektowej:

- projekt koncepcyjny, którego celem jest "Określenie wstępnych, architektonicznych i technicznych zasad rozwiązania i standardów, które winny być podstawą dalszych prac projektowych." W jego skład mogą wchodzić również wizualizacje, modele 3D czy makiety.
- projekt budowlany, którego celem jest "Ostateczne i jednoznaczne określenie wymaganych prawem rozwiązań i tych, które będąc przedmiotem zatwierdzenia przez władze architektoniczno – budowlane, będą musiały być zrealizowane." Służy on do uzyskania pozwolenia na budowę. Jest jedynym etapem, którego zakres regulowany jest w Polsce przez ustawę [13].
- projekty realizacyjne, będące podstawą prowadzenia robót (są to projekty trafiające do rąk wykonawców):
    - przetargowy, na potrzeby przetargu,
    - projekt kontraktowy, stanowiący podstawę do podpisania kontraktu z wykonawcą,
    - projekt wykonawczy, służący do bezpośredniej realizacji prac budowlanych, zawiera szczegóły niezbędne do ich realizacji. W jego skład mogą wchodzić projekty branżowe, takie jak konstrukcyjny, sanitarny, elektryczny.

2.1.1. Faza wstępna

W fazie wstępnej projektowania, zwanej również fazą koncepcyjną powstają uproszczone projekty nieprzykładające wielkiej wagi do szczegółów, które z kolei zostają wprowadzone w fazie szczegółowej. Jednym z zadań w fazie koncepcyjnej jest wzięcie pod uwagę różnych aspektów rozważanego projektu:

- wiedzy charakterystycznej dla konkretnej dziedziny jak standardy ekologiczne, ograniczenia prawne, wymogi bezpieczeństwa, ograniczenia ekonomiczne,
- ustalonych praktyk projektowania lub powszechnie stosowanych wzorców wynikających z doświadczeń,
- specyficznych wymagań konkretnego projektu, wymagań zleceniodawcy itp. [14]

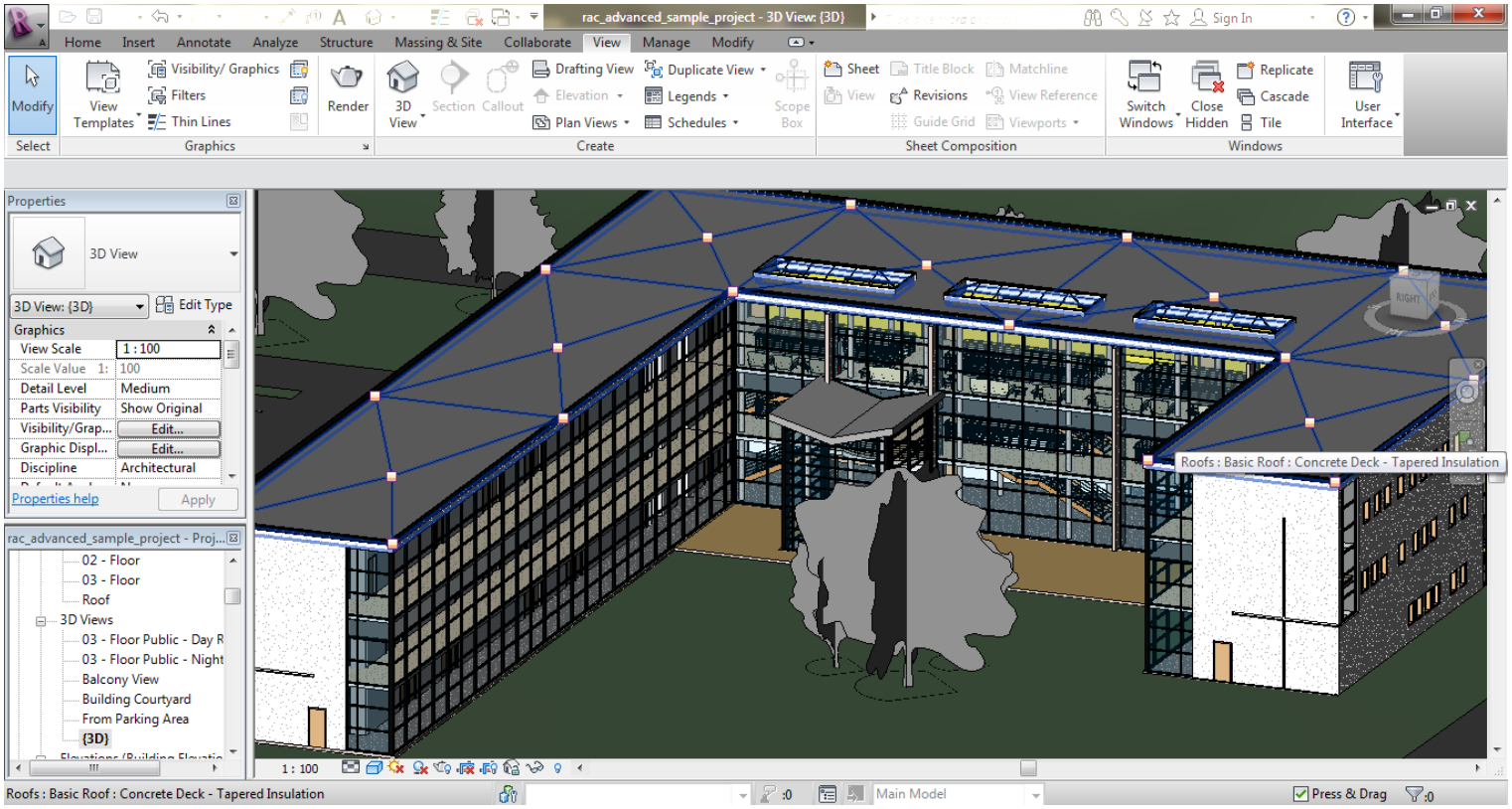
Faza wstępną jest fundamentalna z perspektywy całego procesu projektowania i powstawania budynku [7] [8]. Brak odpowiednich informacji prowadzi do błędów, a błędy powstałe w tej fazie są powielane w kolejnych etapach. Z uwagi na to, że błędy popełnione na wczesnym etapie projektu są trudne i kosztowne w naprawie, poprawność fazy koncepcyjnej jest kluczowa dla powodzenia całego projektu [3]. Efektem końcowym tej fazy nie są szczegółowe plany zawierające dokładne wymiary i szczegóły konstrukcyjne, użyte materiały i komponenty, ale szkice. To właśnie szkice, wspierając myślenie wizualne stanowią znaczną pomoc podczas projektowania [15]. Projektowanie w tej fazie koncentruję się na budynku jako całości składającej się mniejszych komponentów połączonych pewnymi relacjami i wchodzącymi ze sobą w określone zależności [7]. Taki poziom abstrakcji pozwala skoncentrować się na użyteczności budynku i podporządkować całą jego konfigurację funkcjom, jakie ma pełnić budynek i składające się na niego pomieszczenia. Pomimo rezygnacji z wysokiego poziomu szczegółowości, wiele wymagań projektowych może być brane pod uwagę już na tym etapie, między innymi ograniczenia prawne, techniczne, funkcjonalne, finansowe [7].

2.2. Komputerowe wspomaganie projektowania architektonicznego

Początki komputerowego wspierania projektowania architektonicznego sięgają lat 60-tych XX wieku. Klasyczne systemy CAAD (and. Computer-Aided Architectural Design), były implementowane, jako komputerowe wersje deski kreślarskiej. Przechowywały one projekty jako zestaw dwuwymiarowych rzutów, takich jak schematy, widoki pięter i elewacji. W ostatnim czasie dwuwymiarowe systemy CAD-owskie są wypierane przez narzędzia oparte na paradygmacie BIM, które używają bazy danych komponentów 3D jako głównego sposobu zapisu projektów. Elementami składowymi projektu nie są więc dwuwymiarowe linie czy łuki, ale obiekty takie jak ściany, belki, stropy, drzwi, które posiadają informacje o swojej

nazwie, lokalizacji, materiale, przeznaczeniu itp. W tym modelu dwuwymiarowe rzuty z dowolnej perspektywy, jak również sceny 3D, są generowane na podstawie bazy danych BIM. Inaczej niż w przypadku systemów CAD 2D, gdzie zmiana jednego widoku wymaga aby wszystkie inne widoki zostały sprawdzone i uaktualnione, co jest procesem podatnym na błędy [16].

Przykładem systemu opartego na BIM jest Revit Architecture [17]. Jest to nowoczesne, rozbudowane narzędzie CAD-owskie obejmujące całościowo projektowanie architektoniczne, a także projektowanie instalacji oraz elementów konstrukcyjnych budynku (Rysunek 2.1).



Rysunek 2.1 Projektowanie w Revit Architecture

Wynikiem wprowadzenia BIM jest znaczna różnica jakościowa, stwarzająca nieistniejące wcześniej i trudne nawet do przewidzenia możliwości systemów CAD. Przykładem może być sposób projektowania Projektuj-Graj (ang. Design-Play), opisany w pracy „Integrating BIM and gaming for real-time interactive architectural visualization” [18]. Autorzy zbudowali system generujący świat gry na podstawie danych BIM projektu tworzonego za pomocą środowiska Revit Architecture. Projektant może w pewnym momencie przerwać projektowanie i przenieść się do powstającego projektu i ocenić go na podstawie łatwości poruszania się i wykonywania zadań, wcielając się w użytkownika, bądź obsługę techniczną budynku.

Komputerowe wspomaganie projektowania oraz BIM umożliwiają zastosowanie na szeroką skalę nie tylko modeli 3D ale także 4D. Dodanie czasu jako czwartego wymiaru umożliwia zamodelowanie kolejnych etapów konstrukcji budynku. To z kolei umożliwia zaawansowane wspomaganie przebiegu budowy już na etapie projektowania. Ostatnie badania „Building Information Modeling (BIM) and Safety: Automatic Safety Checking of Construction Models and Schedules" [19] pokazują możliwości sprawdzania procedur bezpieczeństwa na budowie oraz automatycznego generowania zabezpieczeń takich jak balustrady i bariery ochronne w poszczególnych fazach konstrukcji. Z kolei inne niedawne prace [20] pokazują możliwość zaawansowanego wyznaczania najkrótszych ścieżek w budynku z uwzględnieniem informacji semantycznych o obiektach.

Kluczową role w wymianie informacji pomiędzy różnymi aplikacjami BIM odgrywa standard IFC (Industry Foundation Classes), rozwijany przez buildingSMART, znane wcześniej jako International Alliance for Interoperability (IAI). IFC zapewnia efektywny i powszechnie zaakceptowany protokół cyfrowej reprezentacji budynków. Jest to otwarty, uniwersalny format, będący poza kontrolą pojedynczego komercyjnego podmiotu, wspierany przez większość narzędzi CAD [20]. IFC zawiera zarówno informacje geometryczne jak i semantyczne o elementach budynku. Format ten będzie w przyszłości dalej rozwijany i używany jako niezależne narzędzie wymiany informacji, obok formatów specyficznych dla konkretnych aplikacji jak DWG czy PLN (ArchiCAD) [6].

Istnienie uniwersalnego formatu zapisu projektów umożliwiło pojawienie się narzędzi wyższego poziomu, takich jak platforma Solibri Model Checker (SMC) [21]. Jest to aplikacja desktopowa, która wczytuje model IFC, a następnie tworzy na jego podstawie wewnętrzną reprezentację oraz udostępnia możliwość jego analizy poprzez szereg udostępnionych funkcji. Niestety jest to platforma komercyjna, która nie udostępnia publicznego interfejsu programistycznego (API), ograniczając tym samym możliwości systemu do funkcji dostarczonych przez Solibri. Natomiast powstanie tego rodzaju narzędzia opartego o otwarty standard umożliwiłoby powstanie szeregu bibliotek do analizy projektów IFC w kontekście wielu różnych zastosowań.

Pomimo że BIM jest odpowiedzią na problemy wywodzące się z wczesnych faz projektowania, nie znalazł on jak na razie zastosowania w fazie wstępnej [6]. Wyzwaniem dla BIM jest również opracowanie jednolitych standardów jego implementacji i zdefiniowanie procesów projektowanie z jego użyciem [22].

W Polsce, w ostatnich latach, BIM zaczyna odgrywać istotną rolę, zwłaszcza w większych biurach projektowych. Według badania przeprowadzonego w 2015 roku na zlecenie Autodesk [23] 25,4% specjalistów z branży architektoniczno-budowlanej brało udział w pracach nad projektem z wykorzystaniem BIM. Świadomość BIM, szacowana na 45%, jest najwyższa wśród architektów (65,4%) oraz osób ze stażem pracy do 10 lat. BIM w Polsce jest słabiej rozpowszechniony niż w Europie Zachodniej czy Stanach Zjednoczonych. Dla porównania, według badania z 2010, użycie BIM już wtedy wynosiło w Wielkiej Brytanii, Francji i Niemczech odpowiednio 35%, 38%, 36% [24]. Rząd Wielkiej Brytanii planuje nawet

wymagać stosowania BIM we wszystkich projektach realizowanych na zlecenie rządu do końca 2016 roku [25].

Specjalną kategorią systemów CAD są systemy CAD z bazą wiedzy opisane w podrozdziale 2.4. Ich zadaniem jest zaawansowane wspieranie projektanta w oparciu o techniki inżynierii wiedzy. Mogą one:

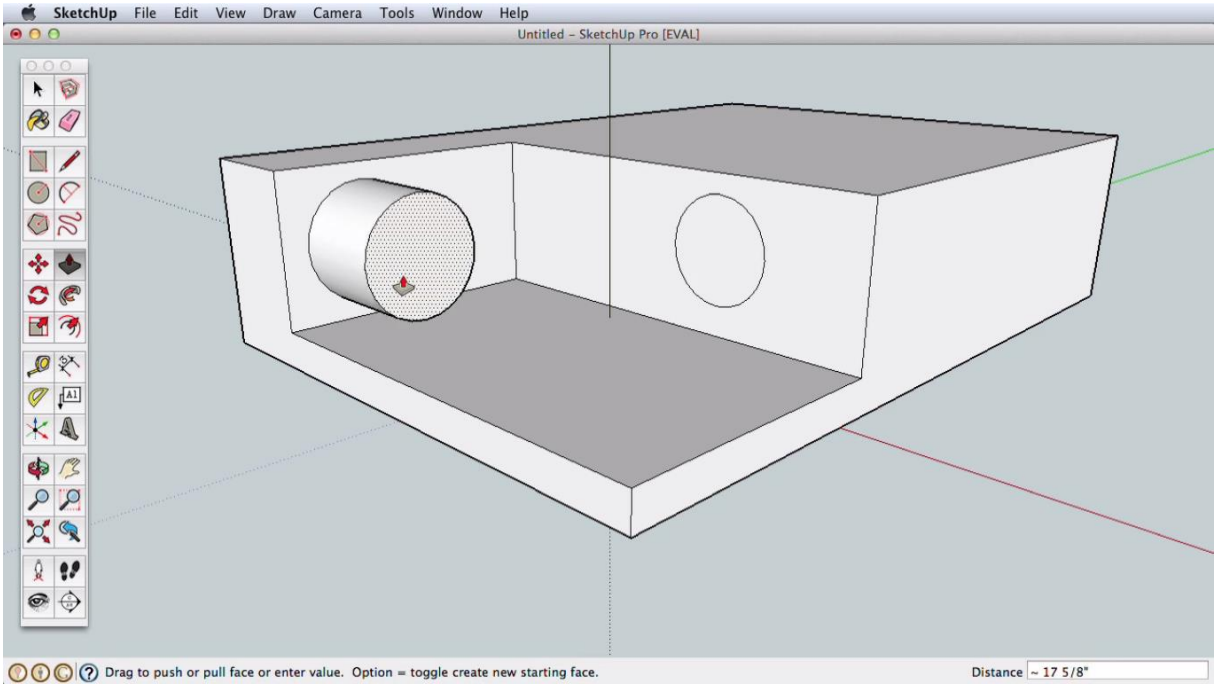
- przeszukiwać zbiór istniejących rozwiązań, aby proponować rozwiązanie istniejącego problemu,
- generować propozycje rozwiązań projektowych,
- optymalizować projekty,
- sprawdzać wymagania i standardy,
- dokonywać złożonych obliczeń i symulacji na podstawie projektu.

Systemu tego rodzaju ograniczają się raczej do niekomercyjnych prób badawczych i są rzadko stosowane na większą skalę [1].

2.2.1. Wsparcie fazy wstępnej

Według wielu ekspertów współczesne narzędzia wspomagające projektowanie nie są przystosowane do wspierania fazy koncepcyjnej. Dla tej fazy nie stworzono dedykowanych narzędzi [3] [7]. Niedawne badania [9] pokazują, że intensywne szkicowanie w fazie wstępnej idzie w parze z lepszymi rezultatami projektu. Obecnie większość architektów nie posługuje się w fazie wstępnej systemami CAD, jej produktami są raczej tradycyjne szkice wykonywane bez użycia komputera [2] [3] [4].

W ostatnich latach pojawił się program CAD firmy Google - SketchUp [26]. W ocenie autora jest to narzędzie, które umożliwia projektowanie architektoniczne w fazie koncepcyjnej. Pozwala ono na tworzenie w łatwy i intuicyjny sposób trójwymiarowych modeli dowolnych obiektów. Dowolny kształt 2D może być transformowany do kształtu 3D (Rysunek 2.2). Istotna jest łatwość i szybkość modelowania, która jest jedną z cech charakterystycznych dla szkicowania. SketchUp kładzie duży nacisk na kształt form i wizualizację. Nie jest natomiast możliwe wprowadzenie przeznaczenia elementów projektu, ani tym bardziej sprawdzanie spójności stworzonego obiektu czy spełniania ograniczeń projektowych.



Rysunek 2.2 Przykład modelowania w SketchUp [26]

Interesujące badania na temat szkicowania w projektowaniu architektonicznym z użyciem SketchUp zostały przedstawione w pracy "Contemporary Digital Techniques in the Early Stages of Design" [5]. Autorzy tłumaczą brak zastosowania komputerów podczas szkicowania w następujący sposób:

„Typowym narzędziom CAAD ciągle nie udaję się zapewnić odpowiedniego środowiska dla szkiców projektowych przeznaczonego do wykonywania akcji projektowych. W środowisku (architektów) dalej wydaje się istnieć głęboko zakorzeniony pogląd, że szkicowanie z użyciem komputera nie jest możliwe. Ostatnie postępy w dziedzinie oprogramowania i mocy komputerów zaczynają to zmieniać. Wielu, jeśli nie większość studentów architektury, używa komputerów podczas swoich pierwszych prób projektowania”.

W badaniach opisanych w wyżej cytowanej pracy, autorzy zastanawiają się nad możliwościami zastąpienia tradycyjnego szkicowania przez komputery. Przeprowadzili oni ciekawy eksperyment. Studenci, podzieleni na dwuosobowe zespoły, pracowali nad zadaniem stworzenia pewnego projektu architektonicznego. Połowa zespołów mogła używać podczas projektowania tylko tradycyjnych narzędzi, jak szkice oraz kartonowe modele, natomiast pozostała cześć mogła szkicować tylko na komputerze, używając narzędzia SketchUp. Badacze zaobserwowali, że występuje zwiększone zaufanie i otwartość w stosunku do narzędzi cyfrowych, które dostarczają wartość dodaną w stosunku do tradycyjnych szkiców. Stwierdzając oni, że przekonanie podkreślające znaczenie odręcznych szkiców dezaktualizuje się. Zauważają utrudnienia w nauce narzędzia SketchUp, spowodowane przyzwyczajeniem do utartego modelu konwencjonalnych systemów CAAD.

Co ciekawe zaobserwowali oni także różnice w ocenie przydatności pewnych reprezentacji wynikające z płci.

Z kolei badanie na temat BIM przeprowadzone przez Khemlani'ego w 2007 roku, wskazało wparcie dla modelowanie w fazie koncepcyjnej (w oryginale „ability to support preliminary conceptual design modeling"), jako jedną z dziesięciu pożądanych cech systemów BIM [27]. Do niedawna żaden z systemów CAD nie oferował analizy spójności szkiców tworzonych przez architektów. Istnieje, więc potrzeba tworzenia narzędzi CAD nowego rodzaju, abstrahujących od detali konstrukcyjnych [7].

2.3. Problematyka zgodności projektów

Projekty architektoniczne poddawane są coraz większej liczbie ograniczeń z wielu dziedzin, od ograniczeń ekonomicznych po standardy bezpieczeństwa. Wymagania te są definiowane przez ludzi w języku naturalnym, a następnie interpretowane przez ludzi, co rodzi możliwość pomyłek oraz niejednoznacznych interpretacji. Treścią wielu wymagań jest niesformalizowana wiedza ekspercka. Sytuację komplikuje dodatkowo mnogość oddzielnych instytucji definiujących wymagania z różnych perspektyw (od budowlanych po prawo pracy) oraz na różnych poziomach (od ponadnarodowych po branżowe), które pracują niezależnie od siebie nawzajem. Same ograniczenia są często skomplikowanymi technicznymi regułami, zapisanymi w postaci tekstu wzbogaconego o diagramy i tabele. Pozostawiają one również pewną swobodę interpretacji podczas stosowania [10].

Autorzy pracy [10] przeanalizowali francuskie przepisy dotyczące dostępności budynków publicznych. Z pomocą ekspertów, w 9 zbiorach przepisów zidentyfikowane zostało ok. 350 ograniczeń. Ze względu na sprawdzalność w formacie IFC zostały one zaklasyfikowane jako:

1. całkowicie interpretowalne IFC– 5%,
2. przeformułowane w IFC – 20%,
3. częściowo interpretowalne -40%,
4. nieinterpretowalne – 35%.

Autorzy wyżej cytowanej pracy stwierdzają, że „teksty zawierające normy napisane są w sposób nieprzystosowany do przetwarzania komputerowego, a wysiłki zmierzające do przełamania tego ograniczenia nie zakończyły się jak na razie powodzeniem. Co więcej standardy oraz przepisy budowlane często składają się z wielu skomplikowanych dokumentów połączonych siecią zależności i poddawanych późniejszym poprawkom, co utrudnia proces automatyzacji ich sprawdzania" [28].

W ostatnim czasie pojawiają się jednak wysiłki mające na celu skorzystanie z możliwości jakie stwarza digitalizacja i Internet dla udostępniania czy wyszukiwania tekstów prawnych. Przykładem tych wysiłków niech będą: projekt Akoma Ntoso [29] zainicjowany przez ONZ, LegalDocumentXML tworzony przez grupę OASIS [30] oraz MetaLex [31] tworzony przez Europejski Komitet Normalizacyjny. Wysiłki te zmierzają do opracowania i rozpowszechnienia otwartych schematów języka XML, za pomocą których tworzone były by dokumenty prawne. Dokumenty w takich formatach są mieszaniną tekstu oraz metadanych, konkretnie tagów XML, które między innymi opisują strukturę dokumentu, rodzaj aktu prawnego, jego wydawcę oraz datę, określają miejsca definiowania pojęć, pozwalają na cytowania i odniesienia do innych dokumentów.

Jeszcze ambitniejsze cele stawia sobie europejska organizacja European project for Standardized Transparent Representations in order to Extend Legal Accessibility - Estrella. W ramach projektu Legal Knowledge Interchange Format (LKIF) [32] zostały opracowane bazujące na podejściu Semantic Web standardy reprezentacji przepisów prawnych mające umożliwić ich maszynowe przetwarzanie przez systemy z bazą wiedzy. Są to między innymi:

- Terminological Knowledge, warstwa pojęć w języku OWL. Została stworzona zawierająca ponad 200 elementów ontologia LKIF-Core zawierająca podstawowe pojęcia prawne [33],
- Legal Rules, język stworzony na bazie języka SWRL, rozszerzonego dodatkowo, aby siła jego wyrazu objęła reguły prawne w formie przybliżonej (według autorów nawet izomorficznej) do ich oryginalnego zapisu.

Dotychczas jedynym sposobem radzenia sobie z mnogością wymagań było zaufanie wiedzy architektów oraz wewnętrzny proces sprawdzania i recenzji projektów w biurach projektowych [11]. Tylko mechaniczna analiza struktury budynku została skomputeryzowana. W ostatnich latach, wraz z pojawieniem się BIM, stało się możliwe zarówno generowanie projektów odpowiadających wybranym kryteriom jak i automatyczne sprawdzanie projektów po wygenerowaniu [34]. Jako że weryfikacja planów wykonywana przez człowieka staje się często kosztownym wąskim gardłem całego procesu projektowania, automatyzacja tego procesu ma znaczny potencjał ograniczenia zarówno jego kosztów jak i czasu.

Autorzy pracy [11] wyróżniają następujące klasy wymagań projektowych:

- Dla wszystkich budynków: normy architektoniczne na poziomie narodowym (dla Europy także europejskim) i regionalnym,
- Branżowe - dla różnych typów budynków: zasady projektowania dla konkretnego typu budynku ze względu na jego przeznaczenie (normy dla szkół, szpitali, biur, lotnisk, itd.),

- Dla konkretnego projektu: wymagania dotyczące powierzchni, przepustowości korytarzy, specjalnych uwarunkowań terenowych i wszystkie inne wymagania postawione przez zleceniodawcę.

Autorzy przytoczonej pracy stwierdzają również, że dotychczasowe prace w tej dziedzinie koncentrowały się na pierwszej kategorii oraz oczekują, że przyszłe prace obejmą również pozostałe dwie klasy wymagań.

2.4. Systemy z bazą wiedzy oraz reprezentacja wiedzy

Niniejszy rozdział szkicuje aktualny stan wiadomości z zakresu inżynierii wiedzy. Podstawowym zadaniem tej dziedziny jest formalizacja specjalistycznej wiedzy z pewnej dziedziny oraz jej zapis w formie mogącej służyć do automatycznego przetwarzania przez systemy z bazą wiedzy, zwane również systemami ekspertowymi. Jest to istotny kierunek badań współczesnej informatyki, a zarazem interdyscyplinarna dziedzina łącząca wiedzę informatyczną z matematyką, kognitywistką, psychologia oraz konkretnymi dziedzinami, do których należą analizowane problemy. Pomimo że szeroko stosowane w przemyśle systemy nie realizują tego paradygmatu, odpowiednia reprezentacja wiedzy oraz metody pozwalające na jej przetwarzanie są pożądaną cechą systemów wspierających projektowanie [35]. W literaturze opisanych jest wiele prób zastosowania go we wspomaganiu projektowania, które będą rozważane w podrozdziale 2.5.

2.4.1. Reprezentacja wiedzy

Reprezentacja wiedzy (ang. knowledge representation) jest uważana za jedno z ważniejszych zagadnień w dziedzinie sztucznej inteligencji. Dziedzina ta bada formalizmy mogące służyć do zapisu wiedzy explicite. Zajmuje się ona problemem symbolicznego zapisu wiedzy oraz wnioskowania, które może być przeprowadzone przez system komputerowy z użyciem tej wiedzy [36]. Zapis ten powinien być na tyle elastyczny, aby umożliwić odwzorowanie wiedzy z danej dziedziny. Musi się też nadawać do komputerowego przetwarzania w efektywny sposób. Wybór metody reprezentacji wiedzy pod kątem problemów, które będą stawiane przed systemem komputerowym jest kluczowy, ponieważ w znacznym stopniu determinuje efektywność oraz możliwości systemu, a także jego architekturę.

Do najpopularniejszych metod reprezentacji wiedzy należą (za [37]):

- Metody bazujące na zastosowaniu logiki
    - Logika konwencjonalna: rachunek zdań, rachunek predykatów, metoda rezolucji,
    - Logika niekonwencjonalna (rozmyta, wielowartościowa),
    - Metody wykorzystujące zapis stwierdzeń,
    - Metody wykorzystujące systemy regułowe (wektory wiedzy),
- Sieci semantyczne,
- Metody oparte na ramach,
- Metody używające modeli obliczeniowych.
    - Sieci neuronowe
    - Algorytmy genetyczne

Dwie ostatnie metody z powyższej listy, które używają modeli obliczeniowych określane są, jako *niesymboliczne*. Bazują one na modelowaniu istniejących w świecie żywych stworzeń. Wiedza w ich przypadku jest zaszyta pod postacią wag, parametrów czy topologii pewnych modeli obliczeniowych. Pozostałe metody określane są jako *symboliczne*, w ich przypadku wiedza wyrażona jest wprost w różnych możliwych formach. Symboliczne metody zapisu wiedzy można z kolei podzielić na proceduralne oraz deklaratywne.

W proceduralnych - wiedza jest zawarta w procedurach stanowiących przepis na rozwiązanie problemu, natomiast w deklaratywnych wiedza zakodowana jest w zbiorze artefaktów takich jak fakty, stwierdzenia bądź reguły, za pomocą, których może zostać rozwiązany problem.

2.4.2. Systemy z bazą wiedzy

Systemy z bazą wiedzy (ang. knowledge-based systems), nazywane również systemami ekspertowymi, należą do metod i technik sztucznej inteligencji, a także do obszaru inżynierii wiedzy. Są to systemy komputerowe operujące na bazie wiedzy, którą można wyodrębnić z systemu [38]. Baza wiedzy jest przetwarzana przez moduły wnioskujące, w celu rozwiązania problemu postawionego przed systemem.

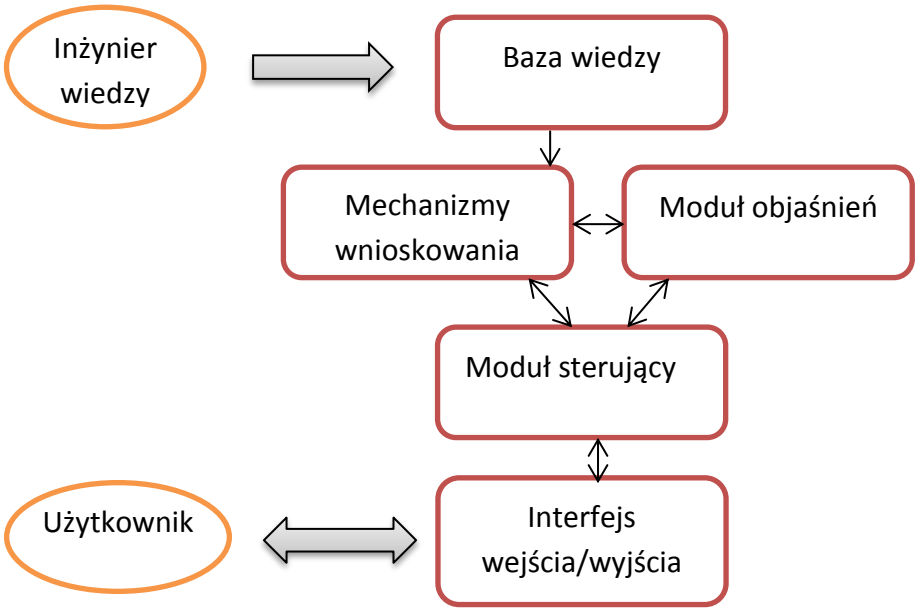
Systemy z bazą wiedzy są złożone. W ich tworzenie są często zaangażowani, oprócz osób tworzących oprogramowanie, również inżynierowie wiedzy (ang. knowledge engineer). Ich rolą jest uzyskanie informacji z dziedziny będącej przedmiotem badań oraz zakodowanie jej w postaci bazy wiedzy używanej przez system. Do tego celu niezbędna jest dobra znajomość analizowanej dziedziny. Koszty tworzenia takich systemów są więc znaczne. Dlatego też są one używane do rozwiązywania trudnych zadań, tradycyjnie rozwiązywanych przez wysoko wykwalifikowanych ekspertów.

Systemy ekspertowe charakteryzują się następującymi cechami (za [1]):

- Wiedza jest odseparowana od logiki aplikacji oraz modułów wnioskujących,
- Wiedza jest zapisana w sposób symboliczny,
- Wiedza jest zapisana w sposób zrozumiały zarówno dla ludzi jak i maszyn. Możliwe jest rozszerzenie bazy wiedzy w efektywny sposób,
- Systemy te zawierają tak zwane heurystyki, które przejmują rolę ludzkich sądów i ekspertyz. Heurystyki są wartościowe, ale ich wyniki są trudne do uzasadnienia i często są wynikiem eksperymentów.

Systemy z bazą wiedzy realizują inny paradygmat niż klasyczne oprogramowanie wytwarzane przez informatyków. Główna różnica polega na operowaniu na *wiedzy*, za pomocą *mechanizmów wnioskowania*, w odróżnieniu od operowania na *danych*, za pomocą *algorytmów*.

Można wyodrębnić wspólną architekturę systemów z bazą wiedzy (Rysunek 2.3):



Rysunek 2.3 Schemat organizacji systemów z bazą wiedzy

W jej skład wchodzą następujące elementy:

- Baza wiedzy – przyjmuję różną formę, w zależności od metody reprezentacji wiedzy przyjętej w systemie,
- Mechanizmy wnioskowania – rozwiązują rozważane w ramach systemu zadania,

- Moduł objaśnień – dostarcza wyjaśnienia wyprodukowanego przez system rozwiązania problemu, w formie mogącej pomóc zrozumieć ścieżkę przeprowadzonego wnioskowania,
- Moduł sterujący – zapewnia wewnętrzną organizacje aplikacji i komunikację pomiędzy poszczególnymi modułami,
- Interfejs wejścia/wyjścia – umożliwia komunikację z systemem.

2.4.3. Metody realizacji systemów z bazą wiedzy

Poniżej scharakteryzowane są po krótce wybrane metody realizacji systemów z bazą wiedzy.

Logika

Logika pierwszego rzędu

Logika pierwszego rzędu jest systemem logicznym, który można traktować jako rozszerzenie rachunku zdań, pozwalające formułować stwierdzenia o zależnościach pomiędzy obiektami indywiduowymi (np. relacjach i funkcjach) [39]. System ten wprowadza kwantyfikatory, które mogą być oparte na zmiennych będących elementami zbiorów (natomiast nie mogą one być oparte na zbiorach). Logika pierwszego rzędu jest zastosowana w niniejszej pracy i jest dokładniej omówiona w podrozdziale 3.5.

Logika opisowa

Logika opisowa (ang. description logic) jest formalizmem stworzonym z myślą o zorientowanych obiektowo metodach reprezentacji wiedzy, takich jak sieci semantyczne. Logika opisowa jest rozstrzygalnym podzbiorem logiki pierwszego rzędu [40], w którym brak jest pojęcia funkcji, a wszystkie relacje są albo unarne, albo binarne. Oferuje ona maksymalną siłę wyrazu przy zachowaniu rozstrzygalności (wszystkie problemy zostaną rozstrzygnięte, a każde wnioskowanie zostanie zakończone w skończonym czasie) [41].

W logice opisowej wyróżnia się terminologię (ozn. TBox) odpowiadającą składni języka, oraz opis świata (ozn. ABox) definiujący uniwersum obiektów będących przedmiotem opisu języka.

TBox

Logika opisowa jest rodziną języków dających różną moc wyrazu, a więc różne możliwości wnioskowania. Podstawowymi elementami terminologii logiki opisowej są:

- koncepty atomowe (ang. atomic concepts), interpretowane jako zbiory obiektów, np.: *Osoba, Mężczyzna, Kobieta*,

- relacje atomowe (ang. atomic roles), interpretowane jako binarne relacje pomiędzy obiektami, np.: *posiadaDziecko, posiadaCórkę, posiadaSyna*,
- koncept uniwersalny(⊤), koncept pusty(⊥).

Koncepty złożone tworzy się za pomocą konstruktorów. Różne języki opisowe powstają przez zmianę zbioru dostępnych konstruktorów. Jako przykład języka niech posłuży język *ALC*, prosty język dający się zastosować w praktyce. Jest on ograniczony do następujących konstruktorów:

- negacja: $\neg \mathrm{C}$,
- koniunkcja: $\mathrm{C} \sqcap \mathrm{D}$,
- suma: $\mathrm{C} \sqcup \mathrm{D}$,
- ograniczenie istnienia: $\exists \mathrm{R}.\, \mathrm{C}$, zbiór obiektów powiązanych co najmniej raz rolą R z obiektem należącym do C,
- ograniczenie wartości: $\forall \mathrm{R}.\, \mathrm{C}$, zbiór obiektów, których wszystkie powiązania rolą R dotyczą obiektów konceptu C.

Do terminologii należą również aksjomaty pozwalające określać zależności pomiędzy konceptami, np.:

$$posiadaDziecko \equiv posiadaCórkę \sqcup posiadaSyna$$

czy

$$\exists posiadaCórkę.\top \sqcap \exists posiadaSyna.\top \sqsubseteq \bot$$

(nie ma osób posiadających jednocześnie córkę oraz syna).

ABox

Opis świata (ABox) składa się z asercji unarnych i binarnych definiujących elementy uniwersum oraz określających rolę pomiędzy parą elementów, np.:

- przynależność do konceptu:

$$x : Mężczyzna$$

$$y : Kobieta$$

- przynależność do roli:

$$< x, y > : posiadaDziecko.$$

Logiczne języki programowania

Programowanie logiczne jest paradygmatem programowania opartym na użyciu logiki. Przykładem języka programowania logicznego jest Prolog [42]. Został on stworzony w latach 70 XX wieku, obecnie posiada wiele różnych implementacji. Programowanie w tym języku polega na wprowadzeniu do systemu pewnej liczby *faktów*, pewnej liczby *zależności* oraz na określeniu *celu*. *Fakty* są pojedynczymi strukturami, które system uznaje za prawdziwe na potrzeby dalszego przetwarzania. *Zależności* opisują reguły, których system może używać do wnioskowania. Natomiast *cel* jest stwierdzeniem, którego prawdziwość chcemy zbadać. Innymi językami programowania logicznego są Datalog, syntaktycznie będący podzbiorem Prologu oraz rodzina języków RuleML [32].

Sieci semantyczne

Sieci semantyczne są strukturami grafowymi służącymi do zapisu wiedzy w postaci wierzchołków i łączących je krawędzi. Sieć semantyczna jest pewnego rodzaju logiką, gdzie relacje między obiektami są zakodowane w krawędziach grafu, a wierzchołki reprezentują same obiekty. Wnioskowanie odpowiada „poruszaniu" się po grafie [37]. Sieci semantyczne są bardzo ogólną i od dawna znaną metodą reprezentacji wiedzy (stosowano je już w starożytnej Grecji). Znajdywały zastosowanie w wielu dziedzinach nauki. Pierwsze komputerowe implementacje powstały na początku lat 60 XX wieku dla celów tłumaczenia maszynowego [43].

Ważnym pojęciem związanym współcześnie z sieciami semantycznymi jest ontologia.

