SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Alen Zubić

Zagreb, 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Nenad Bojčetić, dipl. ing.

Student:

Alen Zubić

Zagreb, 2015.

Izjavljujem da sam ovaj diplomski rad izradio samostalno koristeći se navedenom literaturom i stečenim znanjem tijekom studija.

Zahvaljujem se svom mentoru prof. dr. sc. Nenadu Bojčetiću na ukazanom povjerenju, savjetima i podršci prilikom pisanja ovoga rada. Zahvaljujem se prof. dr. sc. Mariu Štorgi na ukazanoj prilici da temu obrađujem pomoću resursa Daimler AG razvojnog centra u Ulmu. Profesorima dr. sc. Nevenu Pavkoviću i dr. sc. Draganu Žeželju zahvaljujem se na stručnim savjetima i literaturi. Dipl.-Wirt.-Ing. Nico Koehler, Dr.-Ing. Thomas Naumann te ostali kolege i profesori u Ulmu – hvala na pruženoj potpori i što ste me srdačno prihvatili.

Posebno se zahvaljujem svojoj obitelji koja mi je pružala potporu cijelo vrijeme tijekom školovanja.

Zagreb, svibanj 2015. Alen Zubić

_____

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
procesno-energetski, konstrukcijski, brodostrojarski i inženjersko modeliranje i računalne simulacije

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **ALEN ZUBIĆ** Mat. br.: 0035172146

Naslov rada na hrvatskom jeziku: **RAZVOJ METODA ZA PODRŠKU SLJEDIVOSTI AUTOMATSKIM KREIRANJEM VEZA IZMEĐU KOMPONENTI SKLOPA**

Naslov rada na engleskom jeziku: **CONCEPTUAL DEVELOPMENT OF TRACEABILITY METHODS FOR THE AUTOMATED CREATION OF COMPONENT'S RELATIONS**

Opis zadatka:

With today's increased level of product complexity during the product development, traceability is an approach which naturally fits into an environment where future decisions need to be made quickly and accurately. The existing tools and methods for product development are not able to deal with today's complex dependencies in vehicle development. Therefore the effort to manually model engineering object relations is the main obstacle for using traceability tools in practice. The objective of this thesis is to make a conceptual contribution enhancing traceability in technical systems by developing the methods for automatic relations extraction from CAD product representation and proposing usage of those methods in existing traceability environment at Daimler.

The thesis should contain:

- Analysis of existing methods for traceability at Daimler.
- Analysis of existing design methodologies and features for creating relations between components in CAD applications.
- Development of methods based on CAD relations analysis.
- Creation of the software application for usage of developed methods in CAD application.
- Developed software application validation using prototype CAD models.

The thesis is done in cooperation with Daimler using their resources and support. CAD application used in realization of this thesis should be Siemens NX.

In the thesis all the used references need to be cited as well as the obtained help.