Ontologia jest formalną reprezentacją zbioru pojęć i relacji między nimi, w ramach jakiegoś obszaru wiedzy. Warto zaznaczyć, że omawiane są tutaj ontologie w sensie informatyki, w odróżnieniu od filozofii, gdzie to pojęcie jest używane w innym znaczeniu. Według definicji Grubera jest to „specyfikacja konceptualizacji" (ang. "specification of a conceptualization") [44]. Z kolei John F. Sowa definiuje ontologie w następujący sposób:

„Przedmiotem ontologii jest badanie kategorii bytów mogących istnieć w ramach jakieś dziedziny. Rezultat takich badań, zwany dalej ontologią, jest katalogiem typów obiektów, których istnienie zakłada się w ramach dziedziny D, z perspektywy osoby używającej języka L, w celu opisu D." [45]

Semantic Web

Projekt Semantic Web (czasami tłumaczone na język polski jako Semantyczny Internet) jest przykładem sieci semantycznej, która docelowo ma stanowić nowy standard budowy Internetu. Projekt ten został zainicjowany przez Tima Bernersa-Lee, jednego z twórców podstaw współczesnego Internetu. Pomimo, że Internet zrewolucjonizował współczesny świat, posiada on szereg wad. Jedną z nich jest konstrukcja języka HTML, dobra do prezentowania informacji dla ludzi, ale nieadekwatna dla przetwarzania maszynowego [46]. Sematic Web, jako Internet przyszłości, miałby być zespołem standardów i technologii

umożliwiających programom komputerowym „rozumienie” informacji znajdujących się w sieci oraz wydobywanie nowych treści poprzez wnioskowanie. Z pojęciem Semantic Web związane są następujące technologie:

RDF

RDF (Resource Description Framework) [47] jest językiem opartym na XML, używanym do zapisu danych i metadanych w sieci semantycznej. Opiera się on na identyfikacji poprzez Uniform Resource Identifiers (URIs) oraz na określaniu zasobów za pomocą stwierdzeń będących trójkami:

- podmiot (ang. subject) – zasób,
- orzeczenie (ang. predicate) – właściwość podmiotu,
- obiekt (ang. object) –zasób lub literał.

W ten sposób RDF umożliwia zapis stwierdzeń o zasobach jako grafu skierowanego, którego wierzchołki są zasobami, a krawędzie są etykietowane właściwościami (tzw. graf RDF).

RDF Schema

RDF Schema [48] pozwala definiować słownictwo wykorzystywane w grafach RDF. Umożliwia on deklarowanie klas (ang. class) i właściwości (ang. property) zasobów RDF. Jest on rozszerzalnym językiem reprezentacji wiedzy, dostarczającym podstawowe elementy słownika RDF [41].

OWL

OWL (ang. Ontology Web Language) [49] jest ogólnym językiem opisu ontologii, opartym na RDF. Rozszerza swojego poprzednika RDF Schema o możliwości opisu właściwości i klas za pomocą dodatkowych operatorów, między innymi: określania relacji między klasami (klasy rozłączne, przecięcie klas, suma klas, dopełnienie klasy), określania hierarchii właściwości, dziedziny właściwości, właściwości odwrotnych, symetrycznych, przechodnich czy ograniczeń kardynalnych (liczności elementów). OWL jest językiem zaprojektowanym dla aplikacji mających przetwarzać informacje, a nie tylko je prezentować [41]. Istnieją trzy wersje języka OWL, w kolejności od najmniej do najbardziej ogólnego są to: OWL Lite, OWL DL, and OWL Full.

SPARQL

SPARQL [50] jest językiem zapytań dla formatu RDF. Zapytania SPARQL składają się z warunków przypominających trójki RDF, gdzie podmiot, orzeczenie i obiekt mogą dodatkowo zawierać zmienne (rozpoczynające się od znaku „?”), jak również klauzuli określającej źródło danych RDF. Dane spełniają zapytanie jeśli reprezentują podgraf RDF odpowiadający warunkom zapytania.

Sieci neuronowe

Sztuczne sieci neuronowe przetwarzają sygnał wejściowy za pomocą mniej lub bardziej złożonych struktur zbudowanych z neuronów. Są one zbudowane na wzór tkanek nerwowych mózgu. Pojedynczy neuron realizuje proste operacje matematyczne. Uwzględniając wagi poszczególnych wejść przetwarza sygnał wejściowy na sygnał wyjściowy. Wiedza niezbędna do rozwiązania problemu zakodowana jest w wagach wszystkich neuronów oraz w topologii sieci. Proces automatycznego ustalania się wag pod wpływem określonych sygnałów podawanych na wejściu sieci nazywany jest uczeniem sieci. Sztuczne sieci neuronowe dają możliwość modelowania zjawisk oraz procesów słabo ustrukturalizowanych i zalgorytmizowanych, dostarczając modelu bez potrzeby określania natury związków między zmiennymi [37].

Algorytmy genetyczne

Algorytmy genetyczne są podejściem inspirowanym mechanizmami ewolucyjnymi występującymi w naturze, takimi jak mutacja, krzyżowanie, dobór naturalny. Algorytm genetyczny symuluje populacje osobników, które cyklicznie poddawane są eliminacji. Eliminowane są osobniki uzyskujące najniższy wynik funkcji oceny, która to funkcja odzwierciedla efektywność rozwiązywania problemu przez danego osobnika. Kod genetyczny najlepszych osobników, wyrażony jako ciąg bitów, poddawany jest mutacji – losowym zmianom. Najlepsze osobniki poddawane są też krzyżowaniu – wymianie części kodu genetycznego. Algorytmy genetyczne w heurystyczny sposób przeszukują przestrzeń rozwiązań danego problemu.

2.5. Przegląd literatury związanej ze wspomaganiem projektowania

W tym rozdziale przedstawiony jest przegląd wybranych prac z dziedziny wspomagania projektowania architektonicznego oraz wnioskowania na grafach. Daje się zauważyć rozwój badań w dziedzinie wspomagania projektowania, możliwy dzięki wprowadzeniu BIM. Pomimo tego, jedynym systemem sprawdzającym projekty mogącym być użytym w przemyśle jest system CORENET, opracowany w Singapurze na potrzeby weryfikacji tamtejszych norm budowlanych.

Autorzy pracy „Trends in built environment semantic Web applications: Where are we today?” [46] przeanalizowali ponad 120 publikacji dotyczących zastosowań Semantic Web w szeroko rozumianym sektorze budownictwa w okresie 2001-2013. W ich pracy można znaleźć tabelaryczne podsumowanie tej analizy. Stwierdzają oni, że podczas gdy wiele prac

jest poświęconych badaniom, na tym etapie rozwoju mało jest powszechnych zastosowań. Powstał szereg prac dotyczących ograniczenia strat energii oraz problematyki ograniczenia emisji dwutlenku węgla. Coraz więcej prac poświęconych jest zaawansowanym technologiom takim jak Linked Data. Pojawiają się też badania pokazujące nowe możliwości BIM w połączeniu z Semantic Web, obliczeniami w chmurze oraz bezstanowymi usługami WWW (ang. stateless Web Services).

Znacznie mniej prac dotyczy fazy koncepcyjnej. Jedynymi ich przykładami są opisane poniżej prace podejmujące aspekt generacyjny (SEED), podejmujące aspekt generacyjny w połączeniu z analizą wymagań w sensie modelu Function-Structure-Behavior (GraCAD), oraz inżynierski system ConEd wspierający analizę strukturalną. Z kolei system ConDes korzysta z grafów oraz ontologii w celu sprawdzania reguł. Wymusza on jednak na użytkowniku operowanie na grafach zamiast na szkicach, będących naturalnym językiem fazy wstępnej, co uniemożliwia kreatywną prace nad większymi projektami.

2.5.1. Systemy wspierające fazę koncepcyjną

GraCAD

W swojej rozprawie doktorskiej [1] Janusz Szuba prezentuje GraCAD: metodologię oraz realizujące ją narzędzie projektowania w fazie koncepcyjnej.

W metodologii GraCAD przypadki użycia, wymagania funkcjonalne oraz ograniczenia projektowe są zapisywane przy użyciu struktur grafowych, w szczególności diagramów czynności UML (UML jest językiem stworzonym do modelowania systemów w inżynierii oprogramowania). Kolejne fazy tej metodologii projektowania to:

- określenie użytkowników budynku, oraz typowych dla nich przypadków użycia (ang. use cases),
- zapis przypadków użycia w formie diagramu czynności UML,
- stworzenie grafu obszarów funkcjonalnych budynku,
- określenie odwzorowania pomiędzy diagramem czynności a grafem obszarów,
- podział obszarów na pokoje – stworzenie grafu pokoi (ang. room graph),
- doprecyzowanie stworzonego odwzorowania, przypisanie elementom diagramu czynności konkretnych pokoi, w których będą one wykonywane.

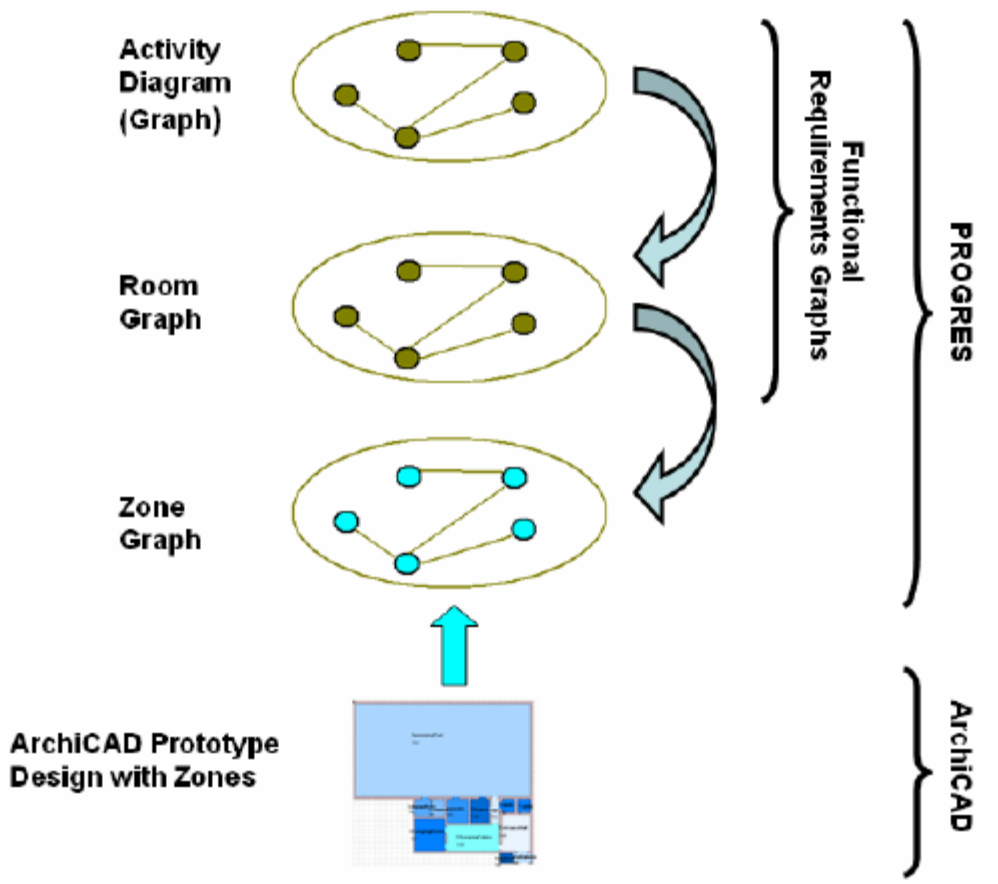
Metodologia GraCAD nawiązuje do modelu Function-Structure-Behavior Johna S. Gero [51], gdzie czynności przypisane do pokoi definiują funkcje (Function), grafy pokoi i obszarów definiują strukturę (Structure), a diagram czynności zachowanie (Behavior).

System GraCAD zbudowany jest w oparciu o system transformacji grafowych PROGRES (PROgrammed Graph REwriting System) [52] oraz o ArchiCAD, komercyjne narzędzie CAD dla architektów. Według twórcy systemu, GraCAD jest opartym na wiedzy systemem

wspomagania fazy koncepcyjnej projektowania architektonicznego. Środowisko to składa się z:

- edytora grafów GraCAD, stworzonego za pomocą narzędzi PROGRES oraz UPGRADE [53]. Za jego pomocą definiuje się diagram czynności, graf pokoi oraz ręcznie zaznacza odwzorowania między nimi,
- programu ArchiCAD z dodatkiem(pluginem) GraCAD, za pomocą którego tworzy się rozkłady pomieszczeń.

Po wprowadzeniu rozkładu pomieszczeń w programie ArchiCAD, generowany jest graf stref (ang. zone graph), zawierający informacje o pomieszczeniach i relacjach między nimi. Następnie, już w edytorze GraCAD, ręcznie przyporządkowuje się odpowiadające sobie elementy grafu pokoi i grafu stref, czyli zapewnia się połączenie między fizycznymi artefaktami rozkładu pomieszczeń oraz elementami grafów reprezentujących wymagania funkcjonalne (Rysunek 2.4).



Rysunek 2.4 Elementy systemu GraCAD [1]

Dzięki określeniu tego odwzorowania możliwe jest przeprowadzanie sprawdzania poprawności projektu. Sprowadzanie to jest realizowane za pomocą operacji na grafach

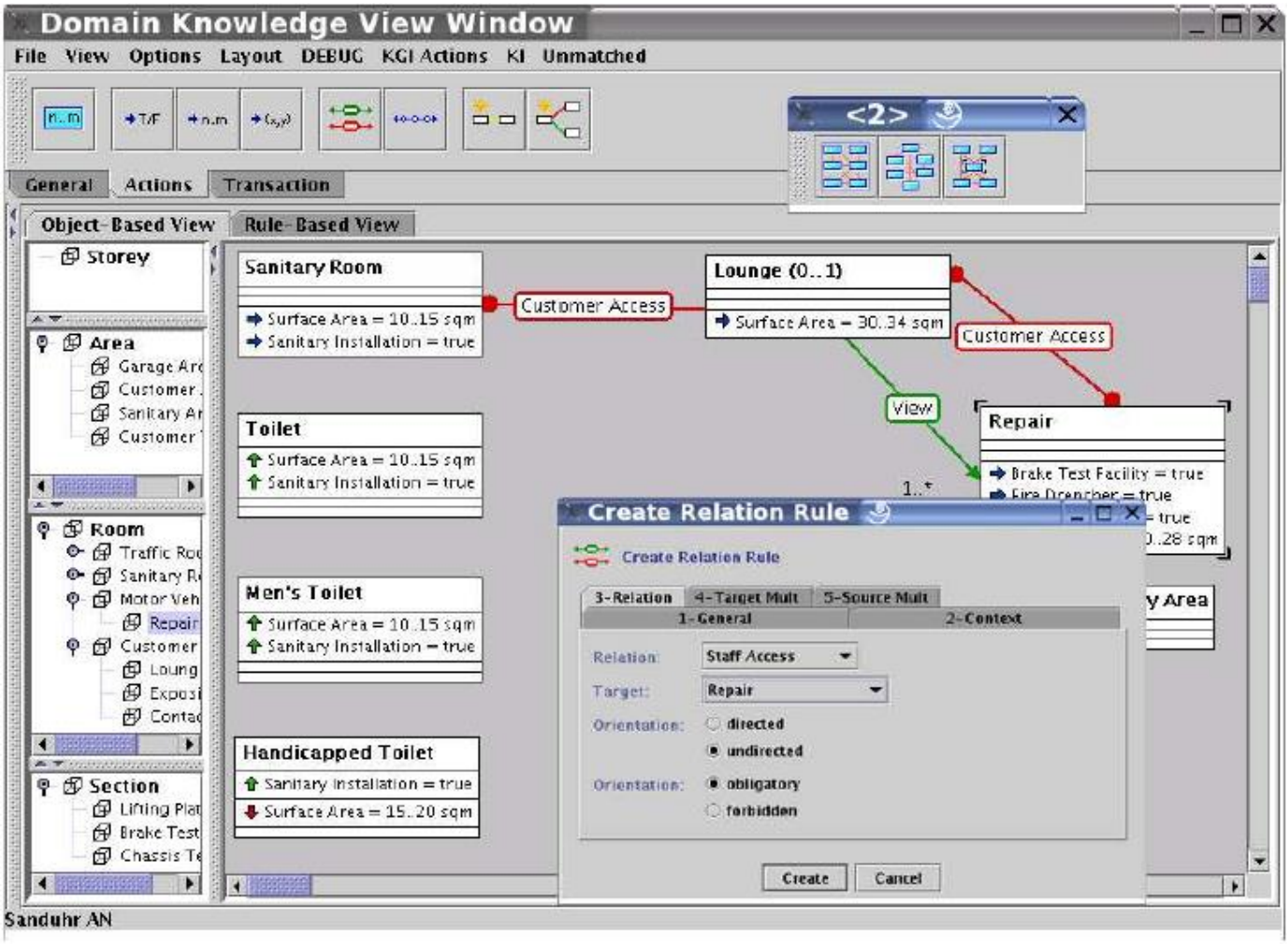
systemu PROGRES. Sprawdzane są: odpowiedniość między wymaganiami funkcjonalnymi i stworzonymi pomieszczeniami oraz czy połączenia między pomieszczeniami odpowiadają sekwencji elementów w grafie czynności (uwzględniane są tutaj połączenia bezpośrednie oraz pośrednie poprzez inne pomieszczenia).

Powyższa praca stanowi ciekawe połączenie fazy analizy wymagań z tworzeniem wstępnych szkiców projektu. Poważnym mankamentem systemu GraCAD jest jednak duża liczba odwzorowań, które trzeba wprowadzić ręcznie, co jest dość monotonnym zajęciem niewnoszącym wartości dodanej do procesu projektowania. Autor słusznie zauważa, że możliwe byłoby generowanie początkowego rozkładu pomieszczeń na podstawie stworzonych grafów wymagań funkcjonalnych. Istotnym rozszerzeniem systemu byłoby sprawdzanie ogólnych ograniczeń architektonicznych, które nie zostało zaimplementowane, gdyż atrybuty lokalizacji elementów projektu nie są obecne na poziomie analizy w PROGRES. Autor zaplanował uzupełnienie tej luki w przyszłych pracach.

ConDes

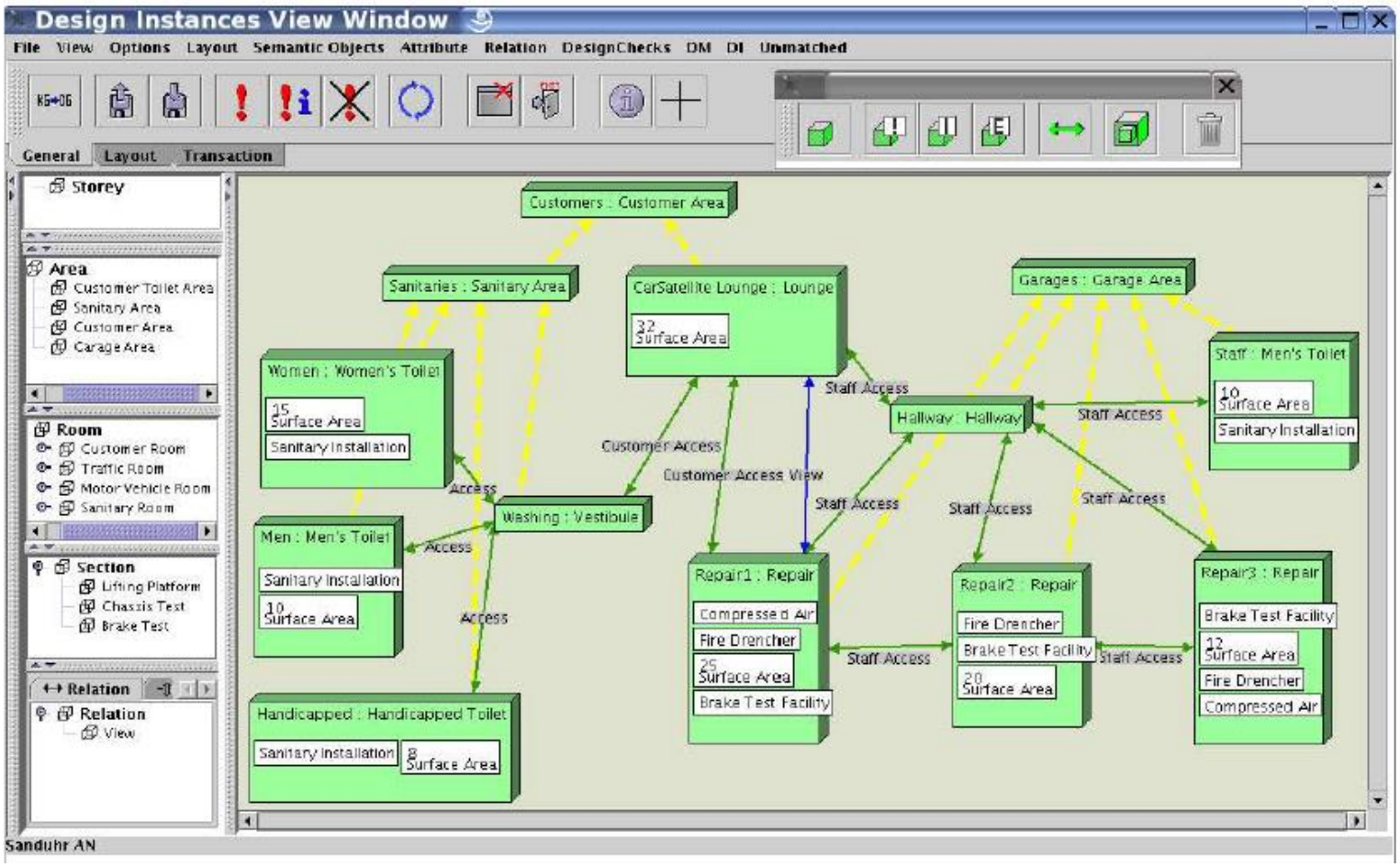
Bodo Kraft oraz Manfred Nagl przedstawiają w swoich pracach [7] system *ConDes*, który koncentruje się na fazie koncepcyjnej. Jest to oparty na grafach system wspomagania projektowania architektonicznego. Składa się on z wielu modułów. Wszystkie one są wyspecjalizowanymi do różnych celów edytorami grafów, stworzonymi przy użyciu narzędzi PROGRES oraz UPGRADE [53]. Za pomocą edytora ontologii (*Domain Ontology Editor*) ekspert z danej dziedziny, w domyśle inżynier wiedzy, dokonuje formalizacji wiedzy zapisując ją w formie ontologii. Autorzy podkreślają możliwość definiowania ontologii przez użytkowników systemu, oraz użyteczność definiowania wielu ontologii dla różnych klas budynków. Elementy ontologii są składowymi reguł projektowych, które następnie są definiowane za pomocą edytora wiedzy (*Domain Knowledge Editor* - Rysunek 2.5) przez inżyniera wiedzy. Dla samego projektowania koncepcyjnego autorzy proponują architektom edytor grafów projektowych (*Design Graph Editor* - Rysunek 2.6). Narzędzie to pozwala tworzyć projekt koncepcyjny, gdzie pomieszczenia oraz większe obszary to wierzchołki grafu, a cechy pomieszczeń takie jak powierzchnia zapisywane są za pomocą atrybutów wierzchołków.

Wprowadzone projekty koncepcyjne poddawane są analizie pod względem spełniania reguł, definiowanych również w postaci grafów. W przypadku niespełniania którejś z reguł, prezentowane są specjalne komunikaty. Analiza wykonywana jest na podstawie operacji grafowych, udostępnianych przez wyżej wspomniane systemy PROGRES oraz UPGRADE. Istniejące klasy reguł to: atrybutowe (wymuszają wartość atrybutu dla obiektu), relacyjne (narzucają lub zakazują istnienie relacji między obiektami) oraz liczbowe (narzucają liczbę pewnych obiektów w projekcie).



Rysunek 2.5 Narzędzie *Domain Knowledge Editor* [7]

Wcześniejsze prace tych autorów [54] badają możliwość interakcji systemów sprawdzających projekty z istniejącym narzędziem CAD - programem ArchiCAD. Wykorzystując podobne mechanizmy zapisu i analizy projektu jak w systemie ConDes, oraz używając pewnych funkcji ArchiCADa, wzbogacają go o możliwość wprowadzania pokoi oraz obszarów. W tle generowany jest zapis grafowy projektu, oraz sprawdzane są reguły. Rozkład pomieszczeń nie jest niestety właściwym elementem projektu ArchiCADa, ale dodatkową warstwą stworzoną za pomocą GDL - języka pierwotnie używanego do nanoszenia dodatkowych elementów na projekt ArchiCAD. Autorzy stworzyli jednak procedurę generującą właściwe ściany, na podstawie rozkładu pomieszczeń stworzonego w GDL. Z kolei w pracy [3] proponują oni przeniesienie mechanizmów sprawdzania poza ArchiCAD, do centralnej bazy wiedzy. Konkretne projekty do sprawdzenia byłyby wysyłane do tej bazy z poziomu ArchiCADa. Takie rozwiązanie motywowane jest większymi możliwościami sprawdzania reguł poza ArchiCADem, oraz łatwością uaktualniana jednej, centralnej bazy.



Rysunek 2.6 Narzędzie *Design Graph Editor* [7]

System *ConDes* posiada wspólne cechy z systemem HSSDR: reprezentacja projektu w formie grafu oraz umożliwienie użytkownikom definiowania reguł projektowych. Jego przewagą w stosunku do HSSDR jest możliwość definiowania ontologii i korzyści płynące z użycia ich podczas sprawdzania projektów, wadą natomiast siła wyrazu, oraz sposób projektowania. Rzeczywiście siła wyrazu reguł wydaje się być zbyt mała, zauważają to również autorzy i planują ją powiększyć o operatory logiczne pozwalające na łączenie reguł ze sobą. Projektowanie poprzez operowanie na grafach zamiast na rozkładach pomieszczeń jest według autora poważną wadą systemu. Doświadczenia z prac nad systemem HSSDR pokazują, że grafy dobrze funkcjonujące jako reprezentacja wewnętrzna nie sprawdzają się podczas tworzenia lub przeglądania projektów – liczba elementów składowych staje się w tym przypadku zbyt duża. Diagramy projektowe natomiast są naturalnym językiem architektów oraz bodźcem dla kreatywności w projektowaniu, przy tym jak pokazuje niniejsza rozprawa, mogą być automatycznie przetwarzane do postaci grafu.

SEED

Projekt „Software Environment to Support the Early Phases in Building Design” (SEED) [55] [56] stara się zapewnić wsparcie obliczeniowe wstępnym fazom projektowania architektonicznego. Cały system został stworzony w architekturze modułowej, z podziałem obowiązków na poszczególne moduły i możliwością rozszerzania. Projekt skupia się na aspekcie generacyjnym wspierania projektowania – generowaniu rozkładów pomieszczeń, a nawet trójwymiarowych konfiguracji na podstawie wstępnych wymagań. Takie podejście stwarza architektom możliwość szybkiej generacji wielu początkowych rozwiązań i dokonywania spośród nich wyboru projektów do dalszych prac. Plany przewidują rozszerzenie systemu o sprawdzanie ograniczeń projektowych [57], jednak nie zostały przedstawione konkretne możliwości systemu w tym względzie.

ConEd

System ConEd, zapoczątkowany jako praca naukowa [58], jest obecnie dostępną na rynku aplikacją komercyjną [59] [60]. Jego zadaniem jest wspomaganie projektowania w fazie koncepcyjnej, jednak inaczej niż prezentowane tutaj systemy obejmuję on aspekt inżynieryjny a nie architektoniczny, a mianowicie analizę strukturalną budynku. Jest to system CAD wyspecjalizowany w projektowaniu struktury nośnej budynku. Inżynierowie planujący elementy nośne budynku mogą dzięki niemu w efektywniejszy sposób opracować rozmieszczenie kolumn, belek i innych elementów nośnych oraz analizować rozkład sił na tych elementach, przez co reagowanie na zmiany w projekcie staje się szybsze i mniej kosztowne. CondEd buduje mechaniczny model budynku na podstawie projektu wprowadzonego przez użytkownika w edytorze graficznym. W tym celu wykonywana jest analiza geometryczna obiektów i relacji pomiędzy nimi, oraz rozkładu sił na poszczególnych elementach. ConEd korzysta z szeregu bibliotek do analizy geometrycznej oraz do obliczeń mechanicznych.

2.5.2. Systemy analizujące projekty w formacie IFC

IFC jest uniwersalnym formatem zapisu projektów wspieranym przez większość narzędzi BIM. Dlatego też jest dobrym formatem wejściowym dla systemów analizujących gotowe projekty. Autorzy pracy [11] zbadali systemy analizujące projekty architektoniczne w formacie IFC i w swojej pracy przedstawili porównanie, w którym znalazło się pięć takich systemów. We wszystkich opisanych systemach reguły są zapisane bądź w kodzie programu bądź w formie umożliwiającej pewne modyfikacje. W drugim przypadku proste konstrukcje jak parametry czy warunki umożliwiają pewne dostosowanie zbioru dostępnych reguł. Według autorów porównania, w dłuższej perspektywie reguły powinny być zapisywane w

ogólnym języku, który umożliwiłby zapis dowolnych reguł oraz operowałby na dowolnym formacie projektu, lub też mógłby być ewaluowany w dowolnym narzędziu wspomagania projektowania.

Wszystkie badane systemy maja duże wymagania co do struktury wejściowego projektu. Pewne atrybuty i nazwy projektu IFC muszą się zgadzać z wymaganymi przez konkretny system, a nadkładanie się pewnych obiektów może uniemożliwić analizę. Powoduje to dodatkowy narzut pracy podczas wstępnej obróbki projektów. Aby uniknąć tego problemu, oprócz zwiększania dojrzałości systemów analizujących projekty IFC, autorzy wyżej wspomnianej pracy wskazują na potrzebę stworzenia uniwersalnych wytycznych tworzenia poprawnych projektów na potrzeby analizy, jakie byłyby stosowane przez twórców systemów wspomagania projektowania.

Format IFC dobrze sprawdzający się jako uniwersalny format wymiany danych pomiędzy aplikacjami nie jest efektywny gdy przychodzi do przetwarzania danych projektu. Ze względu na skomplikowaną, hierarchiczną strukturę pliku, po wczytaniu do pamięci komputera dane zajmują średnio 4-10 razy więcej niż dane w pliku na dysku [20]. Również manipulacja elementami projektu jest nieefektywna. Aby uniknąć tego typu problemów, zwłaszcza podczas prac nad dużymi projektami, można stosować podejście polegające na wczytaniu elementów projektu IFC do centralnej bazy danych, gdzie organizacja danych jest zoptymalizowana pod kątem typowych operacji. Przykładem takiej bazy danych jest BIMserver [61]. Jest to platforma umożliwiająca operacje na modelu IFC, stworzona jednocześnie w oparciu o otwartą architekturę oferującą możliwość tworzenia własnych rozszerzeń jako pluginów oraz wymiany danych za pośrednictwem interfejsów takich jak SOAP [62] czy JSON [63].

Poniżej scharakteryzowane zostało pięć systemów zawartych w porównaniu Eastmana [11] oraz nieobjęte tym porównaniem, nowsze systemy AUTOCodes i C3R.

CORENET

System CORENET [64] jest tworzony w Singapurze przez rządową agencje Singapore Building and Construction Authority. Projekt stara się realizować sprawdzanie standardów prawa budowlanego dla planów budynków oraz zgodności ze standardami instalacji budynku. Sprawdzanie planów obejmuje reguły związane z zarządzaniem budynkiem, przepisami przeciwpożarowymi, dostępnością dla osób niepełnosprawnych, normy środowiskowe. Reguły te są zakodowane w kodzie źródłowym programu. CORENET operuje na danych wejściowych w formacie IFC. Żeby umożliwić sprawdzanie właściwości niedających się bezpośrednio wywieść z danych IFC, została dodana pośrednia warstwa danych o nazwie FORNAX. Obiekty FORNAX reprezentują wysokopoziomowe składowe projektu oraz udostępniają szereg funkcji i atrybutów tych obiektów. Reguły, w trakcie wykonywania odpytują obiekty FORNAX. CORENET jest platformą webową, po sprawdzeniu

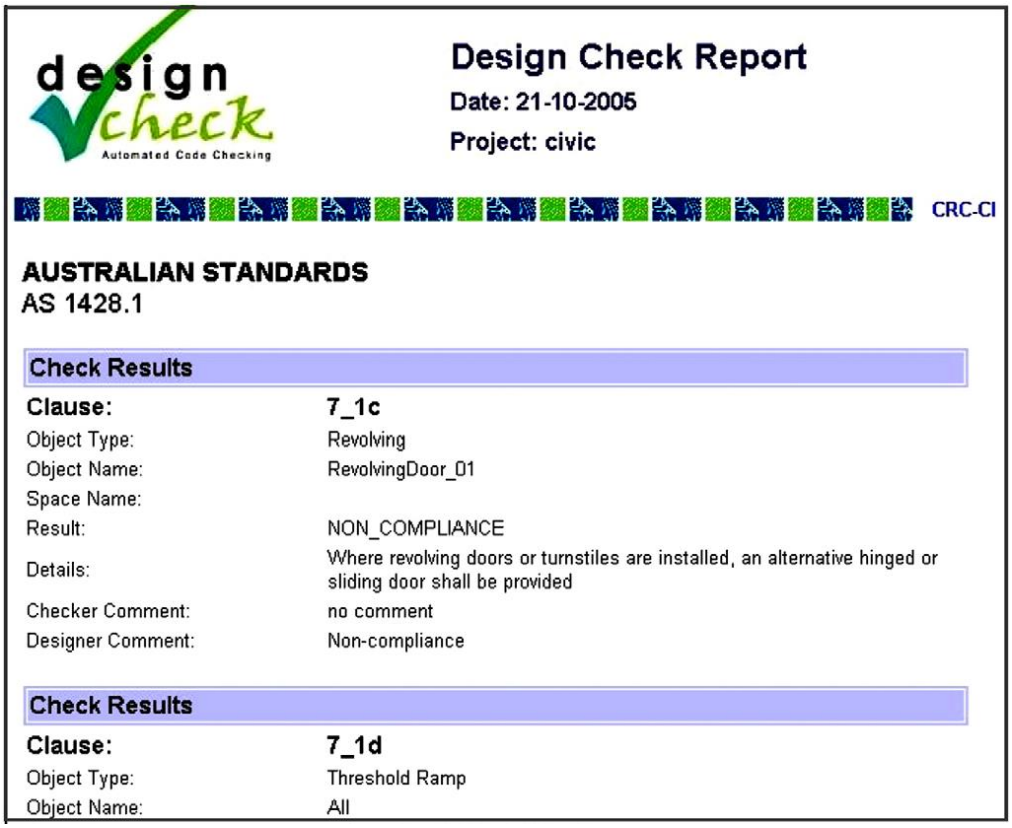
projektu generowany jest raport w różnych formatach takich jak HTML czy PDF. CORENET jest najbardziej dojrzałym systemem automatycznego sprawdzania projektów [11], którego użycie jest rozpowszechnione wśród firm i podmiotów w kraju, z którego pochodzi. Wg twórców zapewnia pokrycie 92% standardów prawa budowlanego dla planów budynków.

HITOS

Norweski projekt HITOS [65] realizowany dla rządowej agencji Statsbygg posiada dwa mechanizmy sprawdzania projektów w formacie IFC. Pierwszy z nich, dRofus ogranicza się do porównywania wymagań odnośnie powierzchni pokoi z faktyczną powierzchnią tych pokoi na planie budynku. Drugi z nich, oparty na Solibri Model Checker, umożliwia sprawdzanie reguł dotyczących dostępności pomieszczeń dla osób niepełnosprawnych. Reguły pochodzą z dwóch międzynarodowych standardów: International *ISO/CD 21542* oraz *International Code Council ICC/ANSI A117.1.* Możliwości systemu ograniczają się do sprawdzania wcześniej wprowadzonych przez twórców reguł. Istnieje możliwość ich parametryzacji w celu dostosowania do potrzeb lokalnych przepisów lub konkretnego projektu. Parametry jakie można modyfikować to np. minimalna szerokość korytarza czy maksymalna wysokość umywalki.

Design Check

Australian Building Codes Board (ABCB) z ramienia rządu Australii realizowała projekt mający na celu stworzenie platformy weryfikującej standardy dostępności budynków zdefiniowane w wytycznych Australian Standard (AS) 1428.1 [66]. W tym celu został użyty komercyjny system Express Data Manager (EDM) [67], będący obiektowym interfejsem dla elementów IFC. Odpytywanie właściwości pojedynczych obiektów nie wystarczy aby zapewnić sprawdzanie relacji dostępności pomiędzy pomieszczeniami, dlatego system został rozszerzony o funkcję budowania grafu dostępności. W definiowaniu reguł został zastosowany krok pośredni pomiędzy opisem słownym, a implementacją w systemie. Standardy budowlane są zapisywane w postaci pseudo kodu, a następnie ręcznie tłumaczone na wewnętrzny język systemu. Krok ten wprowadza pewien stopień formalizacji na etapie definiowania reguł, nie udostępnia jednak możliwości ich edycji przez użytkowników. Twórcy zapewnili bogate możliwości raportowania wykrytych błędów formie tekstowej (Rysunek 2.7).



design check
Automated Code Checking

Design Check Report

Date: 21-10-2005

Project: civic

CRC-CI

AUSTRALIAN STANDARDS
AS 1428.1

Check Results

Clause:	**7_1c**
Object Type:	Revolving
Object Name:	RevolvingDoor_01
Space Name:	
Result:	NON_COMPLIANCE
Details:	Where revolving doors or turnstiles are installed, an alternative hinged or sliding door shall be provided
Checker Comment:	no comment
Designer Comment:	Non-compliance

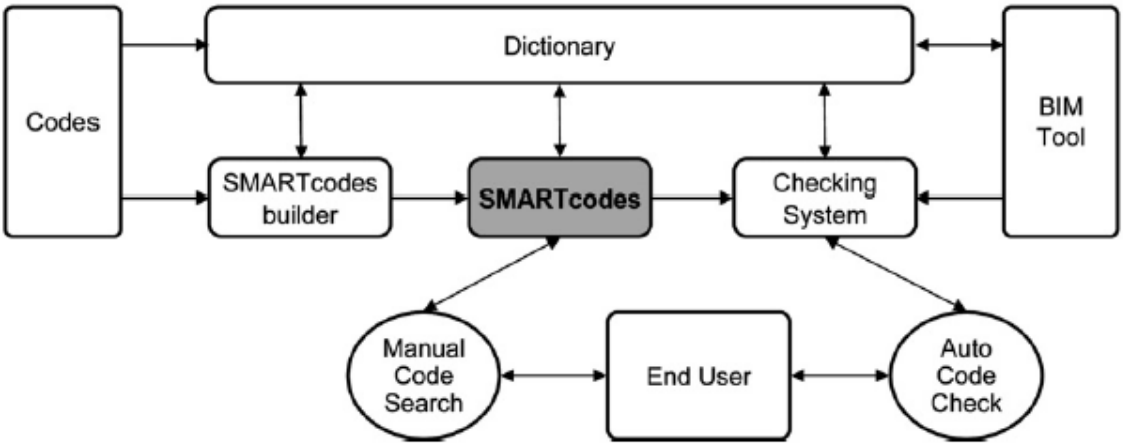
Check Results

Clause:	**7_1d**
Object Type:	Threshold Ramp
Object Name:	All

Rysunek 2.7 Fragment raportu o błędach [11]

SMARTcodes

W Stanach Zjednoczonych agencją odpowiedzialną za tworzenie standardów budowlanych jest International Code Council (ICC) [68]. Agencja ta zainicjowała projekt SMARTcodes mający na celu automatyczne sprawdzanie zgodności projektów. SMARTcodes Builder jest częścią systemu odpowiedzialną za definiowanie reguł. Udostępniając predefiniowany słownik pojęć stara się on eliminować błędy pojawiające się podczas interpretacji reguł w formie przepisów na zapis zrozumiały przez komputery. Dla elementów słownika określone są właściwości, typy danych oraz jednostki miary. Słownik jest używany w procesie interpretacji reguł, a także podczas ekstrakcji danych z modelu IFC. Poniżej przedstawiony jest schemat całego systemu (Rysunek 2.8).

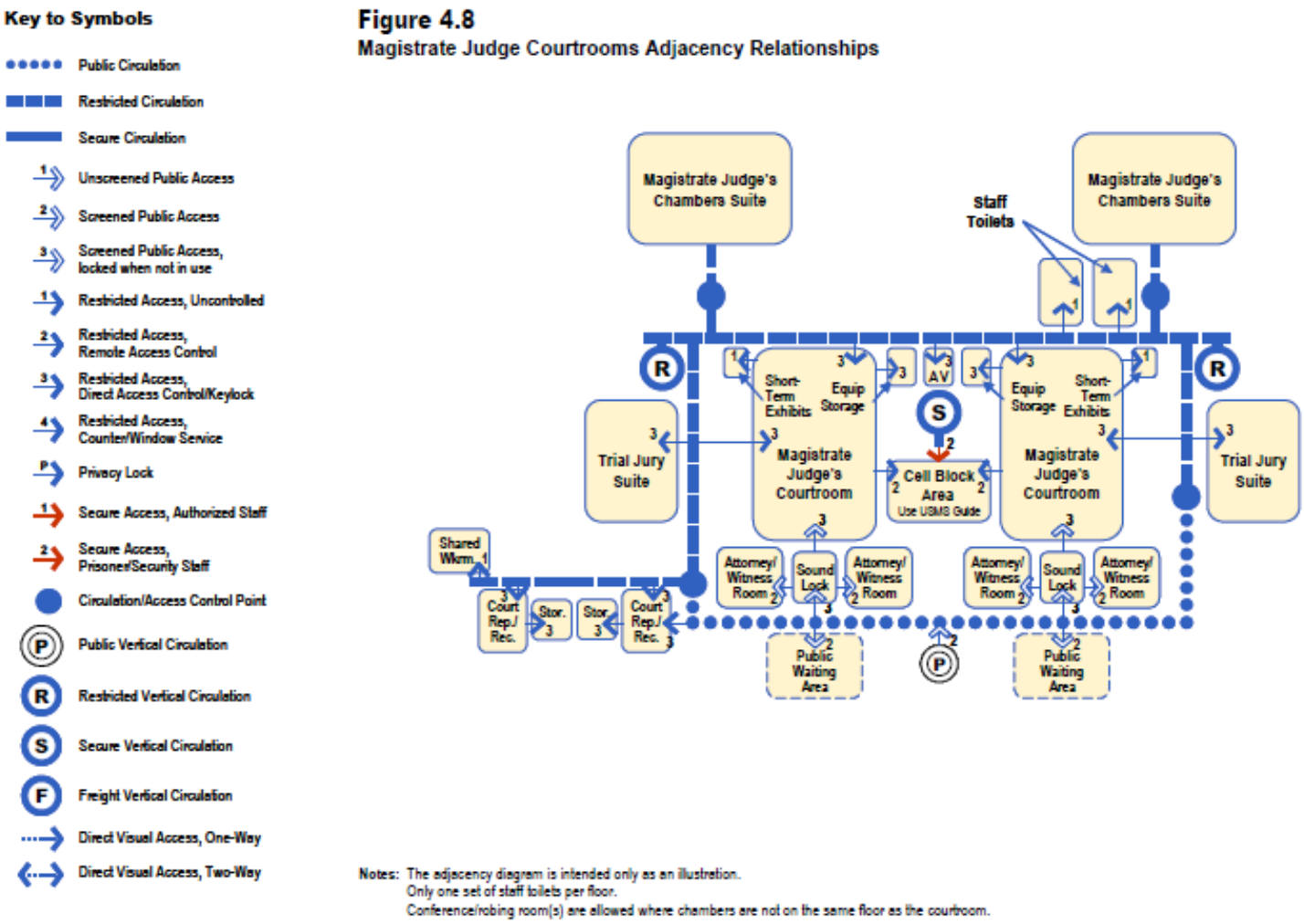


Rysunek 2.8 Schemat systemu bazującego na SMARTcodes [11]

W zamyśle twórców, możliwe będzie wysyłanie projektów do sprawdzanie poprzez stronę internetową, jednak z uwagi na wczesną fazę prac projekty na wejściu systemu ograniczone są do kilku przykładów przygotowanych przez twórców. System prezentuję kompleksowe podejście do problemu, jednak jego możliwości obejmują jedynie proste reguły mówiące o pojedynczych właściwościach obiektów [11]. Projekt prezentował interesujące podejście w którym urząd definiujący standardy budowlane tworzył narzędzia do ich automatycznej weryfikacji, został on niestety zawieszony w 2010 roku z powodu braku funduszy [69].

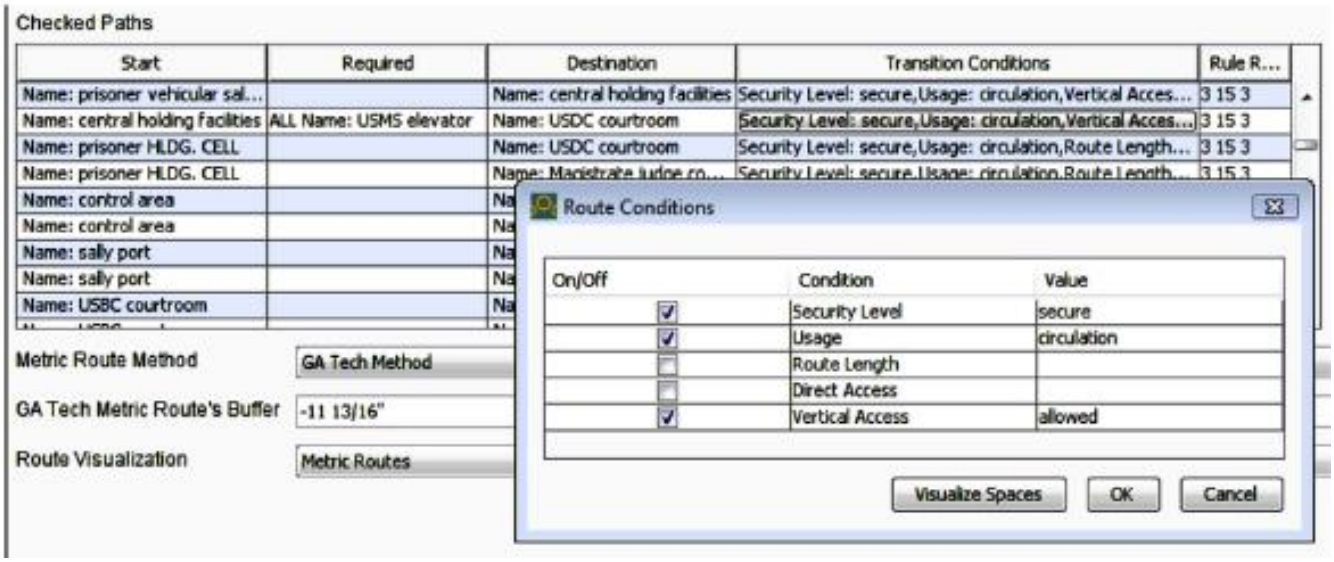
Design Assessment Tool

General Services Administration (GSA) jest agencją rządową, która w ramach programu 3D-4D Building Information Modeling [70] zleciła stworzenie dwóch systemów analizujących projekty architektoniczne. Pierwszy z nich analizuje zgodność powierzchni pomieszczeń w budynku z normami. Weryfikacje bardziej złożonych reguł umożliwia drugi z nich: Design Assessment Tool (DAT), system stworzony w celu weryfikacji reguł projektowania budynków sądów w Stanach Zjednoczonych. GSA tworzy wytyczne odnośnie projektowania budynków sądów, są one zawarte w niezwykle kompleksowym dokumencie U.S. Courts Design Guide [71]. Poniżej znajduje się przykład grafu relacji dostępności pomiędzy pomieszczeniami dla jednego z rodzajów sądów opisanych w dokumencie (Rysunek 2.9). W całym dokumencie zidentyfikowano 302 reguły, które zostały użyte w systemie DAT.



Rysunek 2.9 Graf dostępności pomieszczeń pochodzący z U.S. Courts Design Guide [71]

Aby projekt mógł być sprawdzany przez DAT musi on przestrzegać pewnych konwencji i reguł zdefiniowanych przez GSA, takich jak nazewnictwo pomieszczeń czy określenie pewnych atrybutów. Warunki te są sprawdzane przed rozpoczęciem właściwej analizy. Na potrzeby analizy na podstawie planu budynku generowane są dwa grafy: graf reprezentujący topologie połączeń pomiędzy pomieszczeniami oraz graf ścieżek odzwierciedlający trasy pokonywane przez użytkowników budynku. Wszystkie reguły systemu są ustalone, natomiast możliwa jest modyfikacja niektórych parametrów (Rysunek 2.10). Po wykonaniu analizy prezentowane są raporty o naruszeniach, również w formie graficznej. Cennym elementem informacji o naruszeniu jest odniesienie do fragmentu dokumentu U.S. Courts Design Guide będącego źródłem danej reguły. System jest używany do sprawdzania projektów podlegających GSA. Około 10% jego operacji nie jest zautomatyzowana i wymaga ręcznej interwencji [11].



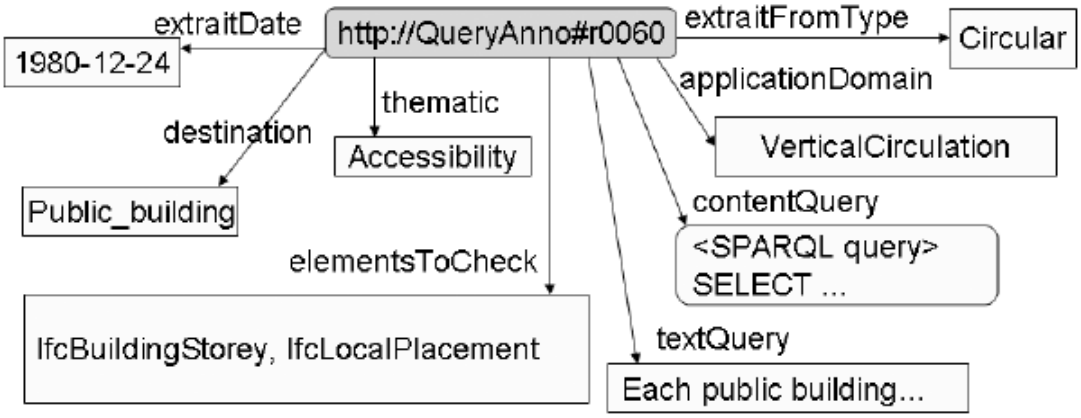
Rysunek 2.10 Parametry systemu DAT [11]

AUTOCodes

Nowym projektem, będącym obecnie w fazie prototypowej jest AUTOCodes [72]. Jest on tworzony przez ICC, Fiatech oraz kilka innych firm oraz jednostek badawczych. Autorzy projektu deklarują, że jego celem jest wyeliminowanie ręcznego sprawdzania zgodności planów architektonicznych przez urzędy w Stanach Zjednoczonych [69]. Dalsze szczegóły projektu nie są jak na razie znane.

C3R

Autorzy prac [28] [41] [73] przedstawiają system C3R, w którym zastosowano podejście ontologiczne do sprawdzania zgodności projektów architektonicznych z normami. Normy te są zapisane jako zapytania SPARQL, odwołujące się do elementów modelu IFC. Zapytania te zostały dodatkowo wzbogacone semantycznymi adnotacjami RDF o takie informacje jak: nazwa, komentarz, odniesienie do źródła reguły, ale też kontekst zastosowania – np. maksymalna wysokość poręczy przy schodach różna dla dorosłych i dla dzieci. Przykład pokazujący takie adnotacje w uproszczonej formie znajduje się na rysunku Rysunek 2.11. Adnotacje semantyczne pozwalają według autorów na zawarcie w systemie ukrytej wiedzy ekspertów z dziedziny, a także na lepszą organizację testów zgodności oraz prezentacje informacji o naruszeniach w bardziej zrozumiałej formie.

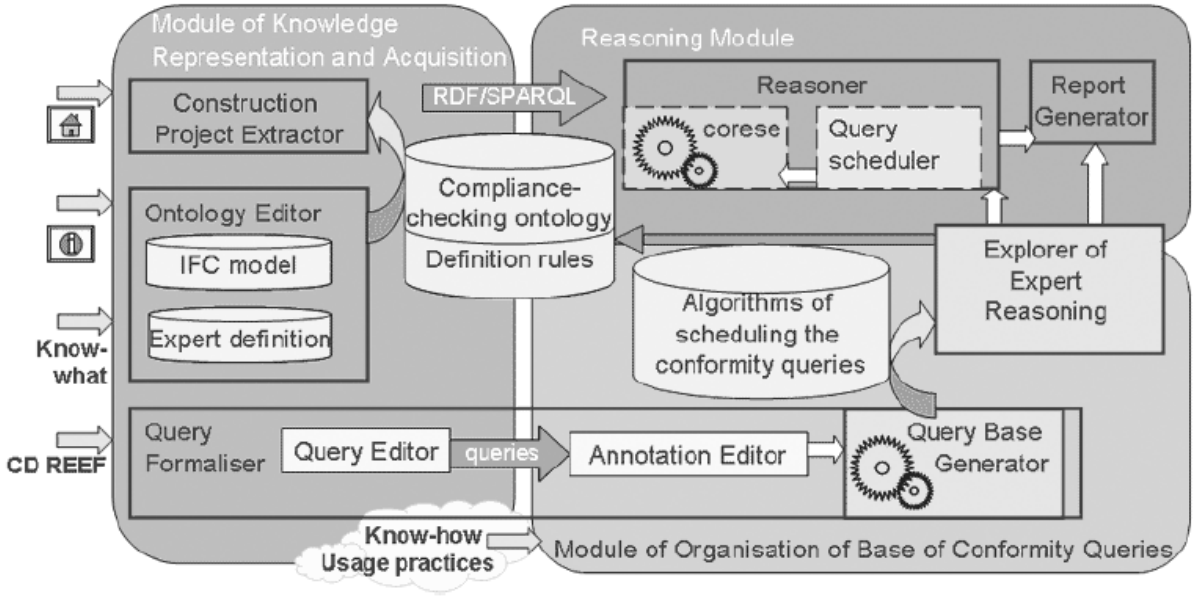


Rysunek 2.11 Przykład adnotacji semantycznej systemy C3R - forma uproszczona [10]

Na podstawie elementów występujących w schemacie ifcXML, dynamicznie tworzona jest ontologia sprawdzania zgodności (ang. conformity-checking ontology). Ograniczana jest ona do pojęć występujących w bazie zgromadzonych zapytań SAPRQL. Ontologia zapisywana jest w języku OWL Lite. Struktura ta wzbogacana jest dodatkowo o wiedzę ekspertów, która również może być zapisana jako grafy RDF. Autorzy podają jako przykład regułę: jeśli piętro znajduje się na poziomie głównego wejścia do budynku, przypisz temu piętru etykietę „parter".

Na podstawie modelu IFC projektu, a dokładniej modelu ifcXML, tworzona jest reprezentacja w języku RDF. W tym celu stworzona przez autorów transformata XSL wykonywana jest na danych wejściowych. Reprezentacja RDF jest wzbogacona o dodatkowe atrybuty poprzez zastosowanie reguł zapisanych wcześniej jako grafy RDF.

Sama walidacja projektu będącą wykonywaniem zapytań SPARQL względem reprezentacji RDF, polega na dopasowywaniu grafów testów zgodności z fragmentami grafu reprezentacji budynku w języku RDF. Architektura systemu C3R przedstawiona jest na ilustracji poniżej (Rysunek 2.12). Zaimplementowane zostały jak na razie podstawowe moduły systemu [10]. W systemie C3R wszystkie reguły zapisane są w konwencji negatywnej, tzn. występowanie podgrafu opisanego jako reguła w projekcie oznacza naruszenie jakiegoś standardu. Możliwy jest również przypadek, że w reprezentacji projektu nie ma wystarczających danych aby sprawdzić pewien warunek, w takim przypadku system również generuje ostrzeżenie.

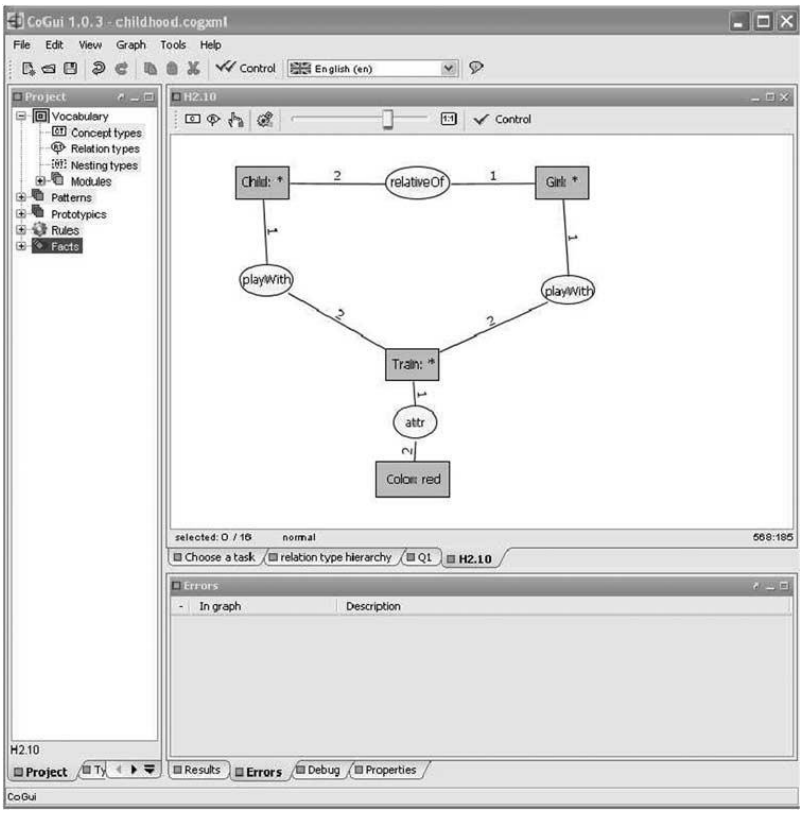


Rysunek 2.12 Architektura prototypu systemy C3R [10]

2.5.3. Pozostałe prace

Grafowy system Cheina

Chein et al. [36] podkreślają rolę reprezentacji pośredniczącej (ang. mediating representation) w systemach z bazą wiedzy. Reprezentacja ta z założenia jest rozumiana przez ekspertów z danej dziedziny wiedzy. Autorzy prezentują system z bazą wiedzy oparty na grafach, będących zarówno reprezentacją pośredniczącą jak i formalną reprezentacją, na której przeprowadzane jest wnioskowanie. W przedstawionym modelu użyta jest logika pierwszego rzędu, jednak jest ona użyta tylko w celu formalnego ugruntowania reprezentacji grafowej i zdefiniowania semantyki grafów. Mechanizmy wnioskujące operują bezpośrednio na grafach. Według autorów takie rozwiązanie sprawia, że proces wnioskowania może być w całości odczytany przez użytkownika systemu. Jako przykład zastosowania tego modelu autorzy prezentują system przeszukujący bazę zdjęć, wzbogaconych dodatkowo o informacje semantyczne (Rysunek 2.13). Samo przeszukiwanie bazy zdjęć sprowadza się do wyszukiwania podgrafów w zbiorze grafów. W tym celu mechanizmy wyszukiwania korzystają z homomorfizmu grafów. W skład bazy wiedzy wchodzi ontologia, pozwalająca formułować ograniczenia i reguły, a także zapytania o obiekty szerszej klasy, nie tylko o konkretne wystąpienia.



Rysunek 2.13 Przykład dodawania semantycznej informacji dla zdjęcia w systemie Cheina [36]

System zaprezentowany w artykule Cheina jest podobny w swojej budowie do systemu HSSDR, jeżeli porównać je na pewnym poziomie abstrakcji. Operuję, podobnie jak HSSDR na grafach hierarchicznych, w których obecne są wieloargumentowe relacje. Pozwala przeszukiwać zbiór grafów, czyli sprawdzać konkretne grafy pod kątem ograniczeń zdefiniowanych za pomocą ontologii i reguł.

SpaceOntology

W pracy [74] autorzy przedstawiają oparte na ontologiach rozszerzenie systemu realizującego zadanie planowania, zastosowanego do planowania akcji robotów mobilnych działających w środowisku budynku. Autorzy, używając języka OWL DL, definiują ontologię zwana SpaceOntology. Rozszerzają oni język PDDL(Planning Domain Definition Language) dodając do niego informacje przestrzenne tworząc Spatial-PDDL. Podczas rozwiązywania problemu planowania sprawdzane jest zachodzenie określonych warunków z użyciem ontologii. System umożliwia również określanie relacji odległości w formie rozmytej.

Praca ta pokazuje możliwości zastosowania ontologii, w szczególności języka OWL DL w formułowaniu problemów na planie rozkładu pomieszczeń. Autorzy pomijają kwestie ekstrakcji wiedzy o planie budynku, zarówno opis świata jak i problemu wprowadzane są do systemu ręcznie w postaci definicji Spatial-PPDL oraz OWL-DL.

3. Komputerowe projektowanie i wnioskowanie z użyciem diagramów, grafowych struktur danych i formuł logicznych

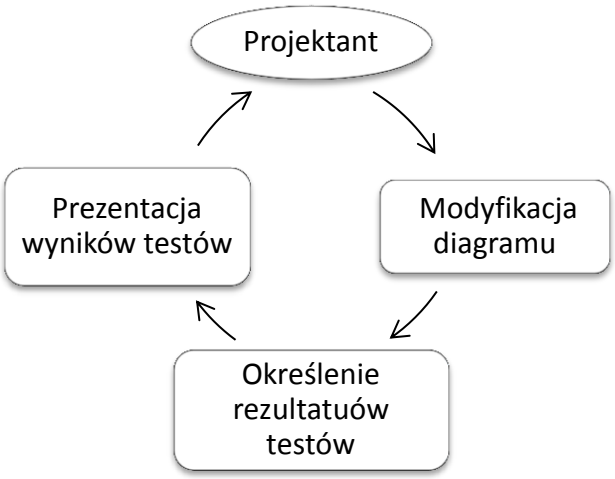
Projektowanie z użyciem systemu HSSDR można scharakteryzować, jako projektowanie od ogółu do szczegółu (ang. top-down). Komunikacja pomiędzy systemem a użytkownikiem odbywa się za pomocą języka wizualnego, którego elementami są proste diagramy projektowe, wzorowane na schematach rozkładów pomieszczeń używanych od dawna przez architektów. Użytkownik rozpoczyna od narysowania obrysu rozkładu pomieszczeń. Następnie może on dzielić wprowadzane pomieszczenia na mniejsze, nanosić dodatkowe elementy oraz cofać swoje akcje. Dodatkowo, przed rozpoczęciem projektowania użytkownik ma możliwość określenia rozmiaru dostępnego arkusza, a także określenia gęstości siatki.

Edytor rozkładów pomieszczeń HSSDR jest przykładem edytora diagramów stworzonego na potrzeby konkretnego zadania. Nie daje on możliwości tworzenia dowolnych rysunków, ale wymusza wprowadzanie kształtów będących elementami pewnego języka wizualnego. Dzięki temu możliwe jest tworzenie przez system wewnętrznej, grafowej reprezentacji projektu, a nie tylko przechowywanie zbioru prymitywów graficznych.

Od współczesnych narzędzi CAD oczekuje się wspierania procesu podejmowania decyzji. System wspomagania projektowania musi być wyposażony w wewnętrzne struktury danych, które będą w stanie przechowywać dane oraz udostępniać je na potrzeby automatycznego przetwarzania. W proponowanym systemie rolę struktury przechowującej wiedzę projektową pełnią *hierarchiczne atrybutowane hipergrafy*. Umożliwiają one zapis informacji o składowych projektu, ich właściwościach oraz o relacjach przestrzennych, w jakich się one znajdują. Informacje o topologii projektu oraz o fizycznych właściwościach jego składowych są dalej udostępniane na potrzeby automatycznego wnioskowania.

W zależności od rodzaju projektowanego obiektu projektant może na jednej z zakładek wybrać zestawy testów, które będą brane pod uwagę. Możliwe jest również tworzenie bibliotek własnych zestawów wymagań projektowych oraz dodawanie ograniczeń opisujących wymagania klienta dla konkretnego projektu.

Projektowanie z użyciem proponowanego systemu przybiera formę dialogu pomiędzy projektantem a środowiskiem CAD. Cykl ten został zilustrowany na rysunku Rysunek 3.1. Projektant, posługując się systemem, przeszukuje obszar potencjalnych rozwiązań, podczas gdy w tle przeprowadzane jest wnioskowanie oceniające aktualny stan projektu pod kątem zgodności z wymaganiami. Na podstawie odpowiedzi z systemu projektant może zauważyć potrzebę zmian lub założyć, że dany problem zostanie rozwiązany później.



Rysunek 3.1 Cykl projektowania w systemie HSSDR

Dalsza część bieżącego rozdziału wyjaśnia budowę i działanie sytemu HSSDR. Opisane zostały szczegółowo jego elementy składowe oraz związane z nimi pojęcia. Ostatni podrozdział zawiera przykłady testów zgodności pokazujące możliwości sprawdzania wymagań projektowych przez system.

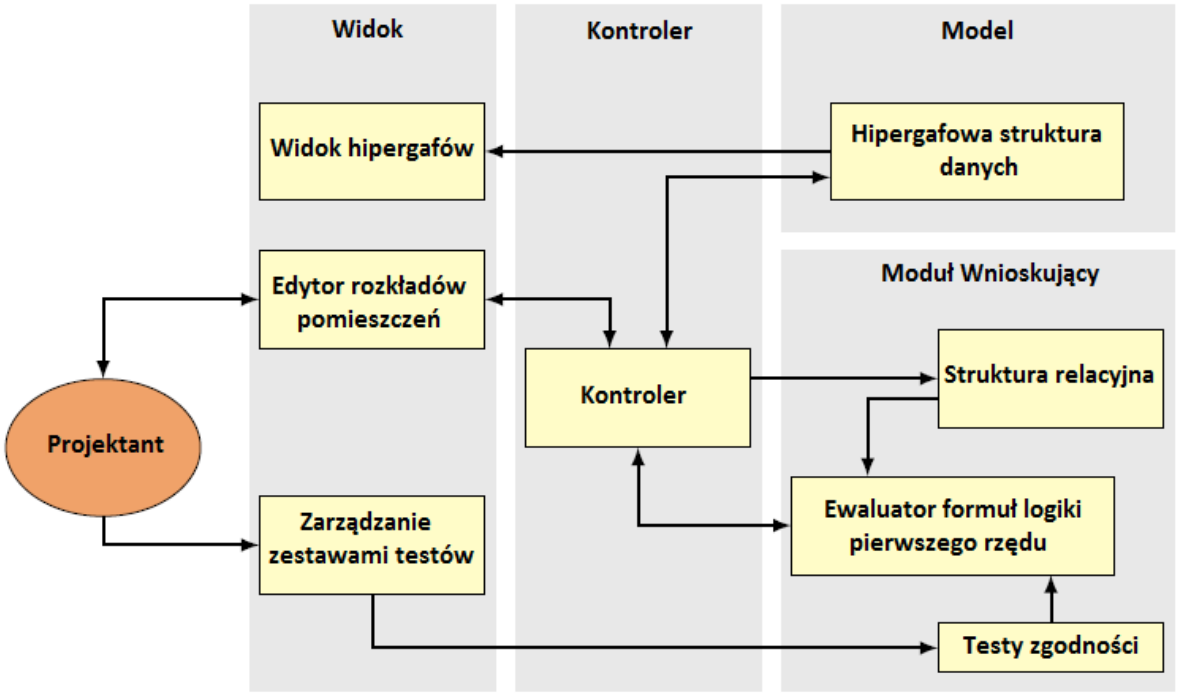
3.1. HSSDR – Prototypowe narzędzie CAD

W niniejszym podrozdziale zostały opisane założenia systemu HSSDR. Jest to prototypowe narzędzie wspomagania projektowania architektonicznego, którego mechanizmy przetwarzania wiedzy projektowej oparte są na strukturach grafowych. Koncentruje się ono na wspomaganiu fazy koncepcyjnej projektowania. System ten został stworzony przez autora niniejszej pracy w latach 2008-2015. Jego założenia tworzone były w ramach badań autora oraz członków zespołu Zakładu Projektowania i Grafiki Komputerowej na Wydziale Fizyki, Astronomii i Informatyki Stosowanej Uniwersytetu Jagiellońskiego kierowanego przez prof. dr hab. Ewę Grabską [75] [76] [77] [78] [79]. Parser języka testów został zaprogramowany przez dr Wojciecha Palacza. System był również konsultowany z dr inż. arch. Agnieszką Ozimek oraz dr inż. arch. Pawłem Ozimek, architektami pracującymi na Politechnice Krakowskiej. Pierwsze wersje HSSDR umożliwiały wnioskowanie o dostosowaniu planu budynku do potrzeb robotów mobilnych [77]. Na tym etapie prac sprawdzane warunki były zaszyte w kodzie programu.

System podejmuje tematykę badaną wcześniej w ramach grantu na Uniwersytecie Jagiellońskim [80]. Próbą wcześniejszej realizacji systemu wspomagającego projektowanie był system opisanym w [81]. Umożliwiał on edytowanie prostych rozkładów pomieszczeń

oraz podgląd odpowiadających im hipergrafów. Inne podejście zostało opisane w artykule [82]. Jego efektem był system umożliwiający edycję rozkładów pomieszczeń, dodatkowo wyposażony w możliwość wyszukiwania pewnych podgrafów w utworzonym hipergrafie. Powyższe próby ograniczały się jednak głównie do uproszczonej edycji rozkładów pomieszczeń oraz automatycznego generowania pewnych struktur wewnętrznych, nie umożliwiały one sprawdzania wymagań projektowych.

HSSDR został zaimplementowany w języku Java. Interfejs graficzny systemu zbudowano z wykorzystaniem biblioteki Swing. Do celów generacji parsera języka testów opisanego w podrozdziale 3.5 używana jest biblioteka ANTLR [83]. Architektura systemu powstała na podstawie wzorca projektowego MVC (ang. model-view-controller). Wzorzec ten pozwala na projektowanie aplikacji w rozszerzalny, elastyczny sposób, dzięki logicznemu odseparowaniu części aplikacji odpowiedzialnych za komunikację z użytkownikiem (Widok), od tych odpowiedzialnych za wewnętrzną logikę programu (Model), za pomocą warstwy pośredniczącej (Kontoler). Na rysunku Rysunek 3.2 przedstawiona jest architektura systemu.



Rysunek 3.2 Architektura sytemu HSSDR

HSSDR posiada graficzny interfejs użytkownika składający się z komponentów bloku Widok: edytora rozkładów pomieszczeń, zakładek wyboru lub edycji zestawów testów oraz podglądu hipergrafów. Za wewnętrzną reprezentację projektu odpowiada hipergrafowa struktura danych. Kolejne moduły są odpowiedzialne za wnioskowanie o tworzonym projekcie. Blok testów zgodności reprezentuje zbiór plików zawierających wymagania definiowane w języku logiki pierwszego rzędu. Ewaluator formuł logiki pierwszego rzędu posiada parser języka testów HSSDR, który dokonuje analizy leksykalnej i syntaktycznej zawartości zestawów testów oraz buduje reprezentacje drzewową formuł. Wartościowanie

formuł przeprowadzane jest w kontekście struktury relacyjnej, która opisuje aktualny stan projektu. Struktura relacyjna zasilana jest danymi zawartymi w hipergrafie hierarchicznym, a także wartościami funkcji i relacji niezawartymi bezpośrednio w opisie projektu, obliczonymi na jego podstawie przez system na potrzeby wnioskowania.

System HSSDR jest systemem opartym na wiedzy. Można go zaliczyć do systemów symbolicznych, deklaratywnych oraz bazujących na zastosowaniu logiki, w szczególności logiki pierwszego rzędu, rozważanej w podrozdziale 3.5.1. Porównując schemat organizacji systemów z bazą wiedzy przedstawiony w podrozdziale 2.4.2 z architekturą systemu HSSDR można określić następujące przyporządkowanie:

- Baza wiedzy – testy zgodności,
- Mechanizmy wnioskowania – ewaluator formuł logiki pierwszego rzędu, struktura relacyjna, hipergrafowa struktura danych,
- Moduł objaśnień – ewaluator formuł logiki pierwszego rzędu,
- Moduł sterujący – kontroler,
- Interfejs wejścia/wyjścia – edytor rozkładów pomieszczeń.

3.2. Diagramy projektowe

Percepcja i wnioskowanie wizualne odgrywają ważną role w procesie projektowania stosowanym w wielu dziedzinach, także niepokrewnych architekturze. We współczesnym świecie ten proces realizowany jest bardzo często przy użyciu komputerów. Nośnikiem wizualizacji w tym środowisku są zwykle diagramy projektowe (Definicja 3.4). Przykładem ich zastosowania niech będzie język UML, służący do obrazowania struktury nowo tworzonych systemów informatycznych czy modelowania procesów biznesowych [84]. Kolejnymi przykładami ich użycia są wizualne języki programowania (np. KTechLab), diagramowe meta języki używane w obróbce muzyki, obrazów, danych, a także wszelkie nieformalne schematy, rysunki i szkice. Te artefakty pobudzają kreatywność oraz umożliwiają wymianę idei i zapis informacji w łatwo przyswajalnej dla człowieka formie.

Projektowanie, w którym kluczową rolę odrywa wizualizacja z użyciem diagramów jest projektowaniem diagramowym. Autorzy pracy [85] wyróżniają dwie klasy edytorów diagramów: otwarte (ang. free-hand) oraz sterowane składniowo (ang. syntax-directed). Edytory otwarte udostępniają użytkownikowi zbiór prymitywów graficznych oraz pozwalają na bezpośrednie tworzenie diagramu z ich użyciem. W celu sprawdzenia poprawności oraz przeprowadzenia analizy syntaktycznej takiego diagramu konieczne jest jego parsowanie. Natomiast edytory sterowane składniowo tworzone są za pomocą predefiniowanego zbioru operacji. Ich wewnętrza, najczęściej grafowa, reprezentacja modyfikowana jest po każdej operacji.

Projektowanie diagramowe zostało szczegółowo opisane w pracy „*Projektowanie wizualne wspomagane komputerem*” [86]. Z tej monografii pochodzi definicja diagramu projektowego zaprezentowana poniżej. Jest on definiowany na podstawie następujących pojęć: przekształcenia dopuszczalnego, alfabetu figur podstawowych oraz konfiguracji przestrzennej.

Definicja 3.1 Przekształcenia dopuszczalne

Grupę przekształceń przestrzeni $\mathbb{R}^n$ w siebie przekształcających podzbiory ograniczone na podzbiory ograniczone nazywamy *przekształceniami dopuszczalnymi*.

Definicja 3.2 Alfabet figur podstawowych

Skończony podzbiorów S ograniczonych podzbiorów $\mathbb{R}^n$ jest *alfabetem figur podstawowych* dla grupy przekształceń dopuszczalnych F jeśli:

$$\forall f \in F \ \ \forall s, p \ \in S\text{:}$$

$$(\ f(s) = p\ ) \ \Rightarrow (\ s = p\ )$$

Alfabet figur podstawowych jest zbiorem kształtów, które podlegając przekształceniom ze zbioru przekształceń dopuszczalnych stanowią elementy składowe diagramu. Przekształcenia dopuszczalne i alfabet figur podstawowych są ze sobą powiązane, im liczniejszy jest alfabet figur podstawowych tym potrzebna jest mniej liczna grupa przekształceń dopuszczalnych i vice versa.

Definicja 3.3 Konfiguracja przestrzenna

Dla alfabetu figur podstawowych S, oraz grupy przekształceń dopuszczalnych F, *konfiguracją przestrzenną* nazywamy skończony, niepusty zbiór Q, taki, że:

$$Q \ \subseteq S \times F$$

Konfiguracja przestrzenna jest zbiorem par (s, f) determinujących wszystkie składowe diagramu postaci $f(s)$. Na podstawie konfiguracji przestrzennej definiuje się pojęcie diagramu projektowego.

Definicja 3.4 Diagram projektowy

Niech Q będzie konfiguracją przestrzenną określoną dla alfabetu figur podstawowych S, oraz grupy przekształceń dopuszczalnych F.

Diagram projektowy $Z(Q)$ z konfiguracją Q jest sumą składowych z konfiguracji Q:

$$Z(Q) \ = \bigcup_{(s,f)\in\, Q} f(s)$$

3.2.1. Diagramy projektowe HSSDR

Diagramy projektowe systemu HSSDR są elementem procesu projektowania opisanego w rozprawie. Używane tutaj do reprezentowania rozkładów pomieszczeń diagramy są uproszczeniem stosowanych powszechnie w architekturze klasycznych diagramów rozkładów pomieszczeń. W sensie powyższych definicji diagramy opisywanego systemu są diagramami projektowymi. System HSSDR jest edytorem diagramów sterowanym składniowo. Pozwala on użytkownikowi na wprowadzenie tylko poprawnych diagramów.

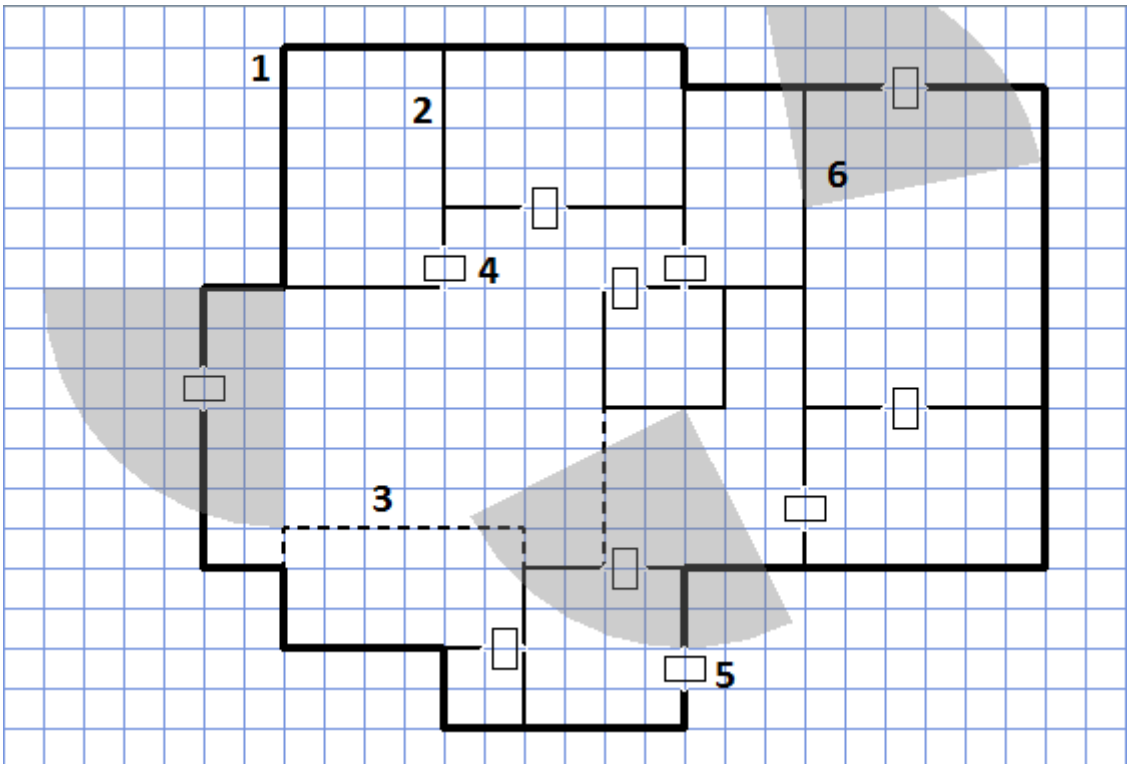
Składowe rozkładów pomieszczeń reprezentowane są podczas projektowania przez diagramy składające się z wielokątów. W końcowym etapie projektu wielokąty reprezentują pomieszczenia, jednak w trakcie projektowania mogą też reprezentować większe sekcje budynku, niepodzielone jeszcze na korytarze i pokoje. Jeżeli dwa wielokąty mają wspólny bok, odpowiadające im pomieszczenia będą miały wspólną ścianę. Dla systemu istotne są relacje dostępności pomiędzy pomieszczeniami, dlatego jest on wyposażony w możliwość zaznaczenia lokalizacji drzwi. Są one reprezentowane przez element wizualny w postaci małego prostokąta, który może być umieszczony na boku wielokąta. W systemie występują też linie przerywane, reprezentują one w zależności od wybranej konfiguracji:

- oddzielenie od siebie pomieszczeń w sensie logicznym, bez oddzielenia ich fizycznie, np. kuchnia połączona z salonem.

- relacje widoczności pomiędzy pomieszczeniami, np. biuro ze szklaną ścianą. W przypadku takiej krawędzi może ona zawierać również symbol drzwi, żeby oznaczyć dostępność.

Na rysunku Rysunek 3.3 znajduje się przykład diagramu projektowego HSSDR z ponumerowanymi elementami składowymi, są to odpowiednio:

1. Linia obrysu planu (grubsza).
2. Linia podziału (cieńsza).
3. Przerywana linia podziału.
4. Symbol wizualny drzwi.
5. Symbol wizualny drzwi zewnętrznych.
6. Czujnik bezpieczeństwa.

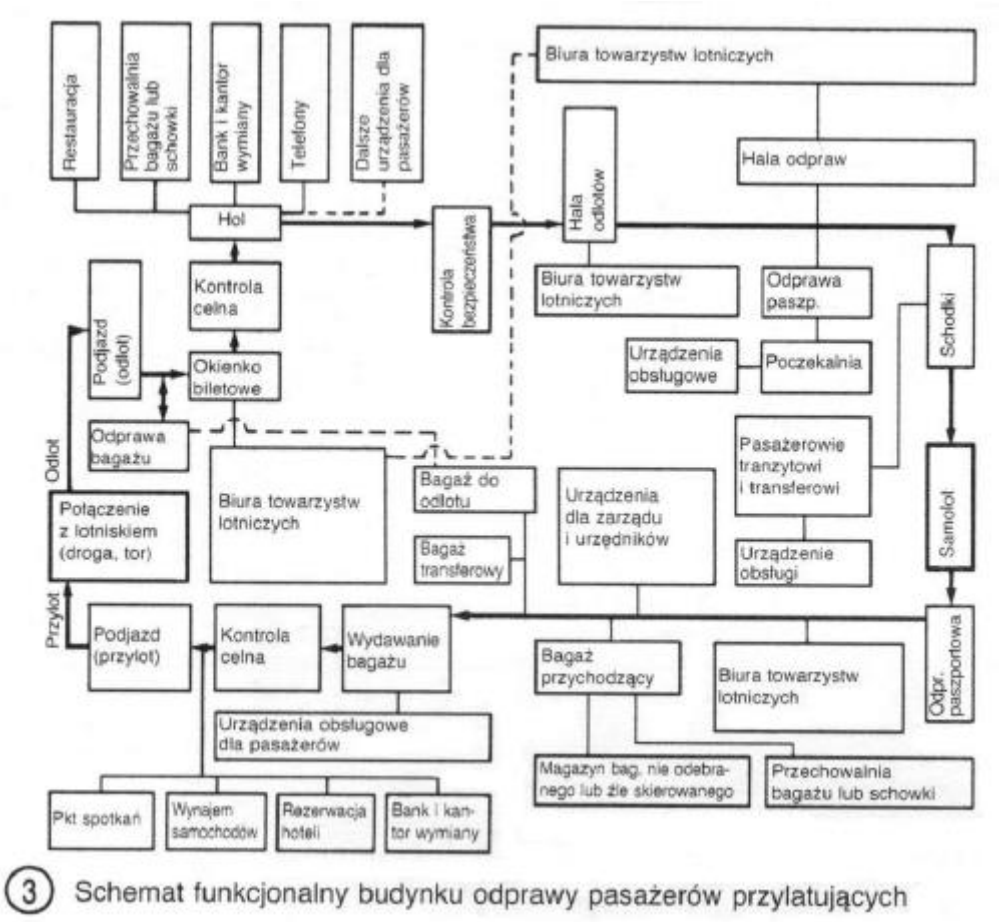


Rysunek 3.3 Elementy składowe diagramów projektowych HSSDR

3.3. *Grafowe struktury danych – hipergrafy*

Aby system wspomagania projektowania był w stanie przechowywać dane projektu oraz udostępniać je na potrzeby automatycznego przetwarzania musi być on wyposażony w wewnętrzne struktury danych. W proponowanym systemie rolę struktury przechowującej wiedzę projektową pełnią *hierarchiczne atrybutowane hipergrafy*. Umożliwiają one zapis informacji o składowych projektu, ich właściwościach oraz o relacjach przestrzennych w jakich się one znajdują. Informacje te są dalej udostępniane na potrzeby automatycznego wnioskowania.

Grafy są znane i stosowane wśród architektów. Przykładem ich zastosowania niech będzie schemat funkcjonalny pochodzący z klasycznego podręcznika do projektowania architektonicznego E. Neuferta [87]. Rysunek 3.4 przedstawia graf skierowany, reprezentujący schemat budynku odprawy pasażerów przylatujących, pochodzący z rozdziału o projektowaniu lotnisk.



3 Schemat funkcjonalny budynku odprawy pasażerów przylatujących

Rysunek 3.4 Graf skierowany reprezentujący schemat funkcjonalny fragmentu lotniska. Schemat pochodzi podręcznika projektowania architektonicznego E. Neuferta.

Struktury grafowe są również często używane w systemach wspomagających projektowanie, przykładami niech będą prace: [1] [7] [36] [88] [85] [89] [90] [91] [92] [93].

Hipergrafy są rozszerzeniem grafów (graf można interpretować jako szczególny przypadek hipergrafu, w którym wszystkie hiperkrawędzie są incydentne do co najwyżej dwóch wierzchołków). W rozprawie doktorskiej Annegret Habel [94] zostały zaprezentowane hipergrafy oraz idea zastępowania hiperkrawędzi przez hipergrafy na wzór produkcji w językach formalnych. Definicja hipergrafu używana w niniejszej pracy różni się w kilku punktach od klasycznej definicji, jaką można znaleźć na przykład w pracy „Handbook of graph grammars and computing by graph transformation” [95]. Oprócz rozszerzeń polegających na dodaniu atrybutów oraz struktury hierarchicznej, istotną modyfikacją jest rozróżnienie hiperkrawędzi obiektowych oraz relacyjnych w zbiorze hiperkrawędzi. Kolejną modyfikacją jest użycie tutaj hipergrafu skierowanego, chociaż ta cecha nie jest na chwilę obecną wykorzystywana w systemie.

W prezentowanej strukturze hipergrafowej możemy wyróżnić trzy kategorie obiektów: hiperkrawędzie relacyjne, hiperkrawędzie obiektowe oraz wierzchołki. Hiperkrawędzie dla

uproszczenia będą również nazywane krawędziami. Hiperkrawędzie relacyjne odpowiadają relacjom pomiędzy składowymi projektu. Te z kolei są reprezentowane przez hiperkrawędzie obiektowe (pomieszczenia, grupy pomieszczeń, piętra) bądź wierzchołki, w przypadku fragmentów składowych łączących się z innymi składowymi (ściany pomieszczeń). Na potrzeby przechowywania informacji o ich fizycznych właściwościach użyta jest funkcja atrybutowania. Atrybutami są rozmiar czy pozycja składowych projektu, ale też typ relacji pomiędzy nimi.

Hipergrafy hierarchiczne, w porównaniu z prostymi strukturami grafowymi mają tę zaletę, że pozwalają operować fragmentami projektu (hiperkrawędzią wraz z jej zewnętrznymi wierzchołkami) tak samo jak całym projektem. Hiperkrawędź reprezentująca dany obiekt, zawiera w sobie hipergraf będący reprezentacją jej składowych. Hierarchiczność pozwala również na odzwierciedlenie natury projektowania metodą od ogółu do szczegółu: hierarchiczna struktura hipergrafu jest zapisem kolejnych akcji wykonanych podczas tworzenia projektu. Umożliwia ona również wnioskowanie o logicznej strukturze projektu poprzez użycie warunków sprawdzających zagnieżdżanie składowych projektu w drzewie hierarchii (Rozdział 3.7.1). Taka możliwość pozwala również uprościć proces wnioskowania o projekcie poprzez zawężenie analizy do jego fragmentu.

Użycie hiperkrawędzi do zapisu relacji między składowymi projektu, pozwala na włączenie do modelu relacji wieloargumentowych, co nie jest możliwe w przypadku reprezentacji relacji jako krawędzi w grafie. Ten cel jest również osiągalny dzięki grafom stosowanym na przykład w pracy [36], gdzie w miejsce hiperkrawędzi relacyjnych używane są wierzchołki relacyjne (ang. *relation nodes*). W przyjętym rozwiązaniu wierzchołki mają natomiast inną funkcję, pełnią one rolę wyodrębnionych fragmentów projektu, które są punktami zaczepienia relacji. Definiowanie relacji pomiędzy wyodrębnionymi fragmentami obiektów, a nie całymi obiektami, umożliwia dokładniejszą analizę projektu pod względem geometrycznym. Prezentowane podejście łączy więc możliwość zapisu wieloargumentowych relacji wraz z zaletami operowania wydzielonymi fragmentami projektu jako punktami zaczepienia relacji.

Interesujące zastosowanie hipergrafów, jako wewnętrznej struktury danych w systemach wspomagania projektowania przedstawiono w artykule „Concepts and realization of a diagram editor generator based on hypergraph transformation” [85]. Opisane jest w nim środowisko generowana edytorów diagramów DIAGEN. Generowane edytory używają hipergrafu jako wewnętrznego modelu diagramu, którego analiza składniowa jest przeprowadzana przez parsery hipergrafów. W otrzymanych edytorach możliwe jest również generowanie diagramów za pomocą produkcji gramatyk hipergrafowych. W artykule tym zostały wyróżnione dwa rodzaje hiperkrawędzi: komponentowe (ang. *component edges*) oraz relacyjne (ang. *relation edges)*. W artykule „Hierarchical Hypergraph Transformations in Engineering Design” [90] podobne hipergrafy zostały użyte do reprezentacji struktury obwodów elektrycznych.

Zastosowanie hipergrafów do reprezentacji rozkładów pomieszczeń zostało po raz pierwszy zaproponowane w pracach „Hypergraphs in Diagrammatic Design” [96] oraz „Hierarchical Layout Hypergraph Operations and Diagrammatic Reasoning” [97].

Jako narzędzie do definiowania hipergrafów oraz ich transformacji grafowych bywa używana teoria kategorii. To podejście zostało zaproponowane w rozprawie doktorskiej [89]. W monografii [86] właśnie podejście kategorialne jest użyte do zdefiniowania hipergrafu planu oraz jego gramatyki grafowej. Produkcje grafowe są definiowane za pomocą morfizmów. W niniejszej pracy gramatyki grafowe nie są stosowane, a do definicji hipergrafu użyty jest klasyczny opis teoriomnogościowy.

W następnym podrozdziale przedstawiona jest definicja hierarchicznego atrybutowanego hipergrafu, używana w systemie HSSDR. Wprowadzona definicja ta jest rozszerzeniem formalizmu z prac [97] oraz [86].

3.3.1. Definicja hierarchicznego atrybutowanego hipergrafu

Poniższe definicje są definicjami pomocniczymi dla definicji Definicja 3.5:

- Niech Σ_G^C oznacza alfabet etykiet hiperkrawędzi obiektowych, a Σ_G^R alfabet etykiet hiperkrawędzi relacyjnych.
- Niech $\Sigma_G^E = \Sigma_G^C \cup \Sigma_G^R$.
- Niech Σ_G^V będzie alfabetem etykiet wierzchołków.
- Niech At_G będzie zbiorem możliwych wartości atrybutów.
- Niech $[i]$, dla $i \in \mathbb{N}$, oznacza przedział początkowy $[1, i]$ liczb naturalnych.

Definicja 3.5 Hierarchiczny atrybutowany hipergraf

Hierarchiczny atrybutowany hipergraf (zwany również hipergrafem planu, bądź w skrócie hipergrafem) nad alfabetem Σ to:

$$G = (E_G, V_G, s_G, t_G, lb_G, att_G, ext_G, ch_G),$$

gdzie:

- E_G jest niepustym, skończonym zbiorem hiperkrawędzi będącym sumą zbiorów hiperkrawędzi obiektowych (E_G^C) oraz relacyjnych (E_G^R):

$$E_G = E_G^C \cup E_G^R, \quad E_G^C \cap E_G^R = \emptyset,$$

- V_G jest niepustym, skończonym zbiorem wierzchołków,

- $s_G: E_G \rightarrow (V_G)^*$ oraz $t_G: E_G \rightarrow (V_G)^*$ są odwzorowaniami przyporządkowującymi każdej hiperkrawędzi ciągi wierzchołków źródłowych oraz docelowych w taki sposób, że:

$$\forall e \in E_G^C: s_G(e) = t_G(e)$$

(ze skierowaniem grafu możemy mieć do czynienia tylko w przypadku hiperkrawędzi relacyjnych),

- zbiór etykiet lb_G jest sumą zbiorów etykiet hiperkrawędzi i wierzchołków: $lb_G = lb_G^E \cup lb_G^V$, gdzie:
  1. $lb_G^E : E_G \rightarrow \Sigma_G^E$ jest funkcją etykietowania hiperkrawędzi, w taki sposób, że:

$$\forall e \in E_G^C: lb_G(e) \in \Sigma_G^C, \; oraz \; \forall e \in E_G^R \; lb_G(e) \in \Sigma_G^R ,$$

  2. $lb_G^V : V_G \rightarrow \Sigma_G^V$ jest funkcją etykietowania wierzchołków,
- $att_G = att_G^E \cup att_G^V$ jest zbiorem atrybutów, gdzie:
  1. $att_G^E : \{ f_1, \dots, f_n , gdzie \; f_i : E_G \rightarrow At_G, dla \; 1 \leq i \leq n \}$ jest rodziną funkcji atrybutowania hiperkrawędzi, oraz
  2. $att_G^V : \{ f_1, \dots, f_m , gdzie \; f_i : V_G \rightarrow At_G, dla \; 1 \leq i \leq m \}$ jest rodziną funkcji atrybutowania wierzchołków,
- ext_G określa ciąg wierzchołków zewnętrznych, $ext_G: [k] \rightarrow V_G$, $k \in \mathbb{N}$,
- ch_G jest funkcją zagnieżdżania:

$$ch_G: E_G^C \rightarrow P(E_G \cup V_G),$$

która spełnia następujące warunki:

1. jest acykliczna, tzn.:

$$\forall e \in E_G^C: e \notin ch_G^+(e),$$

gdzie $ch_G^+(e)$ oznacza wszystkich potomków danej krawędzi,

2. hiperkrawędź lub wierzchołek nie może być zagnieżdżony w dwóch różnych rodzicach, tzn.:

$$\forall a \in E_G \cup V_G, \forall e_1, e_2 \in E_G^C:$$
$$a \in ch_G(e_1) \wedge a \in ch_G(e_2) \Rightarrow e_1 = e_2,$$

3. hiperkrawędź obiektowa oraz jej wszystkie wierzchołki źródłowe i docelowe muszą mieć tego samego rodzica, tzn.:

$$\forall\, e_1, e_2 \in E_G^C\colon\ e_1\ \in\ ch_G(e_2) \Rightarrow t(e_1) \subseteq ch_G(e_2),$$

4. hiperkrawędzie obiektowe oraz ich wierzchołki muszą być zagnieżdżone w tej samej hiperkrawędzi:

$$\forall\, e \in E_G^C, \forall\, v \in V_G\colon$$

$$v\ \in ch_G(e) \Rightarrow \exists e_s \in\ ch_G(e)\colon v \in t(e_s).$$

Poniżej znajduję się definicja atrybutów, czyli lista funkcji atrybutowań wchodzących w skład zbioru att_G. Używane atrybuty zależą od konkretnego zastosowania hipergrafu i ich lista może się zmieniać wraz z dziedziną, w obrębie której stosowany jest hipergraf.

Definicja 3.6 Atrybuty hipergrafu planu

W skład zbioru funkcji atrybutowania att_G wchodzą:

- $area_G\colon E_G^C \to \mathbb{R}$, dla hiperkrawędzi obiektowej, powierzchnia pomieszczenia reprezentowanego przez tą hiperkrawędź,
- $type_G\colon E_G^C \to \{$„Space", $"Kitchen"$, $"Corridor"$, $"Staircase"$, $"Office"$, $"Empty"$, …$\}$, dla hiperkrawędzi obiektowej, typ pomieszczenia reprezentowanego przez tą hiperkrawędź,
- $dir_G\colon V_G \to \{"N","S","W","E"\}$, orientacja ściany reprezentowanej przez dany wierzchołek,
- $length_G\colon V_G \to \mathbb{R}$, długość ściany,
- $coord1_G\colon V_G \to \mathbb{R} \times \mathbb{R}$, współrzędne pierwszego z końców odcinka tworzącego ścianę,
- $coord2_G\colon V_G \to \mathbb{R} \times \mathbb{R}$, współrzędne drugiego z końców odcinka tworzącego ścianę,
- $wallNo_G\colon V_G \to \mathbb{N}$, numeracja wierzchołków połączonych z hiperkrawędzią obiektową,
- $doorCount_G\colon V_G \to \mathbb{N}$, liczba drzwi istniejących dla ściany reprezentowanej przez dany wierzchołek,
- $doorsAttributes_G\colon V_G \to P(\mathbb{R}^5)$, zbiór atrybutów drzwi istniejących dla ściany reprezentowanej przez dany wierzchołek. Atrybuty to $(a_x, a_y, b_x, b_y, w) \in \mathbb{R}^5$, gdzie

a_x, a_y, b_x, b_y to współrzędne końców odcinka reprezentującego drzwi, a w to szerokość drzwi.

- $sensors_G$: $E_G^C \rightarrow P(\mathbb{R}^3)$, dla hiperkrawędzi obiektowej, zbiór czujników bezpieczeństwa znajdujących się w pomieszczeniu reprezentowanym przez daną krawędź. Czujnik bezpieczeństwa jest zdefiniowany jest jako $s = (s_x, s_y, \alpha)$, $s \in \mathbb{R}^3$, gdzie (s_x, s_y) to współrzędne określające lokalizację czujnika, α to kąt skierowania czujnika (podrozdział 3.7.3).

3.3.2. Przykład hipergrafu planu

Niech do zilustrowania powyższych definicji posłuży przykładowy rozkład pomieszczeń wraz z odpowiadającym mu hipergrafem planu. Rozważmy rozkład pomieszczeń przedstawiony na rysunkuRysunek 3.5. Przedstawia on plan dwupokojowego mieszkania o powierzchni 50 metrów kwadratowych. Odpowiadający mu hipergraf G przedstawiony jest z kolei na rysunku Rysunek 3.6. Poniżej rozpisane są wybrane jego elementy.

Zbiory hiperkrawędzi obiektowych oraz relacyjnych hipergrafu G to odpowiednio:

- $E_G^C = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$,
- $E_G^R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}$.

Zbiór wierzchołków jest zdefiniowany, jako:

- $V_G = \{n_1, n_2, \dots, n_{29}\}$,

natomiast ciąg wierzchołków zewnętrznych to:

- $ext_G = (n_1, n_2, \dots, n_{13})$.

Ciągi wierzchołków źródłowych oraz docelowych są określone dla każdej hiperkrawędzi, przykładowo:

- $s_G(e_2) = t_G(e_2) = (n_1, n_{10}, n_{13}, n_{14}, n_{15})$.

Dla poniższych hiperkrawędzi i relacji funkcja etykietowania przybiera następujące wartości:

- $lb_G(e_2) = "Kuchnia"$,
- $lb_G(e_3) = "Pokój\ dzienny"$,
- $lb_G(r_1) = "acc"$,

- $lb_G(r_6) = "adj"$,

gdzie etykieta „acc" definiuje relację dostępności, natomiast etykieta „adj" relacje przylegania. Funkcją zagnieżdżania jest określona w następujący sposób:

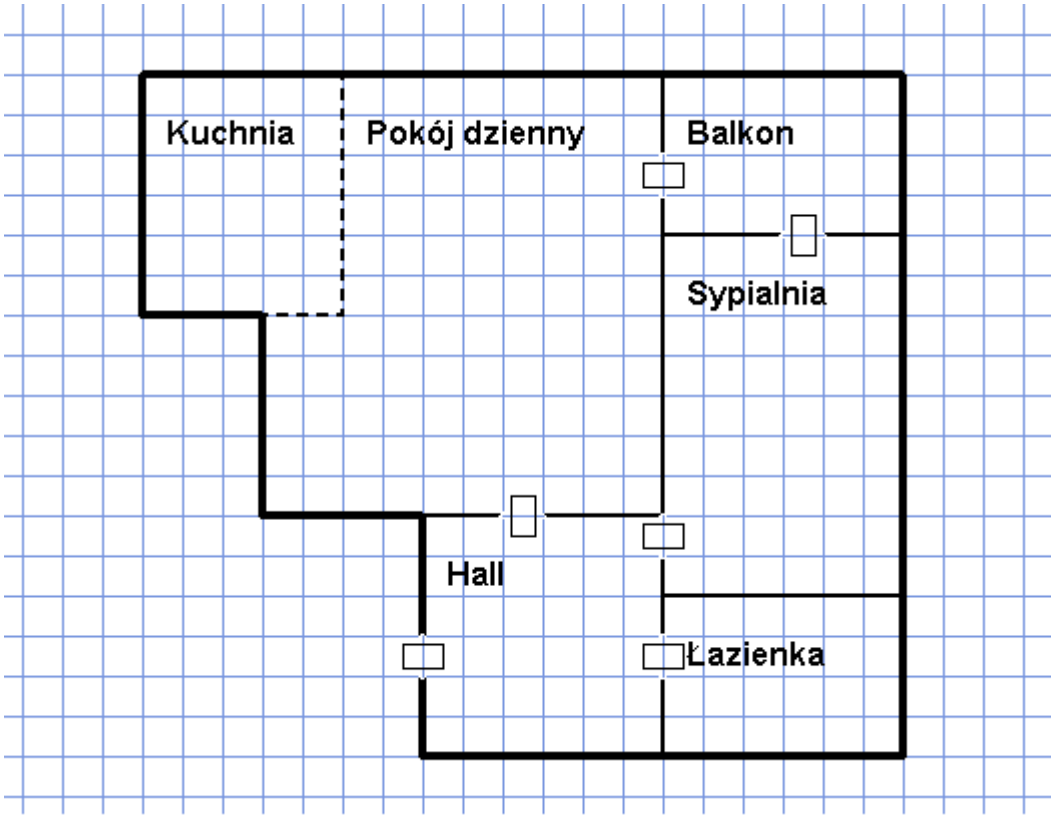
- $ch_G(e_1) = \{e_2, e_3, e_4, e_5, e_6, e_7\}$,
- $ch_G(e_2) = ch_G(e_3) = ch_G(e_4) = ch_G(e_5) = ch_G(e_6) = ch_G(e_7) = \emptyset$.

Poniżej przytoczone są wybrane wartości funkcji atrybutowań. I tak, dla hiperkrawędzi e_6 będą to:

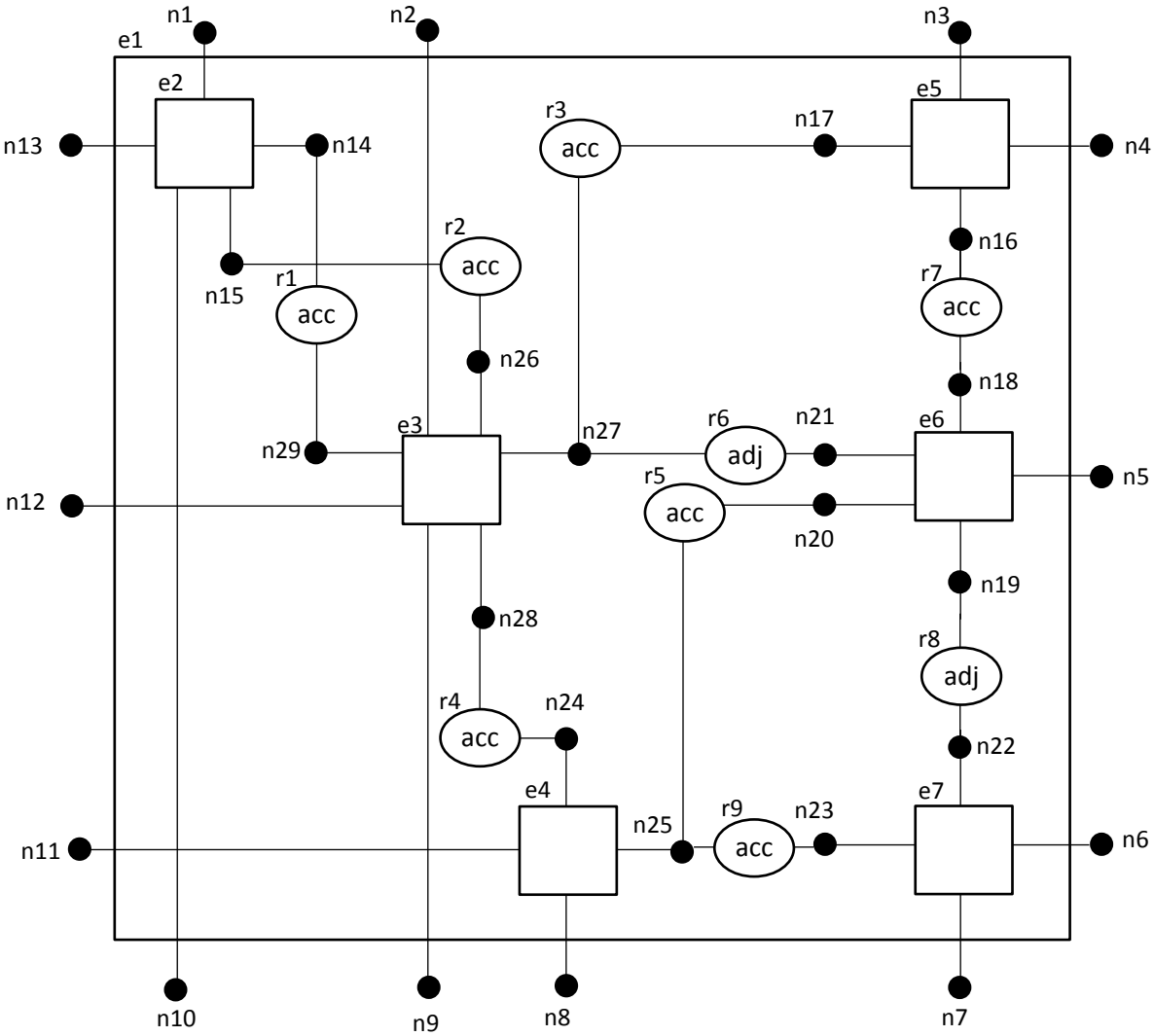
- $type_G(e_6) = "Bedroom"$,
- $area_G(e_6) = 13{,}5$.

Natomiast dla wierzchołka n_{18}:

- $dir_G(n_{18}) = "N"$,
- $length_G(n_{18}) = 3$,
- $wallNo_G(n_{18}) = 0$,
- $doorCount_G(n_{18}) = 1$.



Rysunek 3.5 Przykładowy rozkład pomieszczeń



Rysunek 3.6 Hipergraf *G* odpowiadający rozkładowi pomieszczeń z rysunku 3.5.

3.4. Ekstrakcja i zapis wiedzy o rozkładzie pomieszczeń

W poprzednich podrozdziałach zostały omówione diagramy projektowe, wprowadzona została również definicja hipergrafu. Niniejszy rozdział opisuje algorytm automatycznej zamiany diagramów projektowych reprezentujących rozkłady pomieszczeń na odpowiadające im hipergrafy, które stanowią wewnętrzną strukturę danych systemu.

Modyfikacja wewnętrznej reprezentacji danych następuje po każdej akcji użytkownika. Może on zmieniać projekt za pomocą szeregu opcji udostępnianych przez interfejs. Każda ze zmian projektu będzie pociągała za sobą pewną modyfikację struktury hipergrafowej. Interfejs HSSDR udostępnia użytkownikowi tylko poprawne w danym momencie akcje. Zapewnia on między innymi, że pierwszą akcją na projekcie będzie zawsze stworzenie obrysu

planu, oraz że obrys ten będzie łamaną zamkniętą. Poniżej opisane są kroki realizowane przez system po wykonaniu akcji przez użytkownika.

3.4.1. Definicje pomocnicze

Definicja 3.7 Łamana

Przez łamaną rozumiemy tutaj ciąg odcinków w którym koniec poprzedniego jest początkiem następnego, poza tymi punktami odcinki nie mogą się przecinać. Łamana oznaczana jest jako ciąg punktów $A_1, \dots, A_n$, gdzie A_1 to początek piewszego odcinka, A_n to koniec ostatniego.

Definicja 3.8 Łamana zamknięta

Łamana zamknięta to łamana $A_1, \dots, A_n$, w której $A_1 = A_n$.

Definicja 3.9 Łamana w postaci standardowej

Łamana w postaci standardowej jest łamaną zamkniętą, numerowaną zgodnie z ruchem wskazówek zegara, gdzie pierwszy element w ciągu jest odcinkiem wysuniętym najbardziej na zachód z odcinków wysuniętych najbardziej na północ

3.4.2. Algorytm tworzenia struktur hipergrafowych

Poniżej opisane są kroki algorytmu tworzenia wewnętrznej reprezentacji projektu realizowane po poszczególnych krokach tworzenia rozkładu pomieszczeń.

Stworzenie obrysu planu

1. Stworzenie pustego obiektu hipergrafu, inicjalizacja struktur pomocniczych oraz parametrów.
2. Na podstawie łamanej wprowadzonej przez użytkownika do grafu zostaje dodana hiperkrawędź obiektowa. Realizowane są następujące kroki:
    1. Tworzona jest hiperkrawędź obiektowa reprezentująca obrys.
    2. Łamana wprowadzona przez użytkownika sprowadzana jest do postaci standardowej.

3. Dla każdego odcinka łamanej tworzony jest wierzchołek przylegający do nowo utworzonej hiperkrawędzi w grafie. Etykietą wierzchołka jest numer odcinka w ciągu tworzącym łamaną (numeracja zaczyna się od 1).
4. Na podstawie analizy wnętrza obrysu obliczane są współrzędne środka ciężkości figury, oraz maksymalne wymiary w poziomie i pionie. (Dane te używane są w celu prezentacji hipergrafów).

Podział pomieszczenia.

Z edytora przekazywane są:

- łamana $A = A_1, \dots, A_n$ w postaci standardowej będąca dzielonym pomieszczeniem, reprezentowanym przez hiperkrawędź e_A,

- łamana $B = B_1, \dots, B_m$ będąca linią nowo tworzonego podziału.

Interfejs HSSDR zapewnia, że końce łamanej B znajdują się na łamanej A. Realizowane są następujące kroki:

1. Wyszukiwane są punkty przecięcia łamanej A z B.
    a. Punkt przecięcia s może znajdować się na końcach odcinka wchodzącego w skład łamanej, lub poza nimi. Jeżeli leży on na odcinku $(A_k,\ A_{k+1})$ lecz poza jego końcami, ta składowa łamanej dzielona jest na dwa odcinki: (A_k, s) oraz $(s,\ A_{k+1})$.
2. Tworzone są łamane C oraz D odpowiadające obrysom pomieszczeń powstałych po dokonaniu podziału. Oznaczmy punkty przecięcia łamanej A z B jako p i q. Po operacji z kroku 1a wiadomo, że oba punkty wchodzą w skład ciągu punktów tworzących końce odcinków łamanej A, oznaczmy te odcinki jako A_p oraz A_q. Niech p będzie punktem, który posiada niższy indeks w ciągu. Łamane C i D będą tworzone przez następujące ciągi:
    - $C : B_1, \dots, B_m, A_q, \dots, A_n, A_1, \dots, A_p$
    - $D :\ B_1, \dots, B_m, A_q, \dots, A_p$
3. Dla łamanych C i D tworzone są hiperkrawędzie e_C oraz e_D:
    a. Tworzona jest hiperkrawędź obiektowa reprezentująca łamaną.
    b. Łamana sprowadzana jest do postaci standardowej.
    c. Dla każdego odcinka łamanej tworzony jest wierzchołek n przylegający do nowo utworzonej hiperkrawędzi w grafie.

i. Ustawiane są atrybuty charakterystyczne dla wierzchołka.
ii. Jeżeli odcinek łamanej nie należy do łamanej B, dla tworzonego wierzchołka n wyszukiwany jest tzw. *wierzchołek funkcji osadzenia*, oznaczany jako n_{emb}. Jest to ten z wierzchołków przyległych do e_A, który reprezentował odcinek łamanej A, odpowiadający bieżącemu odcinkowi, lub fragmentowi odcinka, jeżeli bieżący odcinek powstał na skutek podziału w kroku 1a. Dla odcinków należących do łamanej B n_{emb} jest pusty.
iii. Etykietą wierzchołka jest numer odcinka w ciągu tworzącym łamaną, skonkatenowany z kropką oraz etykietą wierzchołka funkcji osadzenia n_{emb} (np. „6.7.5”). Jeżeli n_{emb}jest pusty etykietą jest numer odcinka w łamanej.
iv. Jeżeli n_{emb} nie jest pusty, konieczne jest skopiowanie relacji z n_{emb} do n. Dla każdej relacji, w której uczestniczy n_{emb} sprawdzane jest, czy relacja ta obejmuje odcinki mające część wspólną, z n (jeżeli bieżący odcinek powstał na skutek podziału w kroku 1a, nie musi tak być). W przypadku istnienia części wspólnej tworzona jest nowa hiperkrawędź relacyjna, różniąca się od pierwotnej jedynie zastąpieniem w jednej ze stron relacji n_{emb} wierzchołkiem n.
v. Jeżeli n_{emb} nie jest pusty, konieczne jest przeniesienie informacji o atrybutach drzwi. Sprawdzane jest, czy odcinek reprezentowany przez n zawiera lokalizację drzwi, które leżały odcinku n_{emb}, jeżeli tak odpowiednie atrybuty kopiowane są do n.

d. Na podstawie analizy wnętrza łamanej obliczane są współrzędne środka ciężkości figury, oraz maksymalne wymiary w poziomie i pionie.

4. Nowo utworzone hiperkrawędzie e_C oraz e_D są zagnieżdżane w hiperkrawędzi e_A, reprezentującej pomieszczenie przed podziałem.
5. Dla nowo utworzonych wierzchołków reprezentujących ściany pomiędzy e_C oraz e_D(leżące na łamanej B), tworzone są odpowiednie hiperkrawędzie relacyjne, zagnieżdżane następnie w e_A.
6. Usuwane są wszystkie relacje, do których należą wierzchołki przyległe do e_A.
7. Usuwane są wszystkie wierzchołki przyległe do e_A.

Cofnięcie podziału pomieszczenia.

Operacja odwrotna do opisanej w poprzednim punkcie. W celu odtworzenia sytuacji przed podziałem używane są etykiety wierzchołków.

Umieszczenie drzwi na odcinku ściany.

1. Znajdywane są wierzchołki reprezentujące ścianę, dla której dodawane są drzwi. Jeżeli drzwi zostały umieszczone na obrysie(drzwi zewnętrzne), istnieje tylko jeden taki wierzchołek, w przeciwnym przypadku istnieją dwa.
2. Dla znalezionych wierzchołków dodawane są odpowiednie atrybuty reprezentujące parametry drzwi.
3. W przypadku drzwi które nie są zewnętrzne, znajdywana jest hiperkrawędź relacyjna łącząca dwa wierzchołki, jej typ zmieniany jest na dostępność.

Zmiana rodzaju pomieszczenia.

Odnajdywana jest hiperkrawędź obiektowa reprezentująca dane pomieszczenie, następnie uaktualniany jest jej atrybut oznaczający typ pomieszczenia.

Zmiana etykiety pomieszczenia.

Operacja analogiczna jak w poprzednim punkcie.

3.4.3. Ukośne ściany

System opisany w tej pracy operuje na rozkładach pomieszczeń złożonych z pomieszczeń o ścianach prostopadłych do siebie. W rzeczywistości ściany mogą być również ukośne, może się więc pojawić pytanie czy jest to podyktowane jakimiś istotnymi ograniczeniami w konstrukcji systemu.

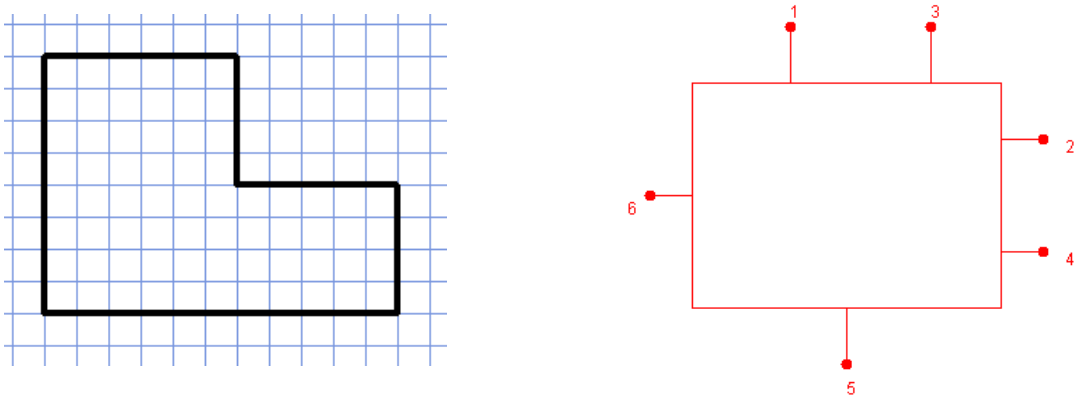
W bieżącym rozdziale wymienione zostały zmiany w systemie jakie należałoby przeprowadzić, aby umożliwić rysowanie ukośnych ścian. Zarówno modyfikacje na poziomie hipergrafu jak i zmiany w algorytmach ekstrakcji wiedzy oraz sprawdzania projektów okazują się nie być duże. Można więc na tej podstawie sadzić, że założenia systemu są uniwersalne.

Istotne zmiany miałyby miejsce w edytorze rozkładów pomieszczeń. Sam edytor, wraz z interfejsem użytkownika, wymagał pewnego nakładu pracy (ponad 7 000 linii kodu), a ograniczenie kierunków ścian było ułatwieniem podczas jego tworzenia. Warto zaznaczyć, że edytor rozkładów pomieszczeń należy traktować jako prototyp badawczy. Sam edytor rozkładów pomieszczeń nie jest jednak główną wartością dodaną niniejszej pracy, są nią

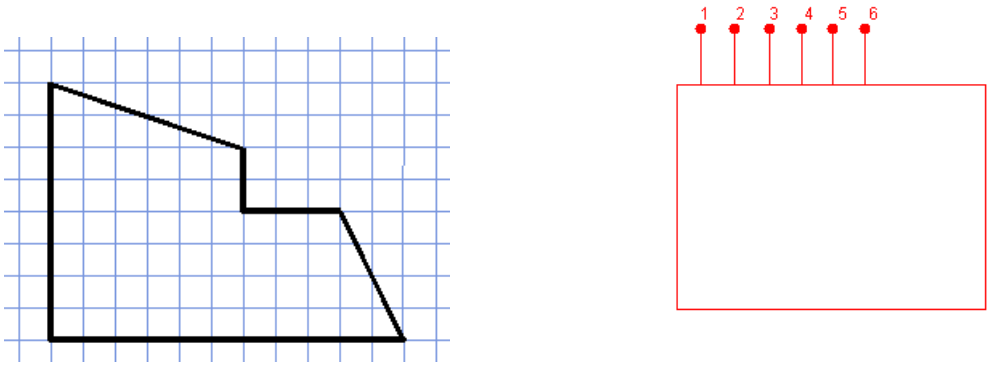
struktury danych oraz algorytmy ekstrakcji i przetwarzania wiedzy projektowej działające na danych wprowadzonych przy użyciu edytora.

Zmiany w hipergrafie

Na poziomie hipergrafu ściany reprezentowane są przez węzły. Posiadają one atrybut, mówiący o orientacji ściany (N, S, W, E). Atrybut ten używany jest między innymi do rozmieszczania węzłów podczas rysowania hipergrafu (Rysunek 3.7). Modyfikacja hipergrafu pozwalająca na wprowadzenie ukośnych ścian polegałaby na usunięciu konieczności posiadania tego atrybutu przez węzły. Atrybut w dalszym ciągu pozostałby, ale jako opcjonalny. Mógłby on być istotny w testach, podczas sprawdzania orientacji ścian niektórych pomieszczeń, jak np. orientacja kuchni na południe. Ścianom ukośnym przypisane byłyby kierunki pośrednie, SW, SE itd. W tym wariancie wszystkie wierzchołki byłyby rysowane przy jednej z krawędzi (Rysunek 3.8).



Rysunek 3.7 Przykład pomieszczenia z prostopadłymi ścianami oraz odpowiadająca mu reprezentacja hipergrafowa



Rysunek 3.8 Przykład pomieszczenia z ukośnymi ścianami oraz odpowiadająca mu reprezentacja hipergrafowa

Zmiany w algorytmach ekstrakcji wiedzy oraz sprawdzania projektów

Zmiana miałaby miejsce w algorytmie podziału pomieszczenia opisanym w rozdziale 3.4.2. W punktach 3.c.ii oraz 3.c.iv zachodzi potrzeba wyszukania fragmentu łamanej mającego część wspólną z inną łamaną. W chwili obecnej, algorytm korzysta w tym kroku z orientacji ściany. Sprawdza on które odcinki są równoległe, następnie które leżą na jednej linii i wreszcie które maja cześć wspólną. W przypadku ścian nachylonych pod dowolnym kątem należałoby zamienić sprawdzanie kierunków pionowy/poziomy na sprawdzanie współczynnika kierunkowego prostej.

Zmiany w edytorze rozkładów pomieszczeń

Ograniczenie możliwości szkicowania do ścian pionowych i poziomych było istotnym ułatwieniem podczas tworzenia kodu edytora rozkładów pomieszczeń. Wprowadzenie ukośnych ścian nie powodowałoby co prawda dużych zmiana podczas rysowania obrysu, natomiast bardziej złożony jest proces dzielenia pomieszczeń. W chwili obecnej system wykrywa gdy łamana spotyka się z inną ścianą pomieszczenia. W przypadku rysowania pod dowolnym kątem użytkownik rysowałby odcinek po odcinku, a punkty przecięcia linii nie musiałyby wypadać na kratkach siatki. Również funkcje tworzące wnętrza pomieszczeń, które są używane w kilku miejscach edytora, musiałby być przystosowane do obsługi dowolnych kształtów nie koniecznie pokrywających się z siatką. Wszystkie te zmiany, nie zawierają wyzwań badawczych.

3.5. Język wymagań projektowych

Język testów HSSDR został stworzony, aby umożliwić zapis wymagań projektowych, które pojawiają się w wielu dziedzinach projektowania architektonicznego. Tego typu ograniczenia nie są możliwe do przewidzenia podczas tworzenia systemu wspomagającego projektowanie. Mogą one w szczególności określać wymagania stawiane konkretnym projektom przez zleceniodawców. Dotychczasowe prace w obszarze wspomagania projektowania dotyczyły norm prawnych i ogólnych standardów projektowania, nie umożliwiając użytkownikom definiowania własnych reguł [11]. System HSSDR początkowo również posiadał wszystkie reguły zapisane na sztywno w kodzie programu. Z punku widzenia tworzenia systemu wspomagającego projektowanie, zamknięcie reguł wewnątrz jest dużo łatwiejszym rozwiązaniem niż udostępnienie możliwości ich zmiany lub tworzenia nowych. Natomiast udostępnienie takich możliwości wprowadza znaczną zmianę jakościową do procesu projektowania. Wiedza projektowa z wiedzy ukrytej, staje się sformalizowaną wiedzą wyrażoną explicite w postaci reguł. Stwarza to możliwość zarządzania bazą wiedzy.

Teoretycznie możliwa jest również jej wymiana pomiędzy różnymi aplikacjami lub pomiędzy podmiotami opracowującymi standardy, a twórcami systemów CAD oraz projektantami.

W opisywanym rozwiązaniu formuły logiki pierwszego rzędu są traktowane jako otwarty język zapisu wymagań projektowych. Jego formuły odnoszą się do obiektów istniejących w analizowanym projekcie, a więc w rozważanych w rozprawie projektach do takich obiektów jak drzwi czy pomieszczenia, a także do relacji i atrybutów charakteryzujących obiekty, zarówno bezpośrednich jak i obliczonych przez system.

3.5.1. Składnia logiki pierwszego rzędu

Logika pierwszego rzędu, zwana jest również rachunkiem kwantyfikatorów lub rachunkiem predykatów [39] [98]. Język logiki pierwszego rzędu jest językiem rachunku zdań rozszerzonym o następujące elementy:

- symbole funkcji, relacji oraz stałych,
- zmienne indywiduowe,
- kwantyfikatory.

Funkcje i relacje charakteryzuje liczba ich argumentów. Stałe są definiowane jako symbole funkcji zeroargumentowych. Dodatkowo zakładamy istnienie nieskończonego, przeliczalnego zbioru symboli nazywanych zmiennymi indywiduowymi.

Definicja 3.10 Sygnatura

Sygnaturą S określa się rodzinę zbiorów $M_n \ \ dla\ n \geq 0$ oraz rodzinę zbiorów $R_m\ dla\ m \geq 1$, gdzie elementy M_n to *symbole funkcji n-argumentowych*, a elementy R_m są *symbolami relacji m-argumentowych.*

Definicja 3.11 Zmienne indywiduowe

Niech X będzie nieskończonym i przeliczalnym zbiorem. Zbiór ten będziemy nazywać *zbiorem symboli,* a jego elementy *zmiennymi indywiduowymi*.

Za pomocą wyżej zdefiniowanych elementów składowych języka, definiuje się *termy, formuły atomowe* oraz *formuły*, z których te ostatnie są pełnymi zdaniami języka logiki pierwszego rzędu.

Definicja 3.12 Termy

Termy są definiowane indukcyjnie:

- zmienne indywiduowe oraz stałe są termami,

- jeśli dla liczby naturalnej $n \in \mathbb{N}$, jeśli $t_1, \ldots, t_n$ są termami, to $f(t_1, \ldots, t_n)$ też jest termem, gdzie $f \in M_n$.

Definicja 3.13 Formuły atomowe

Formuły atomowe definiowane są w następujący sposób:

- symbol fałszu $\perp$ jest formuła atomową,
- dla każdego $m \in \mathbb{N}$, dla każdego symbolu relacji *m*-argumentowej $P \in R_m$, oraz dla dowolnych termów $t_1, \ldots, t_m$ napis $P(t_1, \ldots, t_m)$ jest formułą atomową,
- dla dowolnych termów t_1, t_2 napis $t_1 = t_2$ jest formułą atomową.

Definicja 3.14 Formuły

Formuły nad sygnaturą S oraz zbiorem zmiennych indywiduowych X definiowane są indukcyjnie:

- każda formuła atomowa jest formułą,
- jeśli φ, ψ są formułami, to $(\varphi \implies \psi)$ też jest formułą,
- jeśli φ jest formułą, a x jest zmienną indywiduową, to $\forall_x \varphi$ też jest formułą.

Dodatkowo definiuje się negację, koniunkcję, alternatywę, równoważność formuł oraz symbol prawdy. Kwantyfikator egzystencjalny definiowany jest jako skrót notacyjny:

Definicja 3.15 Kwantyfikator egzystencjalny

$$\exists_x \varphi \ \ oznacza \ \ {\sim}\forall_x {\sim}\varphi$$

3.5.2. Język testów HSSDR

Język testów HSSDR jest oparty na języku logiki pierwszego rzędu. Zawiera on także pewne dodatkowe konstrukcje przydatne podczas definiowania zestawów testów dla projektów architektonicznych. Umożliwia on między innymi dodawanie komentarzy oraz instrukcji określających wiadomości przekazywane użytkownikom po wykonaniu testów. Język ten korzysta ze słów kluczowych, zmiennych oraz predefiniowanych funkcji i relacji, ale pozwala też użytkownikom definiować własne relacje.

Z punktu widzenia składni logiki pierwszego rzędu, w skład sygnatury języka testów HSSDR wchodzą następujące elementy:

- symbole relacji jednoargumentowych:

Areas, Rooms, Doors, Sensors, surveilledDoors, externalDoors,

- symbole relacji dwuargumentowych:

  adjacent, accessible, doorsInRoom, sensorInRoom, isPassageWatched, isDirectChild, isInDescendants,

- symbole funkcji jednoargumentowych:

  type, area, label,

- symbole funkcji dwuargumentowych:

  doorsDist,

- symbole stałych, np.:

  „Apartment”, „Kitchen”, 30, 15.5.

Znaczenia dla symboli użytych w języku testów dostarcza omówiona dokładnie w podrozdziale 3.6 *struktura relacyjna*. Definiuje ona także zbiory, na których oparte są kwantyfikatory. Jest ona definiowana na nowo dla każdego grafu, w praktyce po każdej zmianie analizowanego grafu. Dopiero w kontekście struktury relacyjnej formuły logiczne reprezentujące ograniczenia projektowe mogą być poddane ewaluacji.

W języku testów HSSDR istnieje także możliwość definiowania nowych relacji o dowolnej liczbie argumentów w następujący sposób:

$$nazwa(\,) \Leftrightarrow formuła\,;$$

lub

$$nazwa(x, y, z, \dots) \Leftrightarrow formuła\,;$$

Tak zdefiniowana relacja zostaje dodana do sygnatury oraz struktury relacyjnej i może być następnie używana w dalszych formułach. Dla przykładu, w testach opisujących normy przeciwpożarowe (podrozdział 3.7.2), wprowadzona zostaje definicja relacji jednoargumentowej *ZoneDoors*:

$$ZoneDoors(d) \Leftrightarrow exists\ r\ in\ Rooms:\\ doorsInRoom(d, r)\ and\ type(r) = \text{“}Staircase\text{”};$$

która następnie może być użyta w kolejnych formułach, np. w celu zdefiniowana relacji *FireSafetyDoors*:

$$FireSafetyDoors(d) \Leftrightarrow ExternalDoors(d)\ or\ ZoneDoors(d);$$

Specjalna cecha języka pozwala również na definiowanie relacji w sposób rekurencyjny. Ta możliwość, stworzona na wzór języków programowania, pozwala w znaczny sposób uprościć zapis formuł logicznych, zwłaszcza w przypadku definiowania przechodnich relacji określonych na zbiorze pomieszczeń. Poniżej, jako przykład definicji rekurencyjnej

przytoczona jest definicja silni. W zapisie w języku testów HSSDR relacja x *silnia* y zachodzi wtedy i tylko wtedy gdy $y = x!$.

$$silnia(x,y) \Leftrightarrow x >= 1 \; and \; y >= 1 \; and \; ((x = 1 \; and \; y = 1) \; or \; silnia(x-1, y/x));$$

Zestawy testów zapisywane są w plikach tekstowych. Każdy plik może zawierać jeden lub więcej testów znajdujących się przeważnie na końcu pliku, oraz dowolną liczbę konstrukcji pomocniczych. Każdy taki zestaw interpretowany jest oddzielnie. Więcej informacji na temat realizacji tego procesu znajduje się w rozdziale 3.6.4.

Z punktu widzenia implementacji, interpreter języka testów HSSDR jest programem napisanym w Javie, który do celów parsowania zestawów testów korzysta z biblioteki ANTLR [99]. ANTLR jest generatorem praserów typu LL(*). Jest on często stosowany zarówno w zastosowaniach akademickich jak i w przemyśle informatycznym.

Atomy leksykalne wchodzące w skład języka testów HSSDR, widziane z punktu widzenia parsera ANTLR zostały przedstawione w tabeliTabela 3.1. Niektóre z nich mają równoważne formy zapisu jako słowo kluczowe, symbole LaTeX oraz znaki Unicode.

Znaczenie	**Słowo kluczowe lub sposób zapisu**	**Symbol LaTeX**	**Symbol Unicode**
Kwantyfikator ogólny	forall ... in	\forall ... \in	u2200 ... u220A
Kwantyfikator egzystencjalny	exists ... in	\exists ... \in	u2203 ... u220A
Równoważność formuł, definiowanie relacji	<=>	\Leftrightarrow	u21D4
Implikacja	=>	\Rightarrow	u21D2
Koniunkcja	And	\wedge	u2227
Alternatywa	or	\ vee	u2228
Negacja	not, !	\neg	u00AC
Równość	=		
Zaprzeczenie równości	!=, <>	\neq	u2260

Relacje na liczbach: mniejsze, mniejsze lub równe, większe, większe lub równe	<,<=,>,>=		
Zmienna indywidualna	Spełnia wyrażenie regularne: (a..z|A..Z)(a..z|A..Z|0..9|_)*		
Liczba całkowita lub wymierna	Spełnia wyrażenie regularne: 0..9+(.0..9+)?EXP?|.0..9+EXP? gdzie EXP: (e|E) (+|-)? (0..9)+		
Symbole operacji arytmetycznych, operatory unarne liczb dodatnich i ujemnych	+ (dodawanie), - (odejmowanie), * (mnożenie), / (dzielenie)		
Stała tekstowa	Tekst ujęty w cudzysłowie		
Komentarz	Od znaku # do końca linii		
Wiadomość informująca o niespełnieniu warunku	failure_msg		
Wiadomość informująca o spełnieniu warunku	success_msg		
Predefiniowane zbiory, relacje oraz funkcje	Rooms, $\mathrm{accessible}(x, y)$, Doors, doorsdist(d1, d2)		
Relacje zdefiniowane w ramach języka testów	nazwa_reacji(<argumety>) <=> <definicja reacji>;		
Średnik (wymagany po każdej formule)	;		

Tabela 3.1

3.5.3. Alternatywne formy definiowania wymagań projektowych

Motywacją dla wybrania logiki pierwszego rzędu do zapisu wymagań projektowych była jej siła wyrazu, a także pokrewieństwo ze źródłową formą wymagań, formułowanych w języku naturalnym, dzięki któremu język wymagań mógłby być używany przez duże grono osób, nie tylko przez inżynierów czy osoby techniczne.

Logika pierwszego rzędu posiada znaczną siłę wyrazu, co prowadzi do jej nierozstrzygalności. Nierozstrzygalny jest problem decyzyjny: „Czy dana formuła logiki pierwszego rzędu jest tautologią?" W praktyce, w przypadku rozważań na zbiorach skończonych, do jakich można sprowadzić rozważania o projektach architektonicznych składających się ze skończonej liczby elementów składowych problem nierozstrzygalności nie powoduje problemów. Język logiki prezentowany w rozprawie posiada ograniczenie niezezwalające na obejmowanie kwantyfikatorami zbiorów nieskończonych, takich jak na przykład liczby naturalne. Autor nie napotkał warunku, którego zapis uniemożliwiłoby to ograniczenie. W testach zgodności stawia się tezy, których centralnymi elementami są raczej elementy projektu niż liczby naturalne czy ciągi znaków.

Poniżej przedstawione zostały alternatywne dla logiki pierwszego rzędu formy zapisu ograniczeń projektowych, jakie mogłyby być rozważane dla systemu z bazą wiedzy analizującego projekty architektoniczne.

Logika opisowa

Logika opisowa ma słabszą siłę wyrazu niż logika pierwszego rzędu. Oferuje ona natomiast maksymalną siłę wyrazu przy zachowaniu rozstrzygalności (wszystkie problemy zostaną rozstrzygnięte, a każde wnioskowanie zostanie zakończone w skończonym czasie). Klasy logiki opisowej odpowiadają formułom logiki pierwszego rzędu z jedną zmienną wolną. W logice opisowej nie ma w ogóle możliwości posługiwania się zmiennymi explicite. Nie da się zapisać relacji z dwoma zmiennymi wolnymi, podczas gdy w logice pierwszego rzędu liczba zmiennych wolnych nie jest ograniczona. Brak zmiennych powoduje, że nie można opisać klas, których elementy są uzależnione od istnienia anonimowych obiektów o określonych właściwościach [100]. Nie można na przykład zapisać w logice opisowej następującej formuły [101]:

$$\forall\, p_1, p_2 \in Pomieszczenia:$$
$$sekcjaPubliczna(\, p_1) \wedge sekcjaPrywatna(\, p_2) \wedge dostępny(p_1, p_2)$$
$$\rightarrow przejściePodNadzorem(p_1, p_2)$$

Brak możliwości posługiwania się zmiennymi uniemożliwia również zapis warunków zawierających podformuły z kwantyfikatorami obejmującymi dodatkowe zmienne. Przykładem takiej formuły jest zapis wytycznych opisanych w rozdziale 4.2:

„Winda towarowa nie powinna być traktowana jako część drogi dostępu, chyba że jedynym zastosowanym w obiekcie typem windy są windy osobowo-towarowe."

Warunek ten można natomiast zapisać w logice pierwszego rzędu:

$$windaDozwolona(p) \Leftrightarrow$$
$$windaOsobowa(p) \vee (windaTowarowa(p) \wedge \sim \exists\, q \in Pomieszczenia : windaOsobowa(q))$$

Logika opisowa z kolei zawiera operator domknięcia przechodniego relacji, którego logika pierwszego rzędu nie jest w stanie wyrazić. Istotnie, w definicji domknięcia przechodniego mamy „$istnieje\ ciąg\ elementów\ pośrednich\ p_1..p_n$", gdzie n to dowolnie duża liczba naturalna. Kwantyfikator jest tutaj oparty na zbiorze, a nie na zmiennej. Dlatego też w przypadku relacji bezpośredniej dostępności pomiędzy dwoma pomieszczeniami *accessible(x,y)* nie można zdefiniować wprost przechodniego domknięcia tej relacji, czyli dostępności poprzez dowolną liczbę pomieszczeń. Można ten problem obejść definiując domknięcie przechodnie ograniczone ustaloną liczba pośrednich relacji, co w praktyce zupełnie wystarcza.

Logika wyższego rzędu

Ze względu na zasięg kwantyfikatorów logika wyższego rzędu nie nadaje się do komputerowej analizy.

Grafy oraz gramatyki grafowe

Grafy oraz gramatyki grafowe mogą być użyte w procesie walidacji projektu, przykładem ich zastosowania jest system ConDes opisany w rozdziale 2.5. Innymi opisanymi w tym rozdziale systemami operującymi na grafach są GraCAD oraz system Chein'a, nie sprawdzają one jednak wymagań projektowych. W ConDes możliwe jest określenie reguł, które wymuszają wartość atrybutu dla obiektu, narzucają lub zakazują istnienie relacji między obiektami oraz określających liczbę pewnych obiektów w projekcie. Można by sobie wyobrazić również definiowanie niedozwolonych konstrukcji w postaci grafów, a następnie wyszukiwanie w grafie projektu takich podgrafów. Główną siła mechanizmów grafowych jest opisywanie struktury i topologii obiektów, podczas gdy reguły dla projektów architektonicznych odnoszą się w równej części do struktury jak do właściwości elementów składowych projektu. Dlatego logika pierwszego rzędu wydaje się być bardziej elastyczna umożliwiając operowanie na liczbach, wartościach funkcji, atrybutach obiektów, a jednocześnie formułowanie tez o wzajemnych relacjach pomiędzy obiektami.

Prolog lub inny język programowania logicznego

Zdefiniowany w niniejszej pracy język logiki pierwszego rzędu jest de facto nowym językiem programowania logicznego. Można wziąć pod uwagę również zastosowanie istniejących już języków programowania logicznego, jak np. Prolog [42].

Stworzenie własnego języka na potrzeby systemu HSSDR było rozwiązaniem praktyczniejszym. Jednym z powodów jest brak konieczności włączenia do systemu kompilatora i środowiska uruchomieniowego. Ten krok znacznie spowolniłby proces walidacji projektu. Wiedza wydedukowana z projektu musiałyby być najpierw przekształcona w fakty i zależności programu Prologu, do którego byłyby dołączone wymagania projektowe, jako cel. Następnie taki program byłby uruchamiany i na tym etapie Prolog budowałby z faktów swoje wewnętrzne struktury, na których przeprowadzone byłoby wnioskowanie. W proponowanym rozwiązaniu analogiczne struktury są zasilane bezpośrednio. Kolejnym argumentem jest elastyczność, nie trzeba się godzić na ograniczenia narzucone przez język. Można też w dowolny sposób zdefiniować składnię i słowa kluczowe. Dzięki temu można stworzyć język łatwiejszy do przyswojenia dla użytkowników, takich jak architekci czy inżynierowie wiedzy. Istotnym ograniczeniem byłaby również konieczność, zapisania całej wiedzy o projekcie w postaci faktów przed rozpoczęciem wnioskowania. Nie byłoby możliwości dynamicznego wartościowania kosztownych obliczeniowo właściwości projektu w czasie wartościowania formuł. W proponowanym rozwiązaniu, pewne wartościowania są wykonywane wyłącznie, gdy istnieje potrzeba zwrócenia wartości funkcji dla konkretnych argumentów (Rozdział 3.6.4), co przyśpiesza proces ewaluacji formuł.

3.6. Wiedza projektowa

3.6.1. Struktury relacyjne

W poprzednim rozdziale opisany został język logiki pierwszego rzędu. Formalizm ten definiuje składniowo poprawne formuły w kontekście danej sygnatury, nie mówi jednak nic o ich ewaluacji. Nie podaje on przepisu jak określić czy dana formuła jest prawdziwa lub fałszywa w odniesieniu do konkretnych bytów należących do dziedziny, o której chcemy prowadzić rozważania posługując się logiką pierwszego rzędu. To *struktura relacyjna* dostarcza znaczenia relacjom i funkcjom użytym w formułach oraz zapewnia zbiór zwany *nośnikiem* - dziedzinę wnioskowania do której będą należały wszystkie elementy, których dotyczą formuły. Przyporządkowuje ona symbolom stałych elementy z nośnika, a symbolom funkcji i relacji przyporządkowuje ich interpretacje będące funkcjami i relacjami określonymi na nośniku. Interpretacji zmiennych indywiduowych zawartych w formułach dostarczać będzie natomiast funkcja zwana *wartościowaniem*, przyporządkowująca zmiennym elementy nośnika. Poniżej znajdują się definicje struktury relacyjnej oraz wartościowania przytoczone na wzór [98] oraz [39], które odwołują do wcześniejszych definicji z podrozdziału 3.5.1.

Definicja 3.16 Struktura relacyjna

Dla sygnatury S, struktura relacyjna $\mathfrak{R}$ jest niepustym zbiorem A zwanym nośnikiem, wraz z interpretacją każdego symbolu operacji $f \in M_n$ jako funkcji n-argumentowej $f^{\mathfrak{R}} : A^n \to A$, oraz każdego symbolu relacji $r \in R_m$ jako relacji *m*-argumentowej $r^{\mathfrak{R}} \subseteq A^m$.

Strukturę relacyjną można więc przedstawiać jako krotkę postaci $\mathfrak{R} = (A, f_1^{\mathfrak{R}}, \dots, f_i^{\mathfrak{R}}, r_1^{\mathfrak{R}}, \dots, r_j^{\mathfrak{R}})$, gdzie $f_1, \dots, f_i, r_1, \dots, r_j$, są wszystkimi symbolami danej sygnatury.

Definicja 3.17 Wartościowanie

Wartościowaniem w strukturze relacyjnej $\mathfrak{R}$ dla sygnatury S, będziemy nazywać dowolną funkcję przyporządkowującą elementom zbioru zmiennych X elementy nośnika A, tzn.

$$\varrho : X \to A .$$

Wyżej zdefiniowana funkcja wartościowania posłuży dalej do interpretacji termów oraz formuł. Funkcję ϱ rozszerzamy tak, aby przyporządkować termom elementy zbioru A. Rozszerzenie wartościowania ϱ_T jest zdefiniowane w sposób indukcyjny:

- $\varrho_T(x) = \varrho(x)$,
- $\varrho_T\big(f(t_1, \dots, t_n)\big) = f^{\mathfrak{R}}(\varrho(t_1), \dots, \varrho(t_n))$,

gdzie $t_1, \dots, t_n$ są termami, $f \in M_n$, a $f^{\mathfrak{R}}$ jest interpretacją f w $\mathfrak{R}$.

Dla wartościowania ϱ_T niech $\varrho_T[x/a]$ będzie wartościowaniem równoważnym ϱ_T dla wszystkich argumentów oprócz zmiennej x, której przyporządkowana jest wartość a:

$$\varrho_T[x/a](y) = \begin{cases} \varrho_T(y), dla\ y \neq x \\ a, \quad dla\ y = x \end{cases}$$

Powyższe definicje pozwalają zdefiniować pojęcie *spełnialności* formuły φ w strukturze $\mathfrak{R}$ przy wartościowaniu ϱ, zapisywane jako:

$$(\mathfrak{R}, \varrho) \models \varphi .$$

Definicja 3.18 Formuła spełniona

Formuła φ jest *spełniona* w strukturze $\mathfrak{R}$ przy wartościowaniu ϱ, wtedy i tylko wtedy gdy:

- Nie zachodzi $(\mathfrak{R}, \varrho) \models \perp$ ($\perp$ to symbol fałszu),
- Dla dowolnego $m \geq 1$, $r \in R_m$ oraz dla dowolnych termów $t_1, \dots, t_m$:

$$(\mathfrak{R}, \varrho) \models r(t_1, \dots, t_m) \Leftrightarrow (\varrho_T(t_1), \dots, \varrho_T(t_m)) \in r^{\mathfrak{R}},$$

- $(\mathfrak{R}, \varrho) \models t_1 = t_2 \Leftrightarrow \varrho_T(t_1) = \varrho_T(t_2)$,
- $(\mathfrak{R}, \varrho) \models \varphi \Rightarrow \psi$, gdy nie zachodzi $(\mathfrak{R}, \varrho) \models \varphi$, lub zachodzi $(\mathfrak{R}, \varrho) \models \psi$,

- $(\mathfrak{R}, \varrho) \models \forall_x \varphi$, wtedy i tylko wtedy, gdy dla dowolnego $a \in A$ zachodzi:

$$(\mathfrak{R}, \varrho_T[x/a]) \models \varphi.$$

W formułach rozróżnia się zmienne *wolne* i *związane*. Intuicyjnie, zmienna jest *związana* jeśli jest objęta zasięgiem kwantyfikatora, w przeciwnym przypadku jest ona *zmienną wolną*.

Wykazuje się, że spełnianie formuły zależy jedynie od wartościowania na zmiennych wolnych. Jeśli formuła nie posiada zmiennych wolnych, jej prawdziwość jest niezależna od przyjętego wartościowania.

Definicja 3.19 Spełnialność i prawdziwość formuł, tautologie

Formuła φ jest *spełnialna w strukturze* $\mathfrak{R}$ gdy istnieje wartościowanie ϱ w $\mathfrak{R}$, takie, że zachodzi $(\mathfrak{R}, \varrho) \models \varphi$. Formuła jest *spełnialna*, jeśli istnieje struktura $\mathfrak{R}$ w której ta formuła jest spełnialna.

Formuła φ jest *prawdziwa* w $\mathfrak{R}$, gdy dla każdego wartościowania ϱ w $\mathfrak{R}$ zachodzi:

$$(\mathfrak{R}, \varrho) \models \varphi.$$

Formuła jest *tautologią*, gdy jest prawdziwa w każdej strukturze.

3.6.2. Wielorodzajowe struktury relacyjne

Złożone problemy techniczne wymagają zdefiniowania bardziej złożonego nośnika struktury relacyjnej i w konsekwencji nałożenia dodatkowych warunków na interpretację relacji i funkcji. W takim wypadku stosuje się *struktury wielorodzajowe* (ang. many-sorted). Sygnaturę języka rozszerza się o symbole *rodzajów* (ang. sorts), oznaczane jako k i tworzące zbiór rodzajów K. Spełnione są następujące warunki:

- każdej stałej przyporządkowuje się jej rodzaj,
- elementom zbioru relacji *m*-argumentowych R_m przyporządkowuje się rodzaje ich argumentów $k_1, \dots, k_m$,
- elementom zbioru funkcji n-argumentowych M_n, przyporządkowuje się rodzaje ich argumentów $k_1, \dots, k_n$ oraz rodzaj wartości funkcji k_{n+1}.

Dodatkowo zakłada się istnienie zbioru zmiennych wolnych dla każdego rodzaju. Korzystając z powyższych właściwości dla stałych i funkcji określa się rodzaj termów. W przypadku formuł atomowych postaci $t_1 = t_2$, termy t_1 oraz t_2 musza być tego samego rodzaju. Dla relacji $P(t_1, \dots, t_m)$ rodzajami argumentów relacji są rodzaje termów $t_1, \dots, t_m$.

W strukturze relacyjnej zakłada się istnienie nośnika A_k dla każdego rodzaju k. Analogicznie definiuje się rozszerzenie wartościowania w strukturze relacyjnej.

3.6.3. Struktura relacyjna hipergrafu

W poprzednim podrozdziale podana została ogólna definicja struktury relacyjnej, bieżący podrozdział przytacza natomiast definicję *struktury relacyjnej hipergrafu*. Taka struktura jest używana podczas ewaluacji formuł języka testów HSSDR. Nadaje ona znaczenia relacjom i funkcjom użytym w formułach w odniesieniu do konkretnego hipergrafu. Elementy tego hipergrafu wchodzą w skład nośnika struktury relacyjnej hipergrafu. Wraz z każdą zmianą w hipergrafie, zmienia się dziedzina wnioskowania, a struktura relacyjna jest tworzona od nowa. Każda zmiana w grafie może wpłynąć na prawdziwość formuł.

Dla danego hipergrafu G, jest tworzona odpowiadająca temu hipergrafowi wielorodzajowa struktura relacyjna $\mathfrak{R}_G$ (oznaczenie grafu będzie pomijane, jeśli wynika z kontekstu, o jaki graf chodzi).

W skład wielorodzajowej struktury relacyjnej wchodzą następujące rodzaje:

$$K = (\mathcal{A}, \mathcal{D}, \mathcal{S}, \mathcal{E}, \mathcal{R})$$

Te rodzaje reprezentują odpowiednio: pomieszczenia, drzwi, czujniki (areas, doors, sensors), etykiety hiperkrawędzi oraz wierzchołków oraz liczby rzeczywiste. Dla każdego rodzaju k określony jest jego nośnik A_k.

Definicja 3.20 Nośniki wielorodzajowej struktury relacyjnej $\mathfrak{R}_G$ hipergrafu G

$$A_{\mathcal{A}} = Areas^{\mathfrak{R}}$$
$$A_{\mathcal{D}} = Doors^{\mathfrak{R}}$$
$$A_{\mathcal{S}} = Sensors^{\mathfrak{R}}$$
$$A_{\mathcal{E}} = \Sigma_G^C \cup \Sigma_G^R \cup \Sigma_G^V \cup At_G$$
$$A_{\mathcal{R}} = \mathbb{R}$$

gdzie:

- $Areas^{\mathfrak{R}} = \{ e \in E_G^C \}$,
- $Doors^{\mathfrak{R}} = \bigcup_{v \in V_G} doorsAttributes_G(v)$,
- $Sensors^{\mathfrak{R}} = \bigcup_{e \in E_G^C} sensors_G(e)$,
- Σ_G^C jest zbiorem etykiet krawędzi obiektowych grafu G,
- Σ_G^R jest zbiorem etykiet krawędzi relacyjnych grafu G,

- Σ_G^V jest zbiorem etykiet wierzchołków grafu G,
- At_G jest zbiorem wartości atrybutów krawędzi oraz wierzchołków grafu G.

W skład struktury $\mathfrak{R}$ wchodzi, obok nośnika dla każdego rodzaju, interpretacja symboli relacji, funkcji oraz stałych, należących do sygnatury S języka logiki pierwszego rzędu. To przyporządkowanie dla relacji i funkcji jest określone naturalnie, to znaczy symbolowi funkcji f przyporządkowywana jest funkcja $f^{\mathfrak{R}}$, (np. $type \rightarrow type^{\mathfrak{R}}$). Rodzaje elementów relacji i argumentów funkcji związane z wielorodzajowością struktury relacyjnej wynikają wprost z jej definicji. Stałym tekstowym oraz liczbowym przyporządkowane są elementy zbioru etykiet oraz zbioru atrybutów.

Poniżej znajdują się definicje relacji i funkcji przyporządkowanych symbolom relacji i funkcji języka testów przez strukturę relacyjną $\mathfrak{R}_G$. Są one określone na elementach zbioru nośnika, w odniesieniu do składowych hierarchicznego atrybutowanego hipergrafu (Definicja 3.5). Rodzaje argumentów relacji i funkcji podane są po dwukropku przy każdym argumencie: np.: $relacja\big(argument_1{:}\,rodzaj_{argument_1}, argument_2{:}\,rodzaj_{argument_2}, \dots\big)$.

Definicja 3.21 Relacje wchodzące w skład struktury relacyjnej $\mathfrak{R}_G$ hipergrafu G

- $Areas^{\mathfrak{R}}(a{:}\,\mathcal{A})$,
- $Rooms^{\mathfrak{R}}(a{:}\,\mathcal{A}) \Leftrightarrow\ a \in E_G^C \wedge ch_G(a) = \emptyset$,
- $Doors^{\mathfrak{R}}(d{:}\,\mathcal{D})$,
- $Sensors^{\mathfrak{R}}(s{:}\,\mathcal{S})$,
- $adjacent^{\mathfrak{R}}(a{:}\,\mathcal{A}, b{:}\,\mathcal{A}) \Leftrightarrow$
  $\exists\, r \in E_G^R : lb_G(r) = \text{"adj"} \wedge s_G(a) \cap s_G(r) \neq \emptyset \wedge s_G(r) \cap s_G(b) \neq \emptyset$,
- $accessible^{\mathfrak{R}}(a{:}\,\mathcal{A}, b{:}\,\mathcal{A}) \Leftrightarrow$
  $\exists\, r \in E_G^R : lb_G(r) = \text{"acc"} \wedge s_G(a) \cap s_G(r) \neq \emptyset \wedge s_G(r) \cap s_G(b) \neq \emptyset$,
- $isDirectChild^{\mathfrak{R}}(a{:}\,\mathcal{A}, b{:}\,\mathcal{A}) \Leftrightarrow\ a \in ch_G(b)$
- $isInDescendants^{\mathfrak{R}}(a{:}\,\mathcal{A}, b{:}\,\mathcal{A}) \Leftrightarrow a \in ch_G^+(b)$
- $doorInARoom^{\mathfrak{R}}(d{:}\,\mathcal{D}, a{:}\,\mathcal{A}) \Leftrightarrow \exists v \in V_G : \text{v} \in s_G(a) \wedge \text{d} \in doorsAttributes_G(v)$,
- $externalDoors^{\mathfrak{R}}(d{:}\,\mathcal{D}) \Leftrightarrow$
  $$\forall\, a_1, a_2 \in E_G^C : doorInARoom_{\mathfrak{R}}(d, a_1) \wedge doorInARoom_{\mathfrak{R}}(d, a_2) \Rightarrow a_1 = a_2,$$
- $sensorInRoom^{\mathfrak{R}}(s{:}\,\mathcal{S}, a{:}\,\mathcal{A}) \Leftrightarrow s \in sensors_G(a)$,
- $isPassageWatched^{\mathfrak{R}}(d_1{:}\,\mathcal{D}, d_2{:}\,\mathcal{D}) \Leftrightarrow$
  $\exists\, a \in E_G^C : doorInARoom^{\mathfrak{R}}(d_1, a) \wedge doorInARoom^{\mathfrak{R}}(d_2, a)$,

oraz nie istnieje ścieżka pomiędzy drzwiami d_1 i d_2, taka, że dla każdego czujnika $s \in Sensors^{\Re}$, prowadzi przez pola nieobjęte zasięgiem s (podrozdział 3.7.3),

- $surveilledDoors^{\Re}(d{:}\mathcal{D}) \Leftrightarrow$ nie istnieje czujnik $s \in Sensors^{\Re}$, taki że drzwi d znajdują się w zasięgu s.

Definicja 3.22 Funkcje wchodzące w skład struktury relacyjnej $\Re_G$ hipergrafu G

- $type^{\Re}(a{:}\mathcal{A}) \rightarrow t{:}\mathcal{E}$, taka że:

$$type^{\Re}(a) = type_G(a),$$

- $label^{\Re}(a{:}\mathcal{A}) \rightarrow l{:}\mathcal{E}$, taka że:

$$label^{\Re}(a) = lb_G(a),$$

- $area^{\Re}(r{:}\mathcal{A}) \rightarrow x{:}\mathcal{R}$, taka że:

$$area^{\Re}(r) = area_G(r)$$

(dla hiperkrawędzi obiektowej, powierzchnia pomieszczenia reprezentowanego przez hiperkrawędź),

- $doorsDist^{\Re}(d_1{:}\mathcal{D}, d_2{:}\mathcal{D}) \rightarrow x{:}\mathcal{R}$, taka, że:

$$doorsDist^{\Re}(d_1, d_2) = \begin{cases} dist_G(d_1, d_2), & \text{gdy } \exists\ a \in E_G^C{:}\ doorInARoom^{\Re}(d_1, a) \wedge doorInARoom^{\Re}(d_2, a) \\ -1, & \text{w przeciwnym przypadku} \end{cases}$$

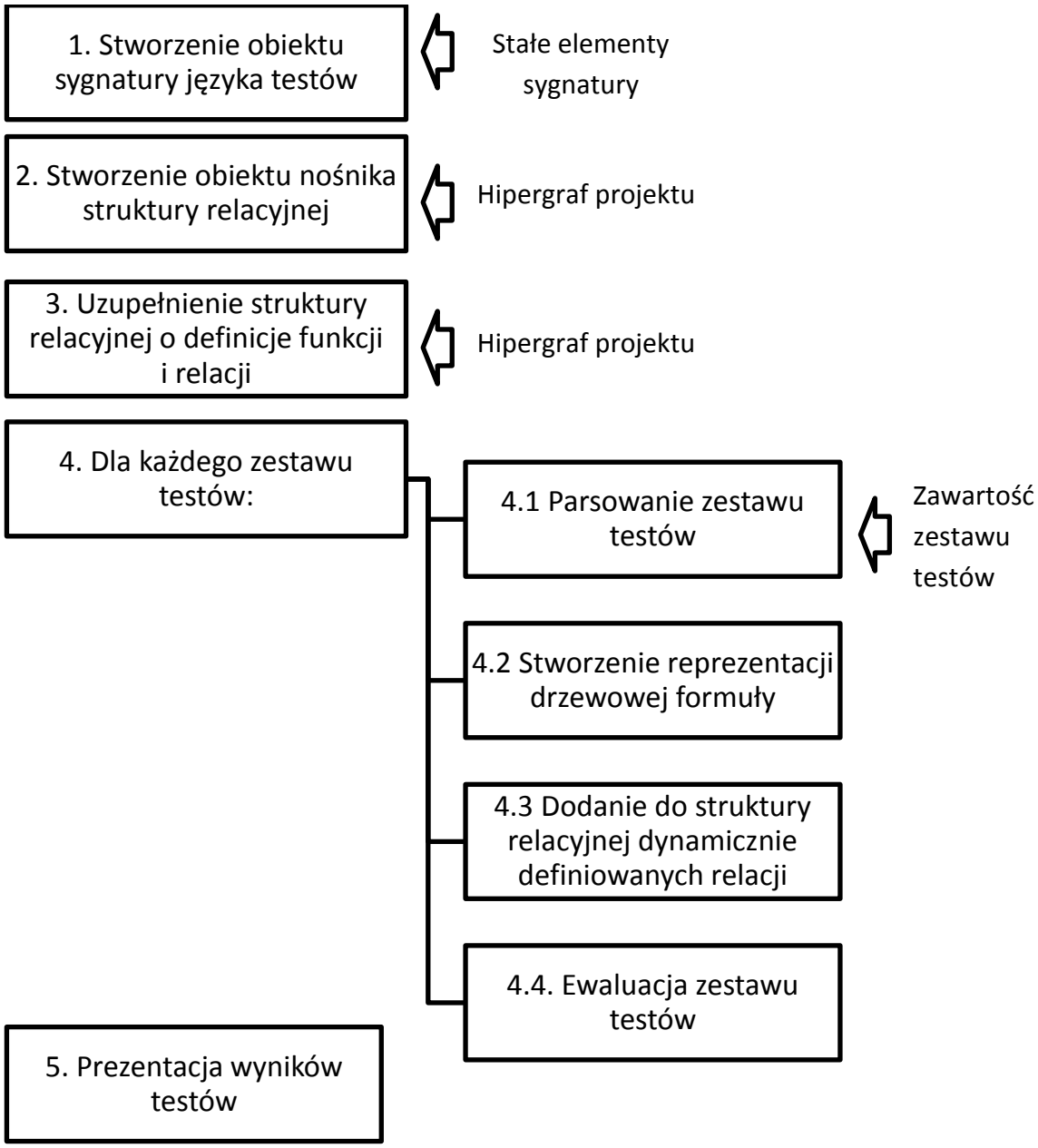
gdzie $dist_G(d_1, d_2)$ oznacza długość w metrach najkrótszej ścieżki pomiędzy drzwiami d_1 i d_2.

3.6.4. Realizacja procesu walidacji

Opisane w poprzednich rozdziałach formalizmy stanowią podstawę automatycznej walidacji projektu pod kątem wymagań projektowych. Jest to proces wywoływany po każdej akcji projektanta. Taka akcja pociąga za sobą zmianę w reprezentacji hipergrafowej projektu oraz ponowne przeprowadzenie ewaluacji zestawów testów dla zmodyfikowanego hipergrafu. Proces ten został przedstawiony we wstępie do rozdziału 3, w bieżącym podrozdziale omówione zostaną szczegółowo dwie jego fazy:

- stworzenie struktury relacyjnej,
- określenie wyników testów.

Na poniższym diagramie (Rysunek 3.9) przedstawione są kolejne kroki, składające się na wyżej wymienione fazy. Za pomocą strzałek dochodzących z prawej strony, pokazane są dane używane w kolejnych krokach.



Rysunek 3.9 Proces walidacji projektu w systemie HSSDR

Pierwszą z nich jest stworzenie nowej struktury relacyjnej oraz zasilenie jej danymi o projekcie. W tym kroku zostaje utworzony obiekt zawierający sygnaturę języka testów HSSDR. Wprowadzone do niego zostają następujące informacje, dla każdej relacji i funkcji wymienionych w podrozdziale 3.5.2:

- nazwa funkcji/relacji,

- liczba argumentów,
- typ operacji: relacja/funkcja.

Lista powyższych wartości jest stałą systemu.

Następnie tworzony jest obiekt struktury relacyjnej (pkt. 2). Dodawane są elementy składowe nośnika: pomieszczenia, drzwi oraz czujniki. W celu utworzenia zbiorów tych elementów przeszukiwany jest hipergraf projektu.

Z formalnego punktu widzenia do nośnika powinny także należeć dwie grupy zbiorów, które w proponowanej implementacji nie wchodzą jednak w jego skład. Są to etykiety elementów grafu oraz wszystkie wartości atrybutów, jakie mogą przyjmować składowe grafu. Te elementy mogą zostać pominięte w nośniku, pod warunkiem, że w formułach nie będzie możliwości, aby po nich przebiegały kwantyfikatory. W praktyce, wymagania przyjmują zwykle postać formuł typu „dla każdego pokoju …", nie ma natomiast potrzeby rozważać w ten sposób zbioru wszystkich możliwych etykiet czy długości ścian. Ten ostatni zbiór jest zresztą nieprzeliczalnym podzbiorem zbioru liczb rzeczywistych, więc komputerowa implementacja przeglądnięcia wszystkich jego elementów jest niemożliwa do zrealizowania. Dlatego wszystkie kwantyfikatory w formułach języka testów HSSDR muszą przebiegać po skończonych zbiorach.

Kolejnym krokiem (pkt. 3) jest uzupełnienie struktury relacyjnej o definicje relacji i funkcji, które wchodzą w skład sygnatury. W systemie HSSDR implementacja relacji oraz funkcji rozróżnia następujące warianty: tablicowe, dynamiczne, oraz dynamiczne z pamięcią podręczną. Zasady ich działania są scharakteryzowane poniżej.

W chwili tworzenia definicji relacji lub funkcji tablicowej następuje przeglądnięcie hipergrafu oraz odpowiednio:

- wyliczenie wszystkich elementów wchodzących w skład relacji

lub

- wyliczenie wszystkich elementów dziedziny funkcji, oraz obliczenie jej wartości dla każdego elementu.

Te informacje przechowywane są w obiekcie struktury relacyjnej pod postacią tablicy haszującej. W wariancie dynamicznym określenie wartości funkcji czy faktu należenia do relacji następuje w momencie zaistnienia takiej konieczności dla konkretnych argumentów w trakcie wartościowania formuł. Takie funkcje stosowane są dla kosztownych obliczeniowo operacji, jak np. obliczanie odległości między drzwiami, co przyspiesza proces walidacji.

Trzecia, ostatnia opcja, działa tak jak wariant dynamiczny, dodatkowo zapewniając, że dla konkretnych argumentów obliczenie wartości będzie miało miejsce co najwyżej raz, natomiast podczas kolejnych odwołań odczytana zapamiętana zostanie wartość.

W ramach następnego kroku system iteruje po wybranych przez użytkownika zestawach testów i dla każdego z nich realizuje operacje oznaczone na rysunku powyżej jako 4.1-4.4 (zestawy testów opisane są w podrozdziale 3.6.5).

Pierwszą z tych operacji jest parsowanie pliku (pkt 4.1). Jest ona realizowana za pomocą parsera wygenerowanego z użyciem biblioteki ANTLR [99], który dokonuje analizy leksykalnej i syntaktycznej zestawu testów. Parser jest wyposażony w zestaw instrukcji wykonywanych po napotkaniu każdego atomu leksykalnego. Za ich pomocą podczas parsowania tworzona jest reprezentacja drzewowa dla każdej formuły znajdującej się w pliku (pkt. 4.2) . Wchodzi ona w skład obiektu zawierającego:

- reprezentację drzewową formuły logiki pierwszego rzędu,
- listę użytych zmiennych,
- listę występujących w formule zmiennych wolnych,
- komunikaty o powodzeniu oraz niepowodzeniu testu.

W tej postaci wszystkie formuły w zestawie testów, a więc zdefiniowane testy oraz relacje pomocnicze, dodawane są do obiektu reprezentującego zestaw testów.

Dalej tworzony jest duplikat struktury relacyjnej (pkt 4.3), aby dodać do niej ewentualne pomocnicze relacje zdefiniowane w zestawie testów. Po analizie pliku duplikat będzie odrzucany, dzięki czemu definicje relacji pomocniczych z jednego zestawu testów nie wpływają na pozostałe.

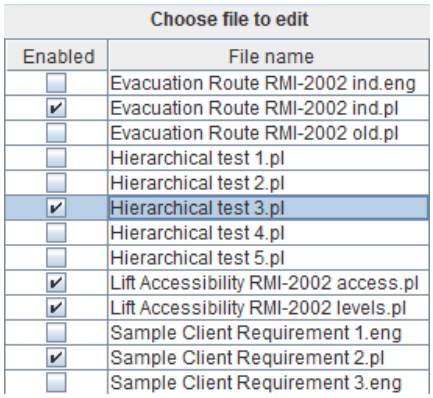
System, wyposażony w dane z poprzednich kroków, może wreszcie dokonać ewaluacji zestawu testów (pkt. 4.4). Ewaluacja jest procesem iteracyjnym wykonywanym dla każdej formuły wchodzącej w skład zestawu. Dla pojedynczej formuły, brana jest pod uwagę jej reprezentacja drzewowa. Za określanie wartości odpowiada metoda *value*, która jest wywoływana na korzeniu drzewa. Jest ona metodą realizującą postfiksowe przeszukiwanie drzewa (ang. *post-order*). Aby określić wartość węzła w drzewie, metoda *value* najpierw wywołuje metodę *value* dla jego dzieci, a następnie mając do dyspozycji zwrócone wartości określa wartość dla bieżącego węzła. Węzły reprezentują operacje pojawiające się w formułach: operatory logiczne, arytmetyczne, porównania, funkcje itd. Aby określić ich wartość konieczne jest korzystanie z odwołań do definicji z bieżącej struktury relacyjnej, a także określenie wartości użytych zmiennych. Dane te są dostarczone poprzez kontekst wartościowania formuły. Osobną kategorią operacji mogących się pojawiać w węzłach są kwantyfikatory. Ich wartościowanie wymaga odwołania do struktury relacyjnej, przeglądnięcia wszystkich obiektów wchodzących w zakres kwantyfikatora oraz określenie wartości formuły pod kwantyfikatorem, dla odpowiednio zmodyfikowanego kontekstu

wartościowania. Następnie, w zależności od rodzaju kwantyfikatora, dokonywana jest ostateczna agregacja rezultatów.

Ostatnim krokiem jest prezentacja wyników walidacji (pkt. 5), opisana dokładniej w podrozdziale 3.6.6. Na wyjściu procedury walidacji projektu znajdują się informacje o powodzeniu bądź niepowodzeniu testów oraz komunikaty o sukcesie bądź porażce. Dodatkowo, dla testów zaczynających się od kwantyfikatora ogólnego przekazywane są zidentyfikowane elementy projektu, dla których warunek objęty tym kwantyfikatorem nie jest spełniony.

3.6.5. Organizacja zestawów testów

Wymagania projektowe różnią się w zależności od projektu. Dlatego w proponowanym rozwiązaniu zostały one zorganizowane w zestawy testów, które następnie projektant może aktywować, dopasowując swój wybór do konkretnego projektu (Rysunek 3.10). Zestaw testów może zawierać jeden lub wiele powiązanych ze sobą testów oraz definicji pomocniczych. Użytkownik może dowolnie modyfikować zestawy testów lub dodawać nowe.



Rysunek 3.10 Wybór zestawów testów

Zestawy testów są zbiorami formuł języka testów HSSDR przechowywanymi w plikach. Mogą one zawierać formuły końcowe oraz definicje pomocnicze. W skład pojedynczego zestawu testów wchodzi:

- dowolna liczba definicji relacji pomocniczych (opisanych w rozdziale 3.5.2),
- dowolna liczba komentarzy w liniach rozpoczynających się od „#”,
- co najmniej jedna formuła końcowa, czyli formuła rozpoczynająca się od kwantyfikatora ogólnego lub egzystencjalnego oraz nie zawierająca zmiennych wolnych.

Należy pamiętać, że zgodnie z algorytmem opisanym w poprzednim rozdziale każdy plik analizowany jest oddzielnie. Oznacza to, że zdefiniowane w nim relacje pomocnicze są dostępne tylko w obrębie pojedynczego pliku.

3.6.6. Prezentacja wyników walidacji

Ostatnim krokiem walidacji projektu jest prezentacja wyników. Projektant jest informowany o tym, które z ograniczeń zostało naruszone, oraz które elementy projektu powodują brak zgodności. Może on następnie zmodyfikować projekt, aby usunąć niezgodności, albo też zignorować je wiedząc, że zostaną one usunięte w wyniku następnych akcji.

Częścią języka testów HSSDR (opisanego w rozdziale 3.5.2) są instrukcje pozwalające definiować wiadomości o powodzeniu lub niepowodzeniu danego warunku:

- *failure_msg* - wyświetlane w przypadku niepowodzenia testu,
- *success_msg* - wyświetlane w przypadku powodzenia testu.

Po słowach kluczowych następuje treść komunikatu ujęta w cudzysłowie. Te instrukcje mogą być umieszczane przed formułą języka testów. Po określeniu czy formuła jest w danej chwili spełniona, na konsoli walidacji projektu wyświetlana jest odpowiednia wiadomość. Instrukcje te są opcjonalne, w przypadku ich braku prezentowane są domyślne komunikaty o wynikach. Komunikaty zawsze zawierają odniesienie do zestawu testów z którego pochodzą. Tam z kolei, w komentarzach, twórca zestawu może umieścić wyjaśnienie przyczyn błędu, sugerowane akcje naprawcze oraz odniesienie do przepisów czy norm, na podstawie których został opracowany dany warunek.

Istotna częścią raportu o niepowodzeniu testów jest wskazanie fragmentu projektu niespełniającego kryteriów. W formułach reprezentujących końcowe warunki mogą występować tylko zmienne związane. Formuły te zaczynają się one od kwantyfikatorów, można więc wyróżnić dwa rodzaje niepowodzeń testów:

- pewne obiekty nie spełniają warunku objętego kwantyfikatorem ogólnym,
- nie istnieje obiekt spełniający warunek objęty kwantyfikatorem egzystencjalnym.

W przypadku tych pierwszych niepowodzeń, informacja o nieprawidłowym fragmencie projektu jest dołączana do komunikatu o błędzie. Wykorzystywana w tym celu jest etykieta przypisana przez użytkownika, natomiast w przypadku jej braku system generuje ciąg znaków zawierający rodzaj oraz położenie obiektu, pozwalający jednoznacznie zidentyfikować dany obiekt. Jeśli tym elementem są pomieszczenia, są one dodatkowo

podświetlane w edytorze. Jeśli przyczyną niezgodności jest brak elementu objętego kwantyfikatorem egzystencjalnym, komunikat o błędzie stanowi jedynie wiadomość o niepowodzeniu testu.

3.7. Przykłady testów zgodności

W bieżącym podrozdziale przedstawione zostały wybrane przykłady zestawów testów zgodności pokazujące możliwości sprawdzania wymagań projektowych przez HSSDR.

3.7.1. Wymagania hierarchiczne

Język testów HSSDR daje możliwość wnioskowania o logicznej strukturze projektu poprzez użycie warunków sprawdzających zagnieżdżanie składowych projektu. Niech jako przykład użycia tego mechanizmu posłużą poniższe warunki, które mogą mieć zastosowanie podczas projektowania restauracji, stołówek, urzędów czy innych obiektów, w których rozróżnia się dwie strefy, obszar dostępny dla ogółu interesantów oraz obszar dostępny wyłącznie dla pracowników:

1. Musi istnieć toaleta w obszarze publicznym.
2. Musi istnieć toaleta w obszarze prywatnym.

W celu zapisania powyższego warunku zdefiniujmy relację pomocniczą grupującą toalety męskie, damskie oraz dla osób niepełnosprawnych.

```
isToilet(x)  <=>
type(x)="ToiletMen" or type(x)="ToiletWomen" or type(x)="ToiledAccessible";
```

Warunki hierarchiczne zapisywane w języku testów HSSDR wymagają użycia relacji *isInDescendants* według schematu:

```
exists area in Areas: exists room in Rooms:
  isInDescendants(room, area)
```

Tak sformułowany warunek zakłada istnienie pewnego obszaru *area* oraz pewnego pokoju *room*, który znajduję się w drzewie hierarchii pod obszarem *area.* Dostępna jest również relacja *isDirectChild*, mówiąca o bezpośrednim zagnieżdżeniu w drzewie hierarchii.

Stosując powyższą konstrukcje do zadanych warunków, oraz z korzystając z relacji pomocniczej, możemy zapisać następujące testy:

```
isToilet(x)  <=>
type(x)="ToiletMen" or type(x)="ToiletWomen" or type(x)="ToiledAccessible";

failure_msg „Brak toalety w sekcji prywatnej”
exists a in Areas: exists r in Rooms:
  type(a) = "Private" and isToilet(r)  and isInDescendants(r,a);

failure_msg “Brak toalety w sekcji publicznej
exists a in Areas: exists r in Rooms:
  type(a) = "Public" and isToilet(r)  and isInDescendants(r,a);
```

3.7.2. Normy przeciwpożarowe

Jako przykład testów sprawdzających zgodność rozkładu pomieszczeń z normami budowlanymi niech posłużą ograniczenia przeciwpożarowe. Polskie normy prawne precyzują warunki jakie muszą spełniać budynki przeznaczone na pobyt ludzi pod względem ochrony przeciwpożarowej. Dotyczą one między innymi dróg ewakuacyjnych, stref pożarowych, odporności pożarowej, usytuowania budynków. Obowiązującym aktem prawnym jest obecnie „Rozporządzenie Ministra Infrastruktury w sprawie warunków technicznych, jakim powinny odpowiadać budynki i ich usytuowanie” z dnia 12 kwietnia 2002 (z późniejszymi zmianami, tekst ujednolicony z 18 września 2015) [102]. W dalszej części przedstawione zostaną testy sprawdzające dwa z ograniczeń narzucanych przez to rozporządzenie, odnoszące się do dróg ewakuacyjnych oraz przejść ewakuacyjnych.

Regulacje prawne

Pojęcie *drogi ewakuacyjnej* określone jest w następujący sposób:

(§ 236. ust 1.) *„Z pomieszczeń przeznaczonych na pobyt ludzi powinna być zapewniona możliwość ewakuacji w bezpieczne miejsce na zewnątrz budynku lub do sąsiedniej strefy pożarowej, bezpośrednio albo drogami komunikacji ogólnej, zwanymi dalej „drogami ewakuacyjnymi” ”*

Określone są także parametry korytarzy, przez które może taka droga przebiegać, a także drzwi, które na tej drodze mogą leżeć. Dojście do klatki schodowej, która spełnia określone właściwości jest uważane za równorzędnie dojściu do innej strefy pożarowej (§ 236. ust 2.). Maksymalna długość drogi jest ograniczona, w zależności od rodzaju budynku. Dla pomieszczeń biurowych jest to 30 metrów (§ 236. ust 3.).

Przejście ewakuacyjne jest scharakteryzowane w następujący sposób:

(§ 237. ust 1.)*„W pomieszczeniach, od najdalszego miejsca, w którym może przebywać człowiek, do wyjścia ewakuacyjnego na drogę ewakuacyjną lub do innej strefy pożarowej albo na zewnątrz budynku, powinno być zapewnione przejście, zwane dalej „przejściem ewakuacyjnym”(...)”*

Parametry przejścia ewakuacyjnego również są poddane pewnym ograniczeniom. Zgodnie z jednym z nich (§ 237. ust 8) *„Przejście, o którym mowa w ust. 1, nie powinno prowadzić łącznie przez więcej niż trzy pomieszczenia*”

Zapis ograniczeń w języku logiki pierwszego rzędu

W dalszej części przedstawiony jest zapis następujących ograniczeń:

1. Droga ewakuacyjna prowadząca do klatki schodowej, lub wyjścia z budynku musi mieć mniej niż 30 metrów
2. Z każdego punktu budynku dostęp do drogi ewakuacyjnej musi prowadzić przez trzy lub mniej pomieszczeń.

Stworzenie testów dla powyższych ograniczeń wymaga zdefiniowania pewnych pomocniczych konstrukcji. Będą to funkcje i relacje niezbędne dla określenia jakie właściwości charakteryzują taką drogę i które pomieszczenia wchodzą w jej skład. Poniżej przedstawione zostaną kroki konieczne do stworzenia testów.

Na początku trzeba wyróżnić drzwi, które kończą drogę ewakuacyjną. Zgodnie z przepisami są to wyjścia z budynku, a także drzwi prowadzące na klatki schodowe:

```
# zbiór drzwi prowadzących do innych stref pożarowych
ZoneDoors(d) <=> exists r in Rooms:
    doorsInRoom(d, r) and  type(r) = “Staircase”;

# zbiór drzwi zapewniających opuszczenie bieżącej strefy
FireSafetyDoors(d) <=>  ExternalDoors(d) or ZoneDoors(d);
```

Następnie klasyfikujemy pomieszczenia ze względu na ich typ w kontekście dróg pożarowych. Wyróżniamy następujące typy:

1. bezpieczne wyjścia: klatki schodowe,
2. wspólne obszary komunikacyjne,
3. pozostałe pomieszczenia.

Klasyfikacji dokonuje następujący kod:

```
# pokoje stanowiące odrębne strefy pożarowe
ZoneRooms(r) <=> Rooms(r) and (type(r) = „Staircase”);

# pomieszczenia które mogą wchodzić w skład drogi ewakuacyjnej
EvacRoute(r) <=> Rooms(r) and (type(r) = “Corridor” or type(r) = “Hall”);

# pomieszczenia które mogą wchodzić w skład przejścia ewakuacyjnego
AccessPath(r) <=> Rooms(r) and !ZoneRooms(r) and !EvacRoute(r);
```

Skupiając się na pierwszym ograniczeniu określmy drzwi, które prowadzą do prawidłowej drogi ewakuacyjnej. Zdefiniujmy relację validRouteFromAdj, do której będą należeć drzwi oddalone o odległość *x* metrów oraz o jeden pokój typu *EvacRoute* od końcowych drzwi drogi ewakuacyjnej.

```
# określa drzwi które są oddalone o co najwyżej x od FireSafetyDoors,
#oddzielone od nich pokojem EvacRoute

validRouteFromAdj(d, x) <=>  Doors(d)  and
 ( exists d2 in Doors : d2!=d and FireSafetyDoors(d2) and
   ( exists p in Rooms:
         doorsInRoom(d2,p) and doorsInRoom(d,p) and EvacRoute(p)
         and x>=doorsDist(d, d2)
   )
 );
```

Następnie, poprzez indukcję, zdefiniujmy zbiór drzwi oddalonych o dystans co najwyżej *x* metrów od *FireSafetyDoors,* ale sąsiadujących z nimi poprzez *n* lub mniej pokoi, gdzie *n* to liczba naturalna.

Druga składowa alternatywy zapewnia nam redukcje czynnika *n* bez zwiększania liczby sąsiadujących pokoi. Dzięki warunkowi:

$$validRouteFromInduction(d, x, n) \Rightarrow validRouteFromInduction(d, x, n + k),$$

gdzie k jest dowolną liczbą naturalną. Istnienie tego warunku jest wymuszone przez specyfikę mechanizmów ewaluacji formuł. W czysto abstrakcyjnym zapisie moglibyśmy sobie zażyczyć, że: „istnieje takie n należące do zbioru liczb naturalnych”. Natomiast w języku testów HSSDR takie użycie kwantyfikatora nie jest możliwe.

```
# określa drzwi które są oddalone o co najwyżej x od FireSafetyDoors,
# oddzielone od nich co najwyżej n pokojami EvacRoute

validRouteFromInduction(d, x, n)<=>
 (n = 1 and validRouteFromAdj(d, x) ) or
 (n <> 1 and Doors(d) and validRouteFromInduction(d, x, n-1)) or
```

```
(n <> 1 and Doors(d) and exists d2 in Doors :  d2!=d and
        ( exists p in Rooms:
            doorsInRoom(d2,p) and doorsInRoom(d,p) andEvacRoute(p)
            and x>=doorsDist(d, d2)
              )
        and validRouteFromInduction(d2, x - doorsDist(d, d2), n-1)
);
```

Korzystając z wcześniejszych definicji, możemy ograniczyć argument x do wartości 30 metrów zadanej w warunku, oraz parametr n sterujący indukcją do „dowolnie dużej", rozsądnej dla rozkładów pomieszczeń wartości.

```
# określa drzwi które są oddalone o co najwyżej x od FireSafetyDoors,
# oddzielone od nich co dowolną liczbą pokoi EvacRoute
validRouteFrom(d, x) <=> validRouteFromInduction(d, x, 99);

# drzwi wchodzące w skład prawidłowej drogi ewakuacyjnej
ValidEvacRouteDoors(d) <=>
    FireSafetyDoors(d) or  validRouteFrom(d, 30);
```

Ostatecznie poniższa relacja definiuje pokoje leżące na prawidłowej drodze ewakuacyjnej:

```
# pokoje wchodzące w skład prawidłowej drogi ewakuacyjnej
ValidEvacRouteRooms(r) <=> EvacRoute(r) and
   ( exists d in ValidEvacRouteDoors: doorsInRoom(d, r) );
```

Przejdźmy do drugiego ograniczenia mówiącego, że dla każdego pokoju droga ewakuacyjna musi być dostępna bezpośrednio, bądź za pomocą przejścia ewakuacyjnego prowadzącego przez nie więcej niż trzy pokoje. Korzystając z wyżej zdefiniowanych relacji, dostęp do drogi ewakuacyjnej jest równoważny dostępowi do drzwi należących do *ValidEvacRouteDoors.* Zdefiniujmy najpierw pokoje leżące bezpośrednio przy drodze ewakuacyjnej:

```
# zbiór pokoi użytkowych posiadających drzwi na drogę ewakuacyjną
steps0(r) <=>  AccessPath(r)  and
    ( exists d in ValidEvacRouteDoors: doorsInRoom(d, r) );
```

Funkcje *steps1* oraz *steps2* określą zbiory pomieszczeń leżących w odległości mniejszej lub równej niż odpowiednio jeden i dwa pokoje od drogi ewakuacyjnej:

```
# zbiór pokoi użytkowych sąsiadujących ze steps0
steps1(r) <=> steps0(r) or
                ( AccessPath(r)  and (exists r2 in steps0: accessible(r, r2) ) );

# zbiór pokoi użytkowych sąsiadujących ze steps1
steps2(r) <=> steps1(r) or
                ( AccessPath(r)  and (exists r2 in steps1: accessible(r, r2) ));
```

Wyposażeni w powyższe funkcje możemy zdefiniować predykat ValidRooms, który sprawdza czy dane pomieszczenie spełnia obydwa ograniczenia. Takie pomieszczenie musi być albo bezpiecznym wyjściem, wchodzić w skład poprawnej drogi ewakuacyjnej, albo posiadać poprawne przejście ewakuacyjne prowadzące na drogę ewakuacyjną:

```
# suma zbiorów pokoi:
# stanowiących odrębne strefy pożarowe
# sąsiadujących z droga ewakuacyjną przez mniej niż trzy pomieszczenia
# wchodzących w skład prawidłowej drogi ewakuacyjnej
ValidRooms(r) <=> ZoneRooms(r) or ValidEvacRouteRooms(r) or steps2(r);
```

Ten predykat jest ostatecznie użyty do sprawdzenia poprawności rozkładu pomieszczeń pod względem zgodności z przepisami przeciw pożarowymi w dwóch opisanych punktach:

```
failure_msg „pewne pokoje nie spełniają ograniczeń przeciwpożarowych”
forall r in Rooms: ValidRooms(r)
```

3.7.3. Czujniki bezpieczeństwa

Obok konkretnych, materialnych elementów składowych projektów istotną rolę mogą odgrywać również artefakty przestrzenne (ang. spatial artefacts). W pracach [103] [104] zostały wyróżnione następujące klasy artefaktów przestrzennych:

- przestrzeń operacyjna (ang. operational space), oznaczającą obszar niezbędny dla danego artefaktu do spełniania jego funkcji, np. obszar zajmowany przez drzwi podczas ich otwierania,
- przestrzeń funkcjonalna (ang. functional space), oznaczająca niezbędny obszar do wejścia w interakcje z danym obiektem, np. przestrzeń potrzebna dla człowieka, aby mógł otworzyć i przejść przez drzwi,
- zasięg (ang. range space), oznaczający obszar objęty zasięgiem czujnika, takiego jak czujnik ruchu, kamery przemysłowe, czujnik dymu, poziomu dwutlenku węgla lub spalin, temperatury, itp.

Artefakty przestrzenne dostarczają wiedzy o ograniczeniach funkcjonalnych pewnych elementów projektu. Ograniczenia te, w połączeniu z ograniczeniami obiektów czysto fizycznych, mogą mieć duże znaczenie dla projektu, natomiast nie są one uwzględniane w narzędziach CAD, które zawierają tylko fizycznie istniejące obiekty [75].

W systemie HSSDR została zawarta możliwość dodawania artefaktów przestrzennych w postaci zasięgu czujnika bezpieczeństwa [75]. Interfejs użytkownika pozwala na umieszczanie czujnika bezpieczeństwa w pomieszczeniach. Język testów został wzbogacony o zbiór czujników (*Sensors*), a także o relacje mówiące o lokalizacji czujnika (sensorInRoom), o tym czy drzwi są objęte zasięgiem czujnika (*surveilledDoors*) oraz czy istnieje ścieżka pomiędzy parą drzwi omijająca zasięg wszystkich czujników w danym pomieszczeniu (*isPassageWatched*).

Przykładem testu zgodności korzystającego z powyższych konstrukcji niech będzie zapis następującego warunku:

Wszystkie ścieżki pomiędzy pomieszczeniami wymagającymi nadzoru, a drzwiami na zewnątrz budynku powinny być objęte zasięgiem czujników bezpieczeństwa.

W celu zapisania powyższego warunku określmy, które pomieszczenia wymagają nadzoru. Niech będą to następujące typy pomieszczeń:

```
#Pomieszczenia wymagające nadzoru
SecuredArea(r) <=> Rooms(r) and (type(r) = "Office" or type(r) = "Technical"
                        or type(r) = "Storage" or type(r) = "Server_Room");
```

Następnie zdefiniujmy relacje *UnwatchedPassage* mówiącą o istnieniu przejścia między parą drzwi w obrębie jednego pomieszczenia, które omija zasięg czujników bezpieczeństwa:

```
UnwatchedPassage(d1, d2) <=> d1!=d2 and  Doors(d1) and  Doors(d2)
        and (exists r in Rooms: doorsInRoom(d1,r) and doorsInRoom(d2,r)
                and not isPassageWatched(d1,d2)
         );
```

Kolejnym krokiem jest stworzenie domknięcia przechodniego tej relacji. W tym celu zastosowany jest mechanizm rekurencji:

```
UnwatchedPassageInduction(d1, d2, n) <=>
 (n = 1 and UnwatchedPassage(d1, d2)) or
 (n <> 1 and UnwatchedPassageInduction(d1, d2, n-1)) or
 (n <> 1 and exists d in Doors:
  UnwatchedPassageInduction(d1, d, n-1) and
  UnwatchedPassage(d, d2));

#Możliwe przejście pomiędzy d1 i d2 omijające zasięg czujników
UnwatchedPassagePlus(d1, d2) <=> UnwatchedPassageInduction(d1, d2, 99);
```

Mając do dyspozycji powyższe relacje pomocnicze, można określić końcowy warunek:

```
success_msg "przejścia zabezpieczone"
failure_msg "istnieje przejście niezabezpieczone"
forall r in Rooms:  forall d1, d2 in Doors:
not (
      ExternalDoors(d1)  and  SecuredArea(r) and doorsInRoom(d2,r)
      and UnwatchedPassagePlus(d1,d2)
);
```

4. Rozszerzenie systemu na budynki wielopiętrowe

Wersja systemu HSSDR omówiona w poprzednim rozdziale ograniczała się do wspomagania projektowania płaskich rozkładów pomieszczeń. Obecnie wprowadzone modyfikacje umożliwiają projektowanie budynków wielopiętrowych. Można mówić w tym przypadku o języku wizualnym 2.5D [76]. Język ten pozwala projektantowi spojrzeć na wiele pięter budynku jednocześnie, a dzięki temu dostrzec wzajemne relacje przestrzenne pomiędzy pomieszczeniami na różnych poziomach. Możliwe jest także wprowadzanie relacji występujących pomiędzy pomieszczeniami na różnych piętrach takich jak dostępność czy widoczność. Zmiana pomiędzy widokiem poszczególnych pięter a widokiem wielu pięter umożliwia projektantowi wykonanie akcji percepcyjnych, które mogły być trudniejsze przy ograniczeniu do widoków poszczególnych pięter. W dalszej części rozdziału opisany jest sposób realizacji zmian na potrzeby języka 2.5D oraz zaprezentowane są przykładowe testy zgodności weryfikujące warunki dla wielu pięter.

4.1. Modyfikacje na potrzeby budynków wielopiętrowych

Poniżej wymienione są modyfikację struktury hipergrafowej (rozdział 3.3.1) oraz języka testów (rozdział 3.6.3) na potrzeby wielopiętrowości w stosunku do stanu opisanego w poprzednich rozdziałach.

- Wprowadzenie węzła oznaczającego połączenia pionowe dla pomieszczenia. W tym celu rozszerzony zostaje zbiór etykiet funkcji atrybutowania dir_G, oznaczającej orientacje ściany reprezentowanej przez dany wierzchołek. Dodana zostaje nowa etykieta *„V”* reprezentujące połączenia pionowe.
- Dodanie relacji między piętrami. Podczas dodawania relacji między piętrami, wykrywane są pomieszczenia pomiędzy którymi zachodzi relacja. Dla reprezentujących je hiperkrawędzi obiektowych, węzły reprezentujące ich połączenia pionowe dodawane są do nowo tworzonej relacji. Możliwe jest dodawanie połączeń między dwoma piętrami, jak również połączeń wielopiętrowych reprezentujących windy czy wielopiętrowe klatki schodowe. W przypadku tych ostatnich zostanie stworzona wieloargumentowa relacja, gdzie liczba argumentów będzie równa liczbie pięter wchodzącej w skład ciągu komunikacyjnego.
- W przypadku dzielenia pomieszczeń wchodzących w skład relacji między piętrami, istotne jest które z nowo powstałych pomieszczeń wejdzie w skład relacji po podziale. Dlatego punkt zaczepienia relacji między piętrowej jest przechowywany jako atrybut relacji. Do procedury podziału pomieszczenia opisanej w rozdziale 3.4.2 dodawany

jest krok przepinający połączenie hiperkrawędzi relacyjnej do odpowiedniej hiperkrawędzi obiektowej.

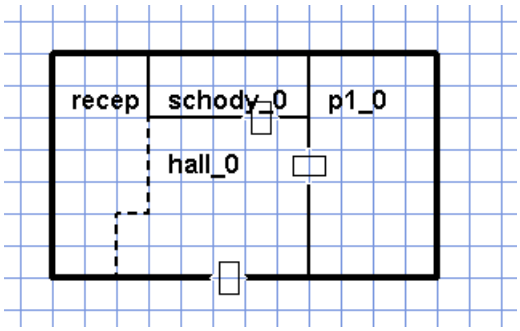
- Informacja o piętrach oddawana jest jako nowy atrybut hiperkrawędzi obiektowej:

$floorNr: E_G^C \to \mathbb{N}$, dla hiperkrawędzi obiektowej, numer piętra na którym się znajduje pomieszczenie reprezentowane przez tą hiperkrawędź.

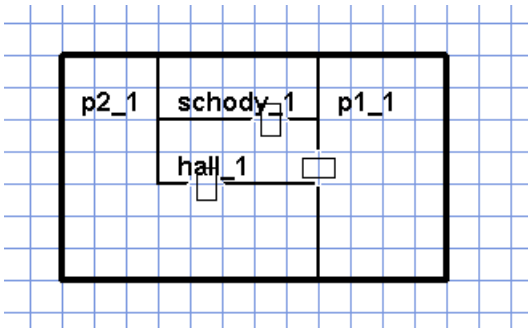
- Do języka testów dodawany jest symbol relacji jednoargumentowej $Floors$, definiującej zbiór pięter w budynku, oraz symbol funkcji $floorNr$, która dla pomieszczenia zwraca atrybut opisany w poprzednim punkcie.

Warto zaznaczyć, że chociaż to rozszerzenie powiększa obszar jaki może być poddany analizie, to struktura reprezentująca budynek nie zmienia się zasadniczo. Graf reprezentujący budynek wielopiętrowy można również interpretować jako „płaski" graf gdzie kolejne piętra sąsiadują są połączone specyficznymi hiperkrawędziami obiektowymi reprezentującymi klatki schodowe czy windy.

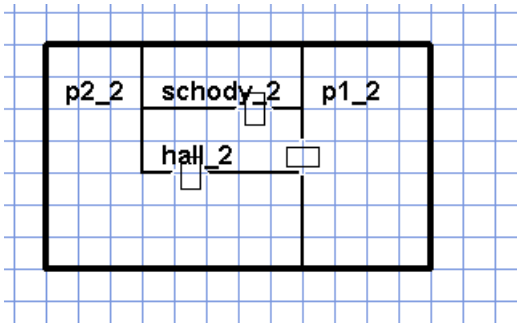
Dla zobrazowania tych zmian poniżej znajduje się przykład uproszczonego (dla przejrzystości prezentacji) trzypoziomowego budynku biurowego. Parter składa się z recepcji, hallu, klatki schodowej i jednego pomieszczenia biurowego, natomiast pozostałe piętra z klatki schodowej, hallu i dwóch pomieszczeń biurowych (Rysunek 4.1-Rysunek 4.3).



Rysunek 4.1 Przykładowy budynek - poziom 0

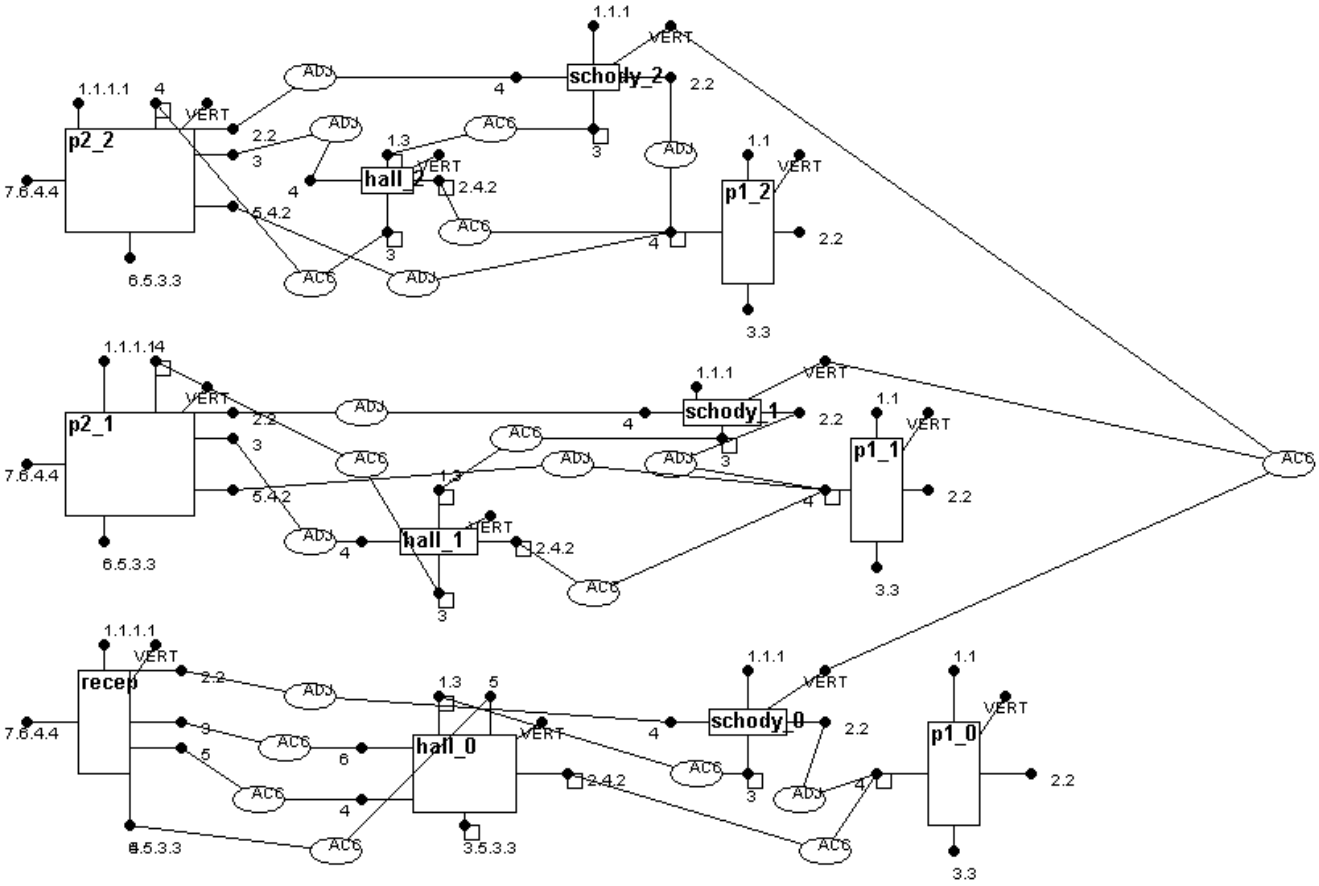


Rysunek 4.2 Przykładowy budynek - poziom 1



Rysunek 4.3 Przykładowy budynek - poziom 3

Odpowiadający temu budynkowi hipergraf wygenerowany przez system HSSDR znajduje się na rysunku Rysunek 4.4. Łatwo zauważyć grupy powiązanych ze sobą elementów odpowiadających trzem poziomom budynku. Jedyny łącznik między poziomami stanowi 3-argumentowa relacja dostępności pomiędzy klatkami schodowymi na wszystkich piętrach widoczna po prawej stronie zrzutu ekranu.



Rysunek 4.4 Hipergraf trzypoziomowego budynku

4.2. Przykłady testów zgodności dla wielu pięter

Jako przykład testów zgodności dla budynków wielopiętrowych niech posłużą wymagania odnośnie wind pochodzące z „Rozporządzenia Ministra Infrastruktury w sprawie warunków technicznych, jakim powinny odpowiadać budynki i ich usytuowanie” oraz z wytycznych „Projektowanie bez barier” [105].

W pierwszej kolejności warto zapisać w języku testów następujący warunek pomocniczy, określający czy winda może być traktowana jako nadającą się do użycia podczas komunikacji [105]:

„Winda towarowa nie powinna być traktowana jako część drogi dostępu, chyba że jedynym zastosowanym w obiekcie typem windy są windy osobowo-towarowe.”

W systemie HSSDR dostępne są trzy typy wind: winda osobowa spełniająca wymagania dostępności dla osób niepełnosprawnych (LiftConform), winda osobowa niespełniająca tych wymagań (LiftNonConform) oraz winda towarowa (LiftCargo). Dwa następujące warunki pomocnicze określają dowolna windę (*isLift*) oraz windę dozwoloną dla komunikacji w świetle wyżej podanych przepisów (*AllowedForDisabled*):

```
# Kazda winda
isLift(x)<=> type(x)="LiftConform" or type(x)="LiftNonConform" or type(x)="LiftCargo";

# Dozwolona winda
AllowedForDisabled(p) <=> type(p)="LiftConform"
or ( type(p)="LiftCargo" and not exists x in Rooms:
   type(x)="LiftConform" or type(x)="LiftNonConform" );
```

Dodatkowo, aby wnioskować o budynkach „wyposażonych w dźwigi”, warto zdefiniować następującą formułę:

```
#Budynek bez windy
BuildingWithoutLift()<=> not exists p in Rooms: isLift(p);
```

Korzystając z tych pomocniczych relacji można łatwiej zapisać następujące warunki, które przepisy nakładają na budynki wyposażone w windy:

(§ 54. 2.) „W budynku mieszkalnym wielorodzinnym, budynku zamieszkania zbiorowego oraz budynku użyteczności publicznej, wyposażanym w dźwigi, należy zapewnić dojazd z poziomu terenu i dostęp na wszystkie kondygnacje użytkowe osobom niepełnosprawnym.” [102]

(§ 194. 1.) „Dostęp do dźwigu powinien być zapewniony z każdej kondygnacji użytkowej.” [102]

Można by tutaj wywnioskować, że jeżeli winda na podstawie pierwszego warunku musi dojeżdżać na wszystkie kondygnacje to w szczególności musi dojeżdżać również na parter. Można jednak wyobrazić sobie budynek w którym na parterze nie ma pomieszczeń lub są wyłączone z użytkowania. Istotne w takich przypadkach jest również to, aby ograniczać do minimum interpretację przepisów i pozostawić ją mechanizmom wnioskującym lub ekspertom. Zapis warunków w języku testów przedstawiony jest poniżej.

```
failure_msg "brak dostepu do windy z poziomu parteru"
success_msg "dostep do windy z poziomu parteru-ok"
BuildingWithoutLift();
```

```
or
( exists lift in Rooms: AllowedForDisabled(lift)
  and floorNr(lift)=0 );

failure_msg "brak windy na wszystkich pietrach"
success_msg "winda na wszystkich pietrach-ok"
BuildingWithoutLift()
or
( forall n in Floors: exists lift in Rooms:
 AllowedForDisabled(lift) and floorNr(lift)=n);
```

Inny paragraf, precyzuje dodatkowo:

(§ 194. 1.) „Co najmniej jeden z dźwigów służących komunikacji ogólnej w budynku z pomieszczeniami przeznaczonymi na pobyt ludzi, a także w każdej wydzielonej w pionie, odrębnej części (segmencie) takiego budynku, powinien być przystosowany do przewozu mebli, chorych na noszach i osób niepełnosprawnych”

Mówi on, że odpowiednia winda powinna być w każdym odrębnym segmencie budynku, a więc któraś z wind powinna być dostępna z każdego pomieszczenia na danym poziomie. Warunek ten, wyrażony w logice pierwszego rzędu brzmi: dla każdego pomieszczenia, musi istnieć znajdująca się na tym samym piętrze winda, dostępna poprzez ścieżkę w ramach tego samego piętra. Aby zapisać warunek dostępności poprzez ścieżkę w ramach tego samego piętra należy stworzyć pomocniczą definicję rekurencyjną *AccessiblePathFloorInduction*. Zapis całego testu przedstawiony jest poniżej:

```
AccessiblePathFloorInduction(p1, p2, n) <=> floorNr(p1)=floorNr(p2) and
( (n = 1 and accessible(p1, p2)) or
 (n <> 1 and AccessiblePathFloorInduction(p1, p2, n-1)) or
 (n <> 1 and exists p3 in Rooms:
   floorNr(p1)=floorNr(p3) and
   AccessiblePathFloorInduction(p1, p3, n-1) and
   accessible(p3, p2)) );

# Możliwe przejście z p1 do p2 przez dowlną liczbe pomieszczen na pietrze
AccessiblePathFloor(p1, p2) <=> AccessiblePathFloorInduction(p1, p2, 99);

failure_msg "brak dostepu do windy z kazdej czesci bunynku"
success_msg "dostep do windy z kazdej czesci bunynku-ok"
BuildingWithoutLift()
or
forall r in Rooms: exists lift in Rooms:
      floorNr(r)=floorNr(lift) and AllowedForDisabled(lift) and AccessiblePathFloor(lift,r);
```

5. Udostępnienie mechanizmów walidacji projektu dla innych narzędzi CAD

Autorzy pracy [11] podkreślają potrzebę stworzenia uniwersalnego języka wymagań projektowych. Jako jedną z zalet takiego języka przedstawiają możliwość użycia go w różnych narzędziach CAD, a także w implementacji serwerowej. Z kolei autorzy pracy [3] koncentrują się na zaletach tych ostatnich - aplikacji serwerowych dyktowanych do walidacji projektów online. Według nich, z uwagi na częstość zmian wymagań projektowych, a także regulacji prawnych łatwiej uaktualniać i utrzymywać centralną bazę wymagań projektowych w wersji serwerowej niż wiele wersji lokalnych. Na podstawie swoich prób stwierdzili oni również, że sprawdzanie wymagań wewnątrz narzędzia ArchiCAD [106] byłoby trudne. Koszty takiego rozwiązania byłyby dodatkowo zwielokrotnione jeśli platforma miałaby wspierać różne narzędzia CAD. Doświadczenia autora tej pracy potwierdzają opinie pochodzące z drugiej z przytoczonych prac. Istotnie implementacja złożonych algorytmów walidacji projektów wewnątrz narzędzi CAD jest trudnym i czasochłonnym zadaniem.

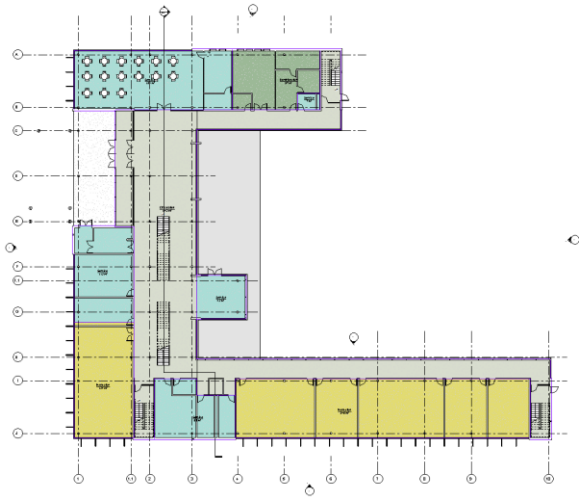
Niemniej jednak udostępnienie mechanizmów walidacji projektu HSSDR dla innych narzędzi CAD jest kuszącą perspektywą. Mechanizmy te mogą zostać zastosowane również dla projektów spoza jego edytora rozkładów pomieszczeń. Biblioteka testów może służyć jako baza wiedzy przechowywana na serwerze i być udostępniana przez sieć wielu klientom, co ułatwia jej organizacje, aktualizacje oraz udostępnianie.

Podejście to nie zostało całkowicie zrealizowane, jednak zostały podjęte próby w tym kierunku, które potwierdziły możliwość wyabstrahowania modułów walidacji projektu i stworzenia z nich odrębnego programu weryfikującego ograniczenia projektowe. Moduły te zostały użyte dla analizy projektów pochodzących z Revit Architecture oraz SketchUp. Na potrzeby komunikacji tych aplikacji z HSSDR zostały stworzone odpowiednie pluginy, które są niewielkimi programami działającymi w środowisku danego narzędzia CAD. HSSDR Server jest programem działającym w architekturze klient-serwer, służącym jako baza wymagań i narzędzie do ich weryfikacji dla innych systemów CAD, który został użyty do komunikacji z programem SketchUp.

5.1. Testowanie projektów Revit Architecture

Przykładem narzędzia BIM dla którego zastosowano mechanizmy walidacji HSSDR jest przedstawiony w rozdziale 2.2 Revit Architecture. Revit umożliwia tworzenie pluginów w języku C# [107] należącym do platformy .NET [108]. W tym języku został stworzony plugin HSSDR, który gromadzi dane o projekcie. Plugin ten jest w stanie zebrać informacje potrzebne do wartościowania większości relacji i funkcji języka HSSDR. Nie przesyła on natomiast informacji o hierarchii pomieszczeń, piętrach oraz czujnikach bezpieczeństwa.

Zebrane informacje są wystarczające do sprawdzenia projektu pod kątem norm przeciwpożarowych omówionych w rozdziale 3.7.2. Testy te zostały wykonane z powodzeniem na udostępnionym przez Revit projekcie trzypoziomowego budynku biurowego zawierającego 91 pomieszczeń oraz 100 drzwi (Rysunek 5.1).



Rysunek 5.1 Projekt budynku biurowego w Revit Architecture

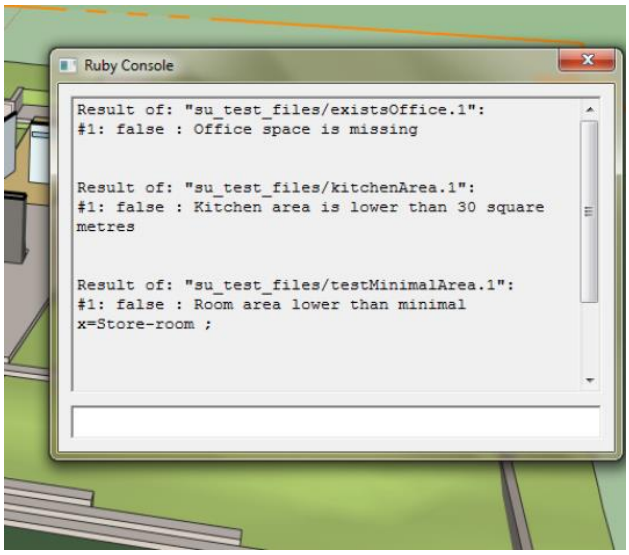
5.2. HSSDR Server

HSSDR Server jest programem napisanym w jawie, który pod określonym adresem IP i numerem portu udostępnia usługę sieciową walidacji projektu. Z tej usługi mogą korzystać narzędzia CAD wyposażone w odpowiednie pluginy HSSDR. Zadania pluginów są ograniczone do:

- zebrania informacji niezbędnych do analizy projektu,
- nawiązania połączenie z serwerem HSSDR,
- wysłania danych projektu oraz odebranie wyniku walidacji,
- prezentacji wyników testów.

Dzięki takiego ograniczeniu funkcjonalności pluginów koszty ich tworzenia są ograniczone do minimum.

Podejście to zostało zaimplementowane w komunikacji z programem SketchUp [26], omówionym w rozdziale 2.1.1. SketchUp udostępnia możliwość tworzenia pluginów w języku programowania Ruby [109]. Korzystając z tej technologii został stworzony plugin HSSDR działający w środowisku SketchUp. Plugin ten realizuje przedstawione wyżej operacje dotyczące komunikacji pomiędzy obydwoma środowiskami. Rezultaty testów prezentowane są na konsoli w programie SketchUp (Rysunek 5.2).



Rysunek 5.2 Wyniki testów na konsoli SketchUp

W tej implementacji gromadzony i przesyłany do analizy zasób informacji o projekcie, jest ograniczony do informacji o pomieszczeniach: ich nazw, wymiarów, oraz powierzchni. Jest to spowodowane ograniczeniami środowiska SketchUp jakie istniały w momencie tworzenia pluginu HSSDR. Natomiast w ostatnim czasie pojawiły się w SketchUp nowe funkcjonalności pozwalające na oznaczanie fragmentów projektu klasami IFC, które mogą usunąć te ograniczenia.

6. Podsumowanie

Badania, których wyniki są opisane w niniejszej rozprawie były wykonywane w latach 2008-2015. W ramach tych badań zostały opracowane zarówno model teoretyczny systemu HSSDR jak i jego implementacja.

HSSDR jest narzędziem projektowania rozkładów pomieszczeń, stworzonym z myślą o wspomaganiu fazy wstępnej projektowania architektonicznego. Projektant porozumiewa się z systemem za pomocą języka wizualnego diagramów stosowanego w architekturze, a także formułując wymagania projektowe z użyciem języka testów HSSDR. Wewnętrzna reprezentacja projektu jest definiowana w postaci hipergrafowej struktury danych będącej hierarchicznym atrybutowanym hipergrafem. Dane zawarte w hipergrafie hierarchicznym, a także wartości funkcji i relacji niezawarte bezpośrednio w opisie projektu ale obliczone na jego podstawie, są przekazywane na potrzeby wnioskowania do struktury relacyjnej. Struktura ta umożliwia wartościowanie formuł języka testów HSSDR.

System HSSDR jest symbolicznym, deklaratywnym systemem opartym na wiedzy. Bazuje on na zastosowaniu logiki pierwszego rzędu. Rolę bazy wiedzy pełnią testy zgodności, zapisane w języku testów HSSDR. Ewaluator formuł logiki pierwszego rzędu posiada parser języka testów HSSDR, który dokonuje analizy leksykalnej i syntaktycznej zawartości zestawów testów. Moduły walidacji projektu z założenia działają w tle, sugerując projektantowi działania w przypadku, gdy projekt nie jest zgodny z aktualnie wybranymi ograniczeniami. Brane pod uwagę formuły wybierane są przez projektanta spośród dostępnych zestawów testów. W przypadku wykrycia naruszenia warunków zapisanych w testach, projektant jest o tym powiadamiany poprzez informacje tekstową lub poprzez wyróżnienie niespełniających wymagań fragmentów projektu.

Implementację systemu HSSDR można traktować jako prototypowe narzędzie wspomagające projektowanie. Nie można co prawda mówić o kompletnym narzędziu gotowym do użycia w dowolnych projektach, jednak wynika to wyłącznie z faktu, że nakład pracy związany z tworzeniem tego typu systemów jest znaczny nawet w przypadku prototypów (np. system HSSDR to ok. 17,5 tysiąca linii kodu).

System HSSDR pozwala na tworzenie złożonych rozkładów pomieszczeń. Możliwe jest również projektowanie budynków wielopiętrowych. Język testów HSSDR stanowi ogólny, abstrakcyjny język wyższego poziomu, pozwalający formułować wymagania projektowe. Formalizacja wiedzy projektowej, która z wiedzy ukrytej staje się sformalizowaną wiedzą wyrażoną explicite w postaci reguł, stwarzając możliwość wielokrotnego jej wykorzystywania, jest wartościową techniką w projektowaniu. Język testów HSSDR został zastosowany do zapisu szeregu wymagań projektowych, w tym złożonych norm prawnych dotyczących ograniczeń przeciwpożarowych pochodzących z polskiego prawa budowlanego.

Fakt stworzenia powyższego systemu potwierdza więc postawioną na początku niniejszej rozprawy tezę:

Istnieje możliwość wspomagania projektowania architektonicznego w fazie wstępnej poprzez automatyczne pozyskiwanie wiedzy projektowej oraz sprawdzanie zgodności projektu z wymaganiami.

Badania opisywane w niniejszej rozprawie można postrzegać jako nowatorską próbę zastosowania języków wizualnych w interakcji pomiędzy projektantem a systemem komputerowym do efektywnego pozyskania wiedzy o projekcie oraz wnioskowania o nim za pomocą formuł języka logiki pierwszego rzędu. Żaden z opisanych w literaturze systemów nie podejmuje problemu analizy diagramów tworzonych w fazie wstępnej projektowania architektonicznego [7]. Brakuje również w literaturze systemów realizujących analizę projektów architektonicznych online pozwalającą na natychmiastową identyfikację problemów oraz uwzględnienie wyników walidacji podczas dalszej pracy. Język testów HSSDR jest również pierwszą implementacją otwartego, abstrakcyjnego języka wyższego poziomu o sile wyrazu wystarczającej na formułowanie wymagań dla projektów architektonicznych. W ocenie autora realizuje on postulaty dotyczące powstania takiego języka zawarte w [11].

6.1. Dalsze prace

Dalsze prace nad systemem mogłyby obejmować przede wszystkich rozszerzenie zbioru dostępnych zestawów testów. Nieraz, dopiero podczas zapisu nowych warunków można dostrzec możliwości rozszerzenia systemu o interesujące funkcjonalności. Takim rozszerzeniem mogłoby być np. wzbogacenie systemu o przestrzeń operacyjną i funkcjonalną dla drzwi, opisaną w rozdziale 3.7.3. Otworzyłoby to możliwość sprawdzania wielu interesujących wymagań projektowych.

Obecnie ontologia rozkładów pomieszczeń może być zapisywana jedynie za pomocą języka testów HSSDR w zestawach testów. Tymczasem mogłaby ona być wyrażona explicite i udostępniona do edycji przez inżynierów wiedzy bądź użytkowników systemu, np. za pomocą języka OWL. Można sobie wyobrazić udostępnienie edytora ontologii rodzajów pomieszczeń oraz udostępnienie w edytorze rodzajów pomieszczeń pochodzących z wybranej ontologii. Ontologie zapisane w OWL mogłyby zasilać strukturę relacyjną i być używane podczas wnioskowania. Byłoby to korzystne ze względu na zwiększenie mocy wnioskowania o fakty wynikające z połączenia np. ontologii typów pomieszczeń z regułami w zestawach testów. Takie rozszerzenie poprawiłoby spójność systemu i eliminowało konieczność powielania pewnych definicji w wielu zestawach testów.

Dalsze prace nad systemem powinny przebiegać w dwóch kierunkach. Z jednej strony interesujące jest wspomaganie fazy wstępnej projektowania, nie tylko poprzez wspieranie projektowania za pomocą edytora rozkładów pomieszczeń HSSDR, ale również poprzez kontynuację prac nad integracją z narzędziem SketchUp opisanych w rozdziale 5.2. Coraz większa otwartość tego typu narzędzi stwarza możliwości wymiany danych między różnymi wyspecjalizowanymi aplikacjami.

Z drugiej strony, jak pokazują prace opisane w rozdziale 5.1, mechanizmy walidacji projektów HSSDR mogą wykraczać poza fazę wstępną projektowania i mogą być stosowane również dla finalnych projektów. Jak najbardziej słuszne okazuje się przyjęte w rozdziale 5 podejście opierania się na wymianie danych pomiędzy aplikacjami, w przeciwieństwie do realizacji walidacji projektu w środowisku jakiegoś edytora BIM. Duże możliwości daje operowanie na projektach w formacie IFC, będącym standardem wymiany danych pomiędzy aplikacjami BIM. Natomiast użycie platforma pośredniczących, takich jak BIMserver wspomniany w rozdziale 2.5.2, na pewno ułatwiłoby prace i pozwoliłoby na uniknięcie wstępnej obróbki projektu oraz uniezależnienie się od szczegółów implementacji oraz zmian w nowych wersjach formatu IFC.

Kompleksowe sprawdzanie wymagań projektowych jest dużym wyzwaniem dla branży [11]. W ocenie autora całościowa realizacja sprawdzania reguł bez ingerencji człowieka, prawdopodobnie nigdy nie będzie możliwa, niezależnie od fazy projektu. Zawsze pozostanie fragment przepisów obejmujący nieścisłe, wymagające arbitralnego osądu warunki. Większość reguł jest możliwa do sformalizowania, jest to jednak zależne również od postępów w dziedzinie formalizacji regulacji prawnych oraz powodzenia projektów takich jak Akoma Ntoso czy Legal Knowledge Interchange Format opisanych w rozdziale 2.3.

7. Bibliografia

[1] J. Szuba, Graphs and Graph Transformations in Design, rozprawa doktorska, Polska Akademia Nauk, 2005.

[2] H.-H. Tang i J. S. Gero, „Cognition-based CAAD," *B. de Vries, J. P. v. Leeuwen, H. H. Achten (Eds.),CAADFutures 01, Eindhoven,* pp. 521-531, 2001.

[3] B. Kraft i N. Wilhelms, „Interactive distributed knowledge support for conceptual building design," *Proc. of the 10th Intl. Conf. on Computing in Civil and Building Engineering (ICCCBE-X) Bauhaus-Universität Weimar.,* pp. 1-14, 2004.

[4] B. Kraft i N. M, „Semantic tool support for conceptual design," *Bridges 10.40704 43.,* 2003.

[5] K. M., D. W., B. A. i H. C., „Contemporary Digital Techniques in the Early Stages of Design," *Computer Aided Architectural Design Futures 2005, Springer Netherlands,* pp. 165-174, 2005.

[6] H. Penttilä, „Early architectural design and BIM," *Computer-Aided Architectural Design Futures (CAADFutures), Springer Netherlands,* pp. 291-302, 2007.

[7] B. Kraft i M. Nagl, „Visual knowledge specification for conceptual design: Definition and tool support," *Advanced Engineering Informatics 21.1,* pp. 67-83, 2007.

[8] W. Zeiler, P. Savanovic i E. M. C. J. Quanjel, „Design decision support for the conceptual phase of the design process," w *S. Poggenpohl (Ed.), Proceedings of the International Association of societies of design research: Emerging trends in design research Conference 2007 (IASDR 2007)*, Hong Kong: School of Design, The Hong Kong Polytechnic University, 2007.

[9] M. Yang, „Observations on concept generation and sketching in engineering design," *Research in Engineering Design,* pp. 20:1-11, 2009.

[10] A. Yurchyshyna, C. Faron-Zucker, A. Zarli i N. Le Thanh, „Knowledge capitalisation and organisation for conformance checking model in construction," *International Journal of Knowledge Engineering and Soft Data Paradigms,* 2010.

[11] C. Eastman, J. M. Lee, Y. S. Jeong i J. K. Lee, „Automatic rule-based checking of building designs," *Automation in Construction 18(8),* pp. 1011-1033., 2009.

[12] „Standardy Wykonywania Zawodu i Zakres Usług Architekta," Izba Architektów RP, [Online]. Available: http://www.izbaarchitektow.pl/pliki/standardy_wykonywania_zawodu_i_zakres_uslug_architekta.pdf. [Data uzyskania dostępu: 7 1 2016].

[13] „Prawo budowlane," Marszałek Sejmu Rzeczypospolitej Polskiej, [Online]. Available: http://isap.sejm.gov.pl/DetailsServlet?id=WDU20130001409. [Data uzyskania dostępu: 7 1 2016].

[14] B. Kraft i M. Nagl, „Parameterized specification of conceptual design tools in civil engineering," *Applications of Graph Transformations with Industrial Relevance, Springer Berlin Heidelberg,* pp. 90-105, 2004.

[15] M. Suwa i B. Tversky, „Constructive perception: A meta-cognitive skill for coordinating perception and conception," *Proc. Twenty-fifth Annual Conf. of the Cognitive Science Society, Hillsdale, NJ, Erlbaum,* pp. 1140-1144, 2003.

[16] „CRC Construction Innovation. Adopting BIM for Facilities Management: Solutions for Managing the Sydney Opera House," *Cooperative Research Center for Construction Innovation, Brisbane, Australia.,* 2007.

[17] Autodesk, „Revit Overview," [Online]. Available: http://www.autodesk.com/products/revit-family/overview. [Data uzyskania dostępu: 8 1 2016].

[18] W. Yan, C. Culp i R. Graf, „Integrating BIM and gaming for real-time interactive architectural visualization," *Automation in Construction 20.4,* pp. 446-458, 2011.

[19] S. Zhang, J. Teizer, J. K. Lee, C. M. Eastman i M. Venugopal, „Building information modeling (BIM) and safety: Automatic safety checking of construction models and schedules," *Automation in Construction,* 2013.

[20] Y. H. Lin, Y. S. Liu, G. Gao, X. G. Han, C. Y. Lai i M. Gu, „The IFC-based path planning for 3D indoor spaces," *Advanced Engineering Informatics,* 2013.

[21] „Solibri Model Checker," Solibri, [Online]. Available: http://www.solibri.com/products/solibri-model-checker/. [Data uzyskania dostępu: 14 12 2015].

[22] S. Azhar, M. Hein i B. Sketo., „Building Information Modeling (BIM): Benefits, Risks and Challenges," *McWhorter School of Building Science, Auburn University. Auburn. Alabama,* 2008.

[23] „BIM: polska perspektywa,” Autodesk, 2015. [Online]. Available: http://www.autodesk.pl/campaigns/bim-day-2015-post-event. [Data uzyskania dostępu: 15 1 2016].

[24] „The Business Value of BIM in Europe,” Autodesk, 2010. [Online]. Available: http://images.autodesk.com/adsk/files/business_value_of_bim_in_europe_smr_final.pdf. [Data uzyskania dostępu: 15 1 2016].

[25] „Government Construction Strategy,” Cabinet Office, 2011. [Online]. Available: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/61152/Government-Construction-Strategy_0.pdf. [Data uzyskania dostępu: 15 1 2016].

[26] SketchUp, Google, [Online]. Available: http://www.sketchup.com/. [Data uzyskania dostępu: 08 05 2015].

[27] L. Khemlani, „“Top Criteria for BIM Solutions”, AECbytes, October issue,” [Online]. Available: http://www.aecbytes.com..

[28] A. Yurchyshyna, C. Faron-Zucker, N. Le Thanh, C. Lima i A. Zarli, „Towards an ontology-based approach for conformance checking modeling in construction,” *PROC. OF 24TH W78 CONFERENCE–BRINGING ITC KNOWLEDGE TO WORK (pp. 195-202).,* 2007.

[29] „Akoma Ntoso,” Akoma Ntoso Group, [Online]. Available: http://www.akomantoso.org/. [Data uzyskania dostępu: 6 1 2016].

[30] „OASIS,” [Online]. Available: https://www.oasis-open.org/. [Data uzyskania dostępu: 6 1 2015].

[31] E. K. Normalizacyjny, „CEN MetaLex,” [Online]. Available: http://www.metalex.eu/. [Data uzyskania dostępu: 6 1 2016].

[32] „Legal Knowledge Interchange Format,” Estrella - European project for Standardized Transparent Representations in order to Extend Legal Accessibility, [Online]. Available: http://www.estrellaproject.org/?page_id=5. [Data uzyskania dostępu: 6 1 2016].

[33] „LKIF Core Ontology,” Estrella, [Online]. Available: http://www.estrellaproject.org/?page_id=3. [Data uzyskania dostępu: 6 1 2016].

[34] C. Eastman, P. Teicholz, R. Sacks i K. Liston, BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors,

Wiley, 2008.

[35] R. Coyne, M. Rosenman, A. Radeford, M. Balachandran i J. Gero, Knowledge based design systems, Addison-Wesley Publishing Company, 1990.

[36] M. Chein, M. Croitoru i M. L. Mugnier, „Visual reasoning with graph-based mechanisms: the good the better the best," *The Knowledge Engineering Review,* tom 28(03), pp. 249-271, 2013.

[37] R. Wieleba, „Inżynieria wiedzy w systemach ekspertowych," *Zeszyty Naukowe Rok 5, Zeszyt 5,* pp. 195-216, 2011.

[38] K. Rudnik, rozprawa doktorska p.t. "Koncepcja i implementacja systemu wnioskującego z probabilistyczno-rozmytą bazą wiedzy", Opole: Politechnika Opolska, Wydział Elektrotechniki, Automatyki i Informatyki, Instytut Automatyki i Informatyki, 2011.

[39] J. Tiuryn, J. Tyszkiewicz i P. Urzyczyn, „Logika dla informatyków," [Online]. Available: http://www.mimuw.edu.pl/~urzy/calosc.pdf. [Data uzyskania dostępu: 26 06 2013].

[40] F. Baader (Ed.), The description logic handbook: theory, implementation, and applications., 2003: Cambridge university press.

[41] A. Yurchyshyna, Rozprawa doktorska p. t. Modélisation du contrôle de conformité en construction: une approche ontologique, 2009.

[42] U. Nilsson i J. Małuszyński, Logic, Programming and Prolog, John Wiley & Sons, 1995.

[43] J. F. Sowa, „Semantic Networks," *Encyclopedia of Artificial Intelligence,* tom dostępny pod adresem http://www.jfsowa.com/pubs/semnet.htm, Wiley, 1987.

[44] T. Gruber, „What is an Ontology," [Online]. Available: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html. [Data uzyskania dostępu: 22 10 2015].

[45] J. F. Sowa, „Ontology by John F. Sowa," 29 11 2010. [Online]. Available: http://www.jfsowa.com/ontology/. [Data uzyskania dostępu: 14 3 2015].

[46] F. H. Abanda, J. H. Tah i R. Keivani, „Trends in built environment semantic Web applications: Where are we today?," *Expert Systems with Applications,* 2013.

[47] W. W. W. C. (W3C), „Resource Description Framework (RDF)," [Online]. Available: http://www.w3.org/RDF/. [Data uzyskania dostępu: 22 4 2015].

[48] W. W. W. C. (W3C), „RDF Schema,” 2014. [Online]. Available: https://www.w3.org/TR/rdf-schema/. [Data uzyskania dostępu: 20 4 2015].

[49] W. W. W. C. (W3C), „Web Ontology Language (OWL),” 2012. [Online]. Available: http://www.w3.org/2004/OWL/. [Data uzyskania dostępu: 15 4 2015].

[50] W. W. W. C. (W3C), „SPARQL Query Language,” 2008. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/. [Data uzyskania dostępu: 15 4 2015].

[51] J. Gero i U. Kannengiesser, „The situated function-behaviour-structure framework,” *J.S. Gero (Ed.), Artificial Intelligence in Design'02, Dordrecht,* pp. 89-104, 2002.

[52] A. Schürr i A. J. Winter, „UML Packages for Programmed Graph Rewriting Systems,” *in: Proc. TAGT'98 - Theory and Application of Graph Transformations, LNCS, Berlin: Springer-Verlag,* pp. 396-409, 2000.

[53] B. Böhlen, D. Jäger, A. Schleicher i B. Westfechtel, „UPGRADE: A Framework for Building Graph-Based Interactive Tools,” *LNCS, A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, Eds. vol. 2505. Barcelona, Spain. Springer ,* p. 270–285, 2002.

[54] B. Kraft i M. Nagl, „Semantic tool support for conceptual design,” *Bridges, 10(40704), 43,* 2003.

[55] U. Flemming i J. Snyder, „Building and Databases: the SEED Experience,” *Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen. Weimar ,* 1997.

[56] U. Flemming, „SEED-Layout Tutorial, School of Architecture and Institute for Complex Engineered Systems Carnegie Mellon University Pittsburgh, PA 15213,” 21 1 1999. [Online]. Available: https://www.andrew.cmu.edu/user/ujf/download_files/SL_tutorial.pdf. [Data uzyskania dostępu: 30 6 2015].

[57] O. Akin, Z. Aygen, M. Cumming, M. Donia, R. Sen i Y. Zhang, „Computational Specification of Building Requirements in theEarly Stages of Design,” *Fourth Design and Decision Support Systems in Architecture and Urban Planning Maastricht,* 1998.

[58] M. Eisfeld i R. Scherer, „Assisting conceptual design of building structures by an interactive description logic based planner,” *Advanced Engineering Informatics ,* tom 17(1), 2003.

[59] „ Eisfeld Ingenieure AG,” [Online]. Available: http://www.e3p.de/de/home/. [Data uzyskania dostępu: 15 12 2015].

[60] „ConED by Michael Eisfeld @ ECLM 2013,” [Online]. Available: https://www.youtube.com/watch?v=rEm3LrpvhS8. [Data uzyskania dostępu: 5 1 2016].

[61] „BIMserver.org,” BIMserver.org, [Online]. Available: http://bimserver.org/. [Data uzyskania dostępu: 1 1 2016].

[62] „XML Soap,” World Wide Web Consortium, [Online]. Available: http://www.w3schools.com/xml/xml_soap.asp. [Data uzyskania dostępu: 26 1 2016].

[63] „JSON: JavaScript Object Notation,” World Wide Web Consortium, [Online]. Available: http://www.w3schools.com/json/default.asp. [Data uzyskania dostępu: 26 1 2016].

[64] „CORENET e-Information System,” Singapore Governement, [Online]. Available: http://www.corenet.gov.sg. [Data uzyskania dostępu: 30 11 2015].

[65] K. Lindberg, „MATL, Development of a new ICT-system for registration and assessment of accessibility to public buildings,” *Property Management, Statsbygg, BAS Conference,* 2006.

[66] L. Ding, R. Drogemuller, M. Rosenman, D. Marchant i J. Gero, „Automating code checking for building designs-DesignCheck,” 2006.

[67] „EDM,” Jotne IT, [Online]. Available: http://www.epmtech.jotne.com/solutions/bim. [Data uzyskania dostępu: 30 11 2015].

[68] „International Code Council,” International Code Council, [Online]. Available: http://www.iccsafe.org/. [Data uzyskania dostępu: 2015 12 10].

[69] J. Dimyadi i R. Amor, „Automated Building Code Compliance Checking–Where is it at?,” *Proceedings of the 19th International CIB World Building Congress. Brisbane, Australia,* 2013.

[70] „3D-4D Building Information Modeling,” GSA, [Online]. Available: http://www.gsa.gov/portal/content/105075. [Data uzyskania dostępu: 9 12 2015].

[71] G. S. Administration, „U.S. Courts Design Guide,” [Online]. Available: http://www.gsa.gov/graphics/pbs/Courts_Design_Guide_07.pdf. [Data uzyskania dostępu: 11 12 2015].

[72] „AutoCodes Project Team Phase II Report,” [Online]. Available: http://fiatech.org/images/stories/projects/FiatechAutoCodesPh2-Report-Sept2015.pdf. [Data uzyskania dostępu: 15 12 2015].

[73] A. Yurchyshyna, C. Faron-Zucker, N. Le Thanh i A. Zarli, „Ontological approach for the conformity-checking modelling in construction," *Proceedings of the 10th International Conference on Enterprise Formation Systems (ICEIS2008), Barcelona, Spain,* tom Vol. 1216, 2008.

[74] L. Belouaer, M. Bouzid i A. I. Mouaddib, „Spatial knowledge in planning language," *International Conference on Knowledge Engineering and Ontology Development,* 2011.

[75] M. Bhatt, S. Gajek, E. Grabska i W. Palacz, „Artefactual reasoning in a hypergraph-based CAD system," *Computer Recognition Systems, Springer Berlin Heidelberg,* pp. 471-478, 2011.

[76] E. Grabska, G. Ślusarczyk i S. Gajek, „Knowledge Representation for Human-Computer Interaction in a System Supporting Conceptual Design," *Fundamenta Informaticae,* 2013.

[77] E. Grabska, A. Borkowski, W. Palacz i S. Gajek, „Hypergraph system supporting design and reasoning," *Computing in Engineering EG-ICE Conference (pp. 134-141),* 2009.

[78] W. Palacz i E. S. Grabska, „Conceptual designing supported by automated checking of design requirements and constraints," *Improving Complex Systems Today, Springer London,* pp. 257-265, 2011.

[79] E. Grabska, G. Slusarczyk i S. Gajek, „Knowledge representation in visual design," *Proc. of the 3rd International Conference on Advanced Cognitive Technologies and Applications,* pp. 41-46, 2011.

[80] „Sprawozdanie z grantu nr 5T07F00225, Transformacje grafowe w konstrukcji systemów wnioskowania diagramowego," Kraków, 2006.

[81] E. Grabska, G. Ślusarczyk i M. Glogaza, „Design description hypergraph language," *Computer Recognition Systems 2 , Springer Berlin Heidelberg,* pp. 763-770, 2007.

[82] E. Grabska, G. Ślusarczyk i T. L. Le, „Visual design and reasoning with the use of hypergraph transformations," *Electronic Communications of the EASST, 10,* 2008.

[83] A. T. Parr, „ANTLR (ANother Tool for Language Recognition)," 2014. [Online]. Available: http://www.antlr.org/. [Data uzyskania dostępu: 20 1 2014].

[84] G. Booch, J. Rumbaugh i I. Jacobson, The Unified Modeling Language User Guide, Addison Wesley Longman, 1999.

[85] M. Minas, „Concepts and realization of a diagram editor generator based on hypergraph transformation,” *Science of Computer Programming,* nr 44, pp. 157-180, 2002.

[86] E. Grabska, Projektowanie wizualne wspomagane komputerem, Warszawa: Akademicka Oficyna Wydawnicza EXIT, 2007.

[87] E. Neufert, Podręcznik projektowania architektoniczno-budowlanego, Warszawa: Arkady, 2000.

[88] Department of Computer Science 3, RWTH Aachen University, „PROGRES { PROgrammed Graph Rewriting Systems,” [Online]. Available: http://www.se-rwth.de/research/. [Data uzyskania dostępu: 2009].

[89] W. Palacz, rozprawa doktorska p.t. Hierarchical Graphs in Computer-Aided Design Systems, Wydział Matematyki i Informatyki Uniwersytetu Jagillońskiego, 2006.

[90] G. Ślusarczyk, „Hierarchical Hypergraph Transformations in Engineering Design,” *Journal of Applied Computer Science,* nr 11(2), pp. 67-82, 2003.

[91] R. Bardohl, „GenGEd: A generic graphical editor for visual languages based on algebraic graph grammars,” *VL'98, Proc. 1998 IEEE Symp. on Visual Languages, IEEE Computer Society,* p. 48–55, 1998.

[92] J. Rekers i A. Schürr, „A graph based framework for the implementation of visual environments,” *in: Proc.1996 IEEE Symp. on Visual Languages, Boulder, Colorado, IEEE Computer Society Press, Silverspring,* p. 148–155, 1996.

[93] M. Nagl, A. Schürr i M. Münch, Applications of Graph Transformations with Industrial Relevance, Springer-Verlag, 1999.

[94] A. Habel, Hyperedge Replacement: Grammars and Languages, rozprawa doktorska, Universität Bremen, 1980.

[95] G. Rozenberg, Handbook of graph grammars and computing by graph transformation, vol 1: Foundations, World Scientific, 1997.

[96] E. Grabska, K. Grzesiak-Kopeć, J. Lembas, A. Łachwa i G. Ślusarczyk, „Hypergraphs in Diagrammatic Design,” *Computer Vision and Graphics, Springer Netherlands,* pp. 111-117, 2006.

[97] E. Grabska, A. Łachwa, G. Ślusarczyk, K. Grzesiak-Kopec i J. Lembas, „Hierarchical Layout Hypergraph Operations and Diagrammatic Reasoning,” *Machine Graphics &*

Vision, tom 1, nr 16, pp. 23-38, 2007.

[98] R. Fagin, Y. Moses, J. Y. Halpern i M. Y. Vardi, Reasoning About Knowledge, MIT Press, 2003.

[99] „Strona Projektu ANTLR," [Online]. Available: http://www.antlr.org. [Data uzyskania dostępu: 14 02 2013].

[100] B. N. Grosof, I. Horrocks, R. Volz i S. Decker, „Description logic programs: Combining logic programs with description logic," *Proceedings of the 12th international conference on World Wide Web,* nr (pp. 48-57). ACM, 2003.

[101] „Specification of the Legal Knowledge Interchange," ESTRELLA, European project for Standardized Transparent Representations in order to Extend Legal Accessibility, 2007. [Online]. Available: http://www.estrellaproject.org/doc/D1.1-LKIF-Specification.pdf. [Data uzyskania dostępu: 28 1 2016].

[102] Ministerstwo Infrastruktury, „Rozporządzenie Ministra Infrastruktur w sprawie warunków technicznych, jakim powinny odpowiadać budynki i ich usytuowanie(teskt jednolity)," Sejm RP, 2015. [Online]. Available: http://isip.sejm.gov.pl/DetailsServlet?id=WDU20020750690. [Data uzyskania dostępu: 7 12 2012].

[103] M. Bhatt i C. Freksa, „Spatial Computing for Design—an Artificial Intelligence Perspective," *Studying Visual and Spatial Reasoning for Design Creativity ,* 2015.

[104] M. Bhatt, F. Dylla i J. Hois, „Spatio-terminological inference for the design of ambient environments," *Spatial Information Theory ,* 2009.

[105] K. Kowalski, „Projektowanie bez barier - wytyczne," 2014. [Online]. Available: http://www.integracja.org/wp-content/uploads/2014/05/projektowanieBB21.pdf. [Data uzyskania dostępu: 15 1 2015].

[106] „ARCHICAD," GRAPHISOFT, 2016. [Online]. Available: http://www.graphisoft.com/archicad/. [Data uzyskania dostępu: 20 1 2016].

[107] „Visual C#," Microsoft, 2016. [Online]. Available: https://msdn.microsoft.com/pl-pl/library/kx37x362.aspx. [Data uzyskania dostępu: 20 1 2016].

[108] „Platforma .NET Framework," Microsoft, 2016. [Online]. Available: https://msdn.microsoft.com/pl-pl/vstudio/aa496123.aspx. [Data uzyskania dostępu: 20 1 2016].

[109] „Język Ruby,” Ruby Central, 2016. [Online]. Available: https://www.ruby-lang.org/pl/. [Data uzyskania dostępu: 20 1 2016].

[110] M. w. Javy, „Pobieranie oprogramowania Java,” Oracle, 2016. [Online]. Available: https://www.java.com/pl/download/. [Data uzyskania dostępu: 29 2 2016].

Dodatek: HSSDR – Opis działania programu

Program komputerowy HSSDR jest aplikacją desktopową wyposażoną w graficzny interfejs użytkownika. Został on zaimplementowany w języku Java. W celu uruchomienia programu wymagane jest posiadanie na danym komputerze zainstalowanej maszyny wirtualnej Javy w wersji 6 lub nowszej. Maszynę wirtualną można pobrać z następującej strony: [110].

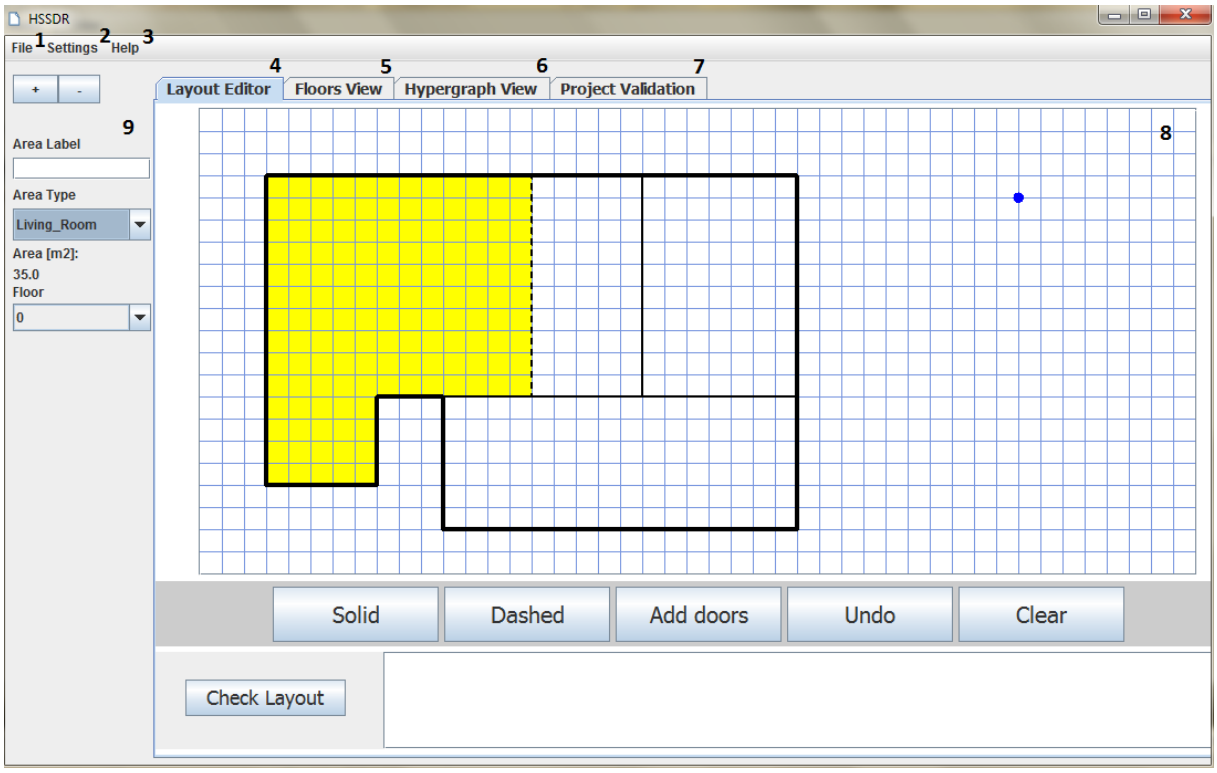
W katalogu programu znajdują się:

- plik wykonywalny *HSSDR.jar*, służący do uruchomienia programu,
- katalog *test_files*, zawierający zestawy testów HSSDR.

Elementy głównego okna programu

Po uruchomieniu programu widoczne jest główne okno z wybraną zakładką edycji rozkładów pomieszczeń (Rysunek A). Jego podstawowe elementy to:

1. Menu głównych opcji,
2. Menu ustawień,
3. Menu pomocy,
4. Zakładka edycji rozkładów pomieszczeń,
5. Zakładka widoku pięter,
6. Zakładka podglądu hipergrafów,
7. Zakładka walidacji projektu,
8. Panel główny,
9. Lewy panel.



Rysunek A Główne okno programu.

Za pomocą zakładek (4,5,6,7) możliwa jest zmiana kontekstu pracy pomiędzy edycją rozkładu pomieszczeń pojedynczego piętra, edycją relacji pomiędzy piętrami, a edycją lub wyborem zestawów testów zgodności które są wykonywane dla projektu. Podczas zmiany zakładek zmienia się też zawartość panelu głównego (8) oraz akcje dostępne w lewym panelu (9).

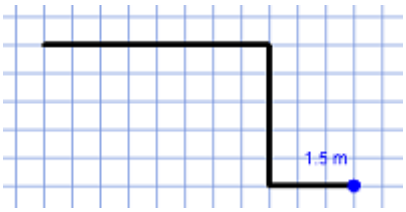
Tworzenie rozkładu pomieszczeń

Tworzenie rozkładu pomieszczeń rozpoczyna się od narysowania obrysu. Następnie może on być dzielony na mniejsze pomieszczenia. W dalszej kolejności można nanosić dodatkowe elementy oraz cofać swoje akcje. Aby określić rozmiaru dostępnego arkusza, gęstości siatki i inne parametry należy przed rozpoczęciem projektowania skorzystać z ustawień systemu. Możliwa jest zmiana wielkości widoku za pomocą przycisków ("+", "-") w górnej części lewego panelu.

Tworzenie obrysu

Aby rozpocząć rysowanie obrysu należy nacisnąć lewy przycisk myszy na kratkowanym obszarze. Następnie, w miarę przesuwania kursora po siatce, do obrysu dodawane są kolejne odcinki (Rysunek B). Aby wycofać się z aktualnie rysowanej linii należy

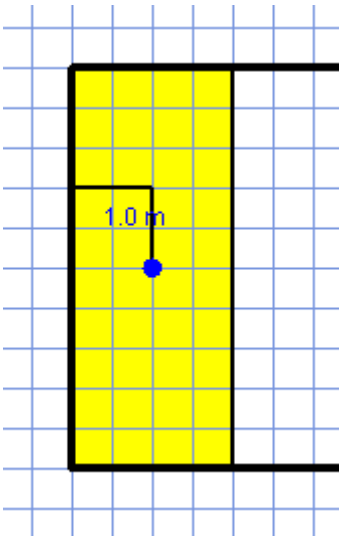
nacisnąć prawy przycisk myszy. Rysowanie obrysu kończy się w momencie zapętlenia łamanej.



Rysunek B Rysowanie obrysu

Dzielenie obszarów

Po stworzeniu obrysu można dowolnie dzielić obszary na mniejsze. Aby dokonać podziału należy zaznaczyć wybrany obszar klikając na niego lewym przyciskiem myszy. Następnie, po kliknięciu na krawędzi zaznaczonego obszaru, rozpoczyna się rysowanie linii podziału (Rysunek C). Aby wycofać się z aktualnie rysowanej linii, należy nacisnąć prawy przycisk myszy. Rysowanie linii podziału kończy się w momencie osiągniecia krawędzi obszaru, w tym samym momencie obszar jest dzielony.

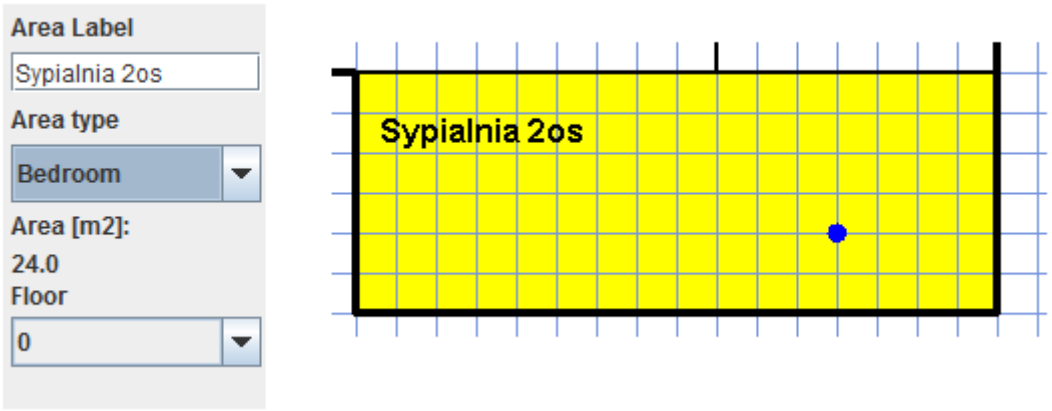


Rysunek C Rysowanie linii podziału

Właściwości obszarów

W celu wprowadzenia właściwości obszaru należy go najpierw zaznaczyć, klikając lewym przyciskiem myszy. Zaznaczone obszary kolorowane są na żółto, natomiast obszary znajdujące się pod kursorem na niebiesko. W celu odznaczenia, należy ponownie kliknąć na zaznaczony obszar lub zaznaczyć inny. Dla zaznaczonego obszaru w lewym panelu można wprowadzić jego nazwę (pole „Area Label”) oraz wybrać rodzaj pomieszczenia (list rozwijana

„Area Type”). W kolejnym polu wyświetlana jest powierzchnia obszaru w metrach kwadratowych.

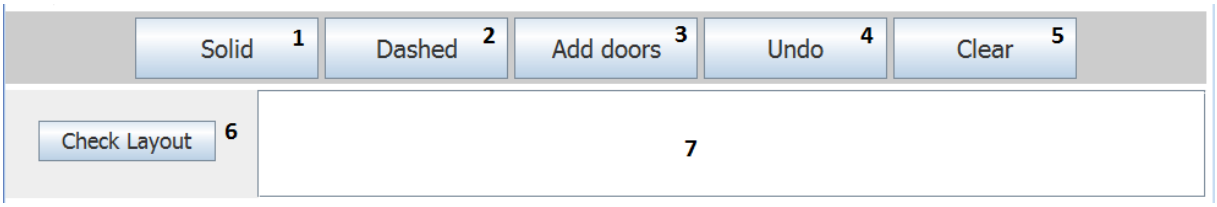


Rysunek D Zaznaczony obszar oraz jego właściwości

Drzwi, linie przerywane, cofanie podziału, czujniki bezpieczeństwa, testy zgodności

Dolne menu składa się z następujących elementów (Rysunek E):

1. Przycisk zmiany na ciągłą linie podziału.
2. Przycisk zmiany na przerywaną linie podziału.
3. Przycisk dodawania drzwi.
4. Przycisk cofania podziału.
5. Przycisk kasowania projektu.
6. Przycisk wywoływania testów zgodności.
7. Konsola walidacji projektu.

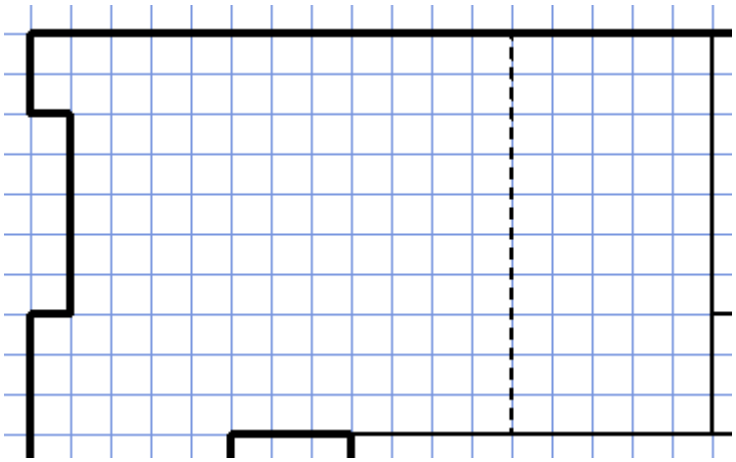


Rysunek E Elementy dolnego menu

Przyciski 1 i 2 pozwalają na zmianę linii podziału pomiędzy linią ciągłą a przerywaną. Linia przerywana (Rysunek F) może oznaczać, w zależności od wybranej konfiguracji:

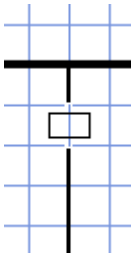
- oddzielenie od siebie pomieszczeń w sensie logicznym, bez oddzielenia ich fizycznie, np. kuchnia połączona z salonem, lub

- relacje widoczności pomiędzy pomieszczeniami, np. biuro ze szklaną ścianą. W przypadku takiej krawędzi może ona zawierać również symbol drzwi, żeby oznaczyć dostępność.



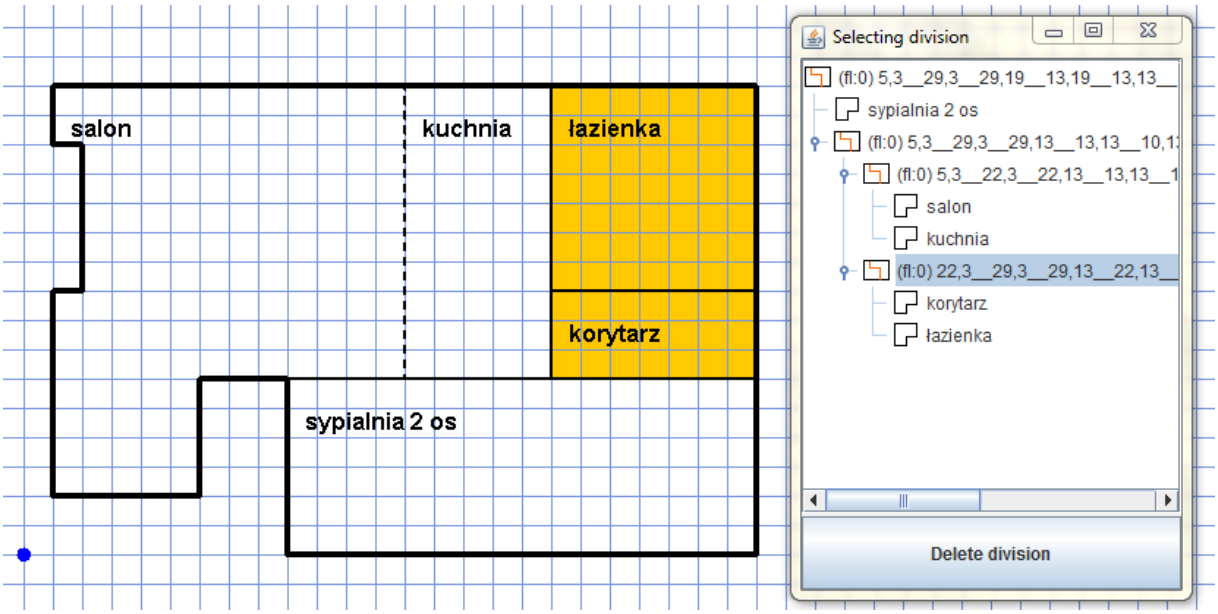
Rysunek F Przerywana linia podziału.

Naciśnie przycisku dodawania drzwi (3) powoduje przejście w tryb dodawania drzwi. Mogą one być umieszczane na ścianach, za pomocą lewego przycisku myszy. Aby wyjść z trybu dodawania drzwi, należy ponownie nacisnąć przycisk dodawania drzwi.



Rysunek G Dodawanie drzwi

Za pomocą przycisku cofania podziału (4), aktywuje się nowe okno pokazujące historię podziałów dokonanych w projekcie (Rysunek H). Po kliknięciu na węzeł w drzewie podziałów, pomieszczenia powstałe w jego wyniku zostają podświetlone. Następnie, klikając przycisk poniżej, można usunąć zaznaczony podział. Tylko podziały na dole hierarchii mogą być usuwane. Aby usunąć podział znajdujący się wyżej w hierarchii, należy najpierw usunąć późniejsze podziały. Aby zamknąć okno historii podziałów, należy ponownie nacisnąć przycisku cofania podziału (4).



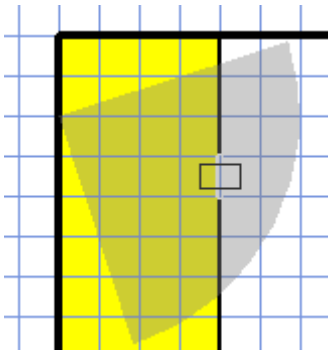
Rysunek H Historia podziałów

Przycisk kasowania (5) usuwa cały projekt.

Przycisk wywoływania testów zgodności (6) powoduje wykonanie aktualnie wybranych testów. Ich wyniki prezentowane są w konsoli walidacji projektu (7).

Na rozkład pomieszczeń można nanosić czujniki bezpieczeństwa (Rysunek I). W tym celu należy:

1. Zaznaczyć pomieszczenie w którym ma być dodany czujnik.
2. Kliknąć prawym przyciskiem myszy, aby wybrać lokalizacje czujnika.
3. Kliknąć drugi raz prawym przyciskiem myszy, aby wybrać kąt skierowania czujnika.

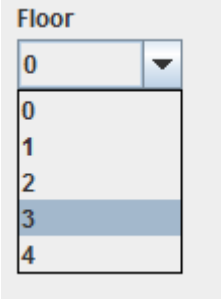


Rysunek I Czujnik bezpieczeństwa

Widok pięter

Ustalanie liczby pięter

W przypadku projektowania budynków wielopiętrowych należy rozpocząć od wybrania liczby pięter w ustawieniach. Aby wybrać piętro do edycji należy skorzystać z listy rozwijanej w lewym panelu.



Rysunek J Wybór liczby pięter

W celu ułatwienia projektowania wyższych pięter, obrys parteru jest zaznaczany jasnoszarą linią również na wyższych piętrach.

Rysunek K Linia obrysu parteru widoczna na pozostałych piętrach

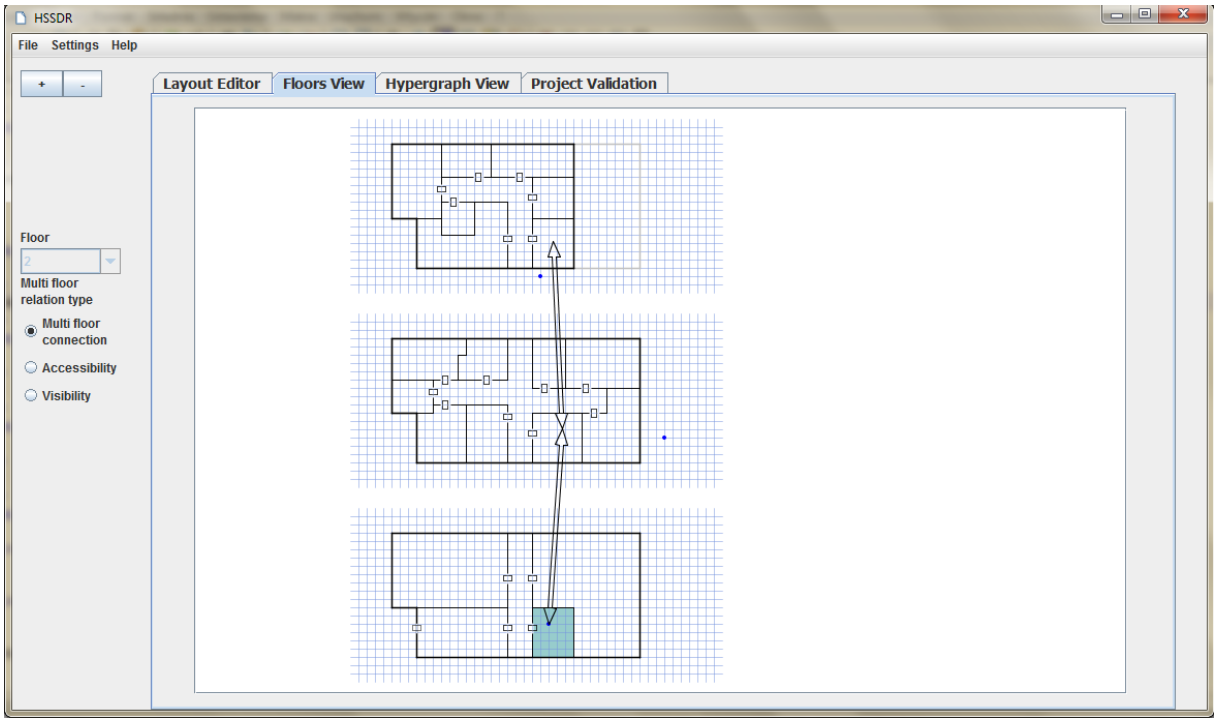
Widok wielu pięter i relacje między piętrami

Po wybraniu zakładki widoku pięter (Rysunek L) prezentowane są wszystkie piętra, z parterem najniżej i ostatnim piętrem najwyżej. W tym widoku można zobaczyć w całości projektowany budynek oraz wprowadzić relacje międzypiętrowe.

Dostępne są następujące typy relacji międzypiętrowych:

- Połączenie wielopiętrowe („Multi Floor Connection”), oznaczające klatki schodowe czy windy,
- Dostępność („Accessibility”), oznaczająca połączenie między dwoma piętrami,
- Widoczność („Visibility”), stosowana w celu zaznaczenia widoczności między poziomami, np. scena widoczna z trybun na wyższych piętrach. Widoczność reprezentowana jest przez grubsze strzałki.

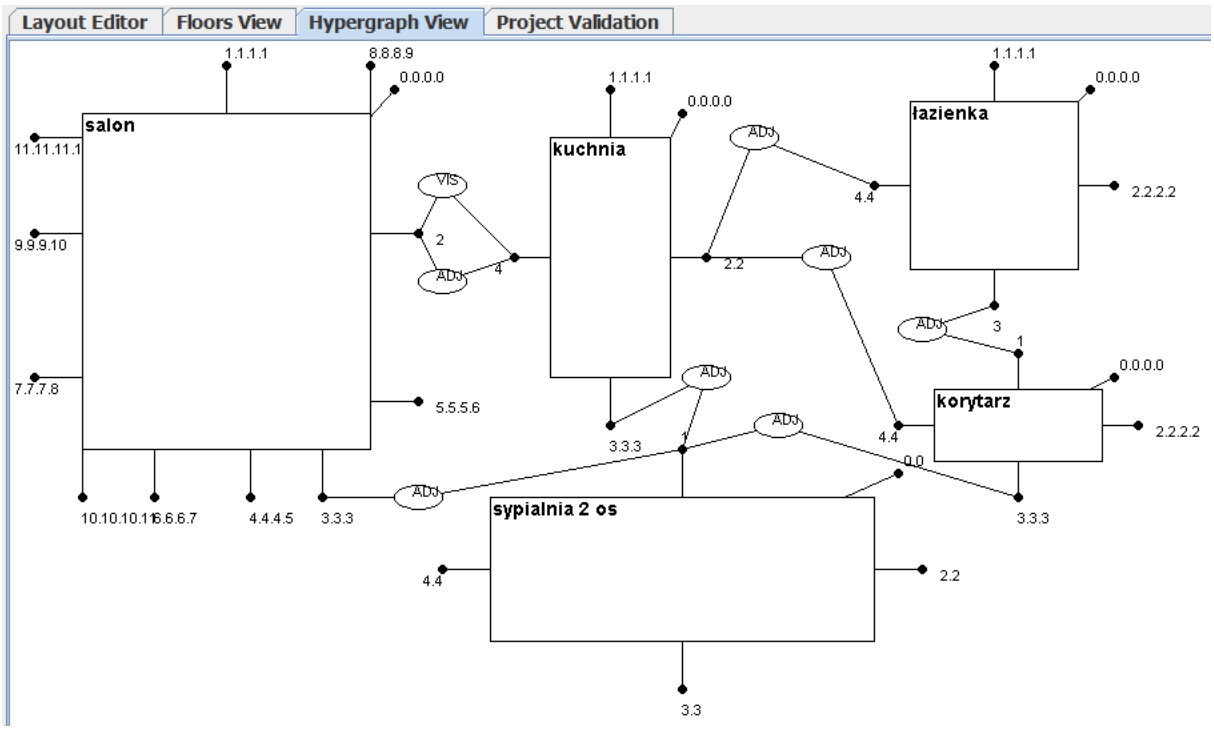
Aby dodać relację między piętrami, należy po wybraniu typu relacji, kliknąć lewym przyciskiem myszy na pomieszczenie na jednym z pięter. Pomieszczenie znajdujące się pod kursorem zostanie podświetlone i będzie ono wchodziło w skład relacji. Po wybraniu pierwszego pomieszczenia pojawi się strzałka symbolizująca relację. Następnie należy kliknąć na kolejne pomieszczenie, aby dodać je do relacji. Użycie prawego przycisku myszy kończy wprowadzanie relacji.



Rysunek L Zakładka widoku pięter z widoczną relacją dostępności między piętrami

Podgląd hipergrafów

Podgląd hipergrafów nie jest niezbędny do używania systemu przez projektanta, zakładka ta służy na potrzeby analizy działania programu. Na liście rozwijanej w lewym panelu możliwy jest wybór trybu wyświetlania hipergrafu: dla poszczególnych pięter lub dla całego budynku. Możliwe jest przesuwanie elementów na planszy. W celu powiększenia/pomniejszania należy skorzystać z przycisków ("+", "-") w górnej części lewego panelu.

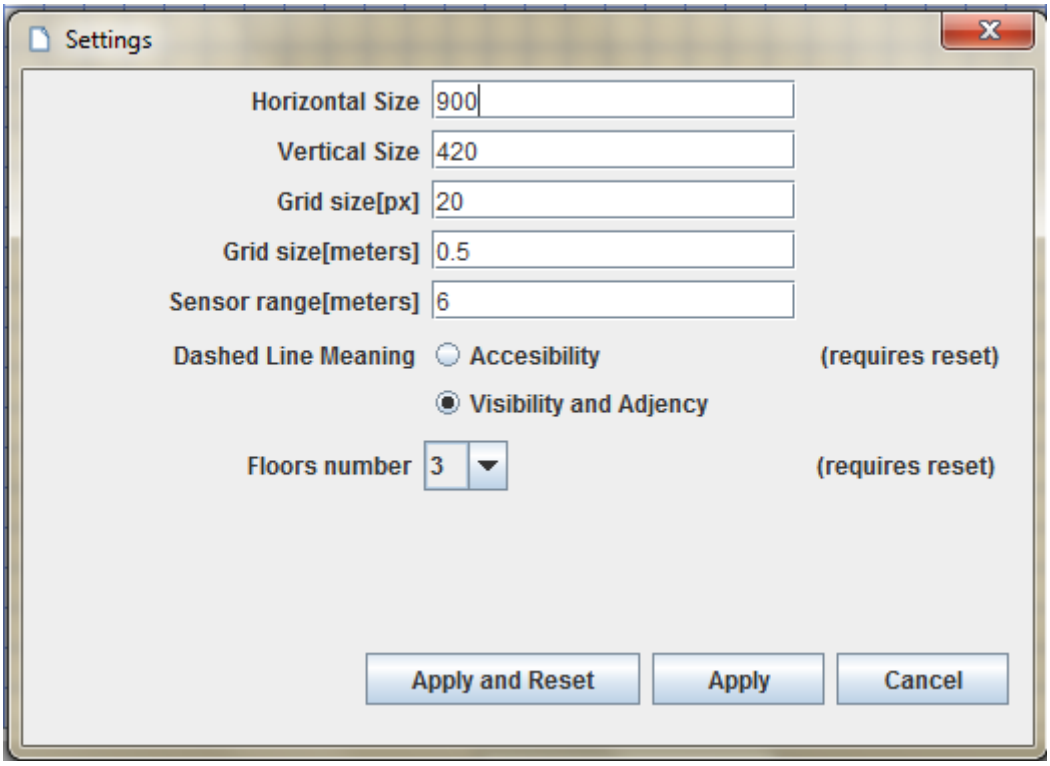


Ustawienia sytemu

Po wybraniu opcji ustawień z menu „Settings” możliwa jest zmiana następujących ustawień systemu:

1. Wielkość arkusza używanego do projektowania („Horizontal”, „Vertical Size”). Wielkość wyrażona w pikselach.
2. Gęstość siatki na jaką podzielony jest arkusz („Grid Size (px)”). Wielkość wyrażona w pikselach.
3. Długość w metrach reprezentowaną przez jedną kratkę arkusza („Grid size[meters]”).
4. Zasięg czujników bezpieczeństwa („Sensor range[meters]”). Wielkość wyrażona w metrach.
5. Znaczenie przerywanej linii podziału pomieszczenia („Dashed Line Meaning”).
6. Liczba pięter („Floors numer”).

Zmiana dwóch ostatnich parametrów wymaga usunięcia bieżącego projektu (przycisk „Apply and Reset”). Pozostałe parametry mogą być zmienione w dowolnym momencie projektowania (przycisk „Apply”).

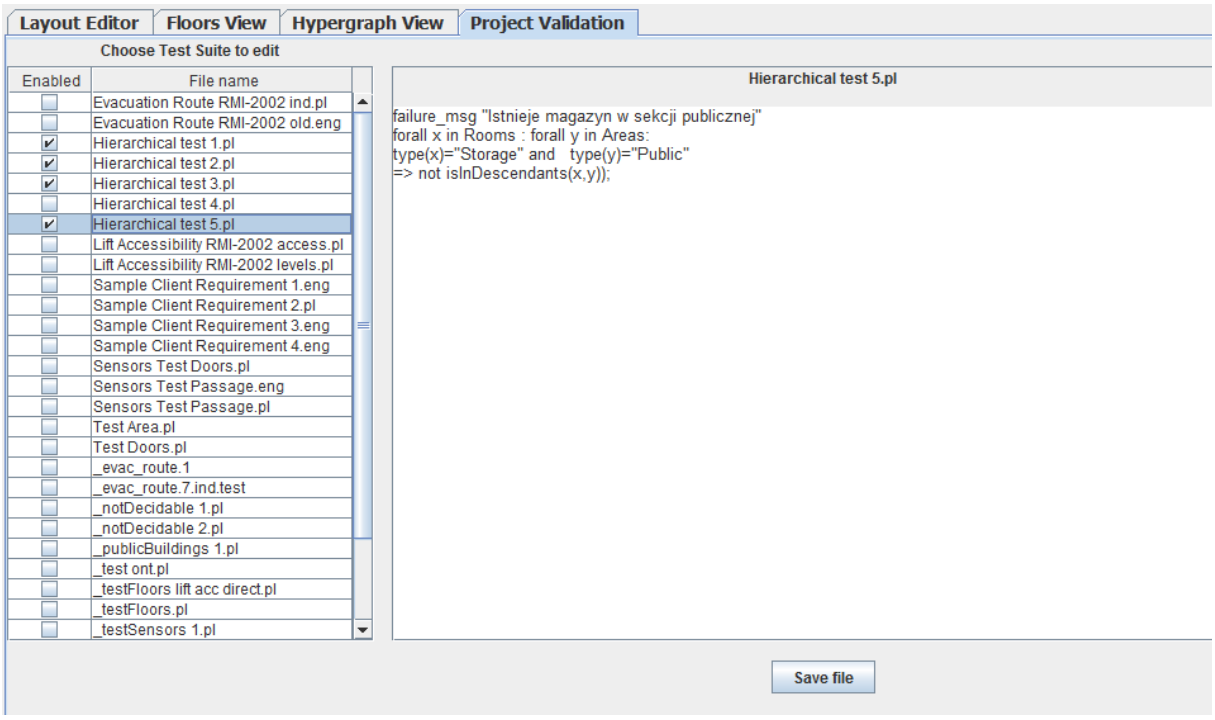


Rysunek N Ustawienia systemu

Wybór zestawów testów

Na zakładce walidacji projektu (Rysunek O) można włączać/wyłączać zestawy testów, a także modyfikować ich zawartość.

Każdy zestaw testów to plik tekstowy, identyfikowany przez nazwę pliku. W lewej części ekranu znajduje się lista zestawów testów. W celu stworzenia nowego zestawu testów należy dodać nowy plik tekstowy do folderu *test_files*. Aby zestaw był brany pod uwagę podczas pracy nad projektem, należy zaznaczyć odpowiednie pole w kolumnie „Enabled”.
Po wybraniu pliku na liście, jego zawartość jest wyświetlana w prawej części ekranu. Zawartość zestawu testów można modyfikować w edytorze znajdującym się w prawej części ekranu. Aby zmiany zostały zapisane należy skorzystać z przycisku zapisu („Save File”).



Rysunek O Zakładka walidacji projektu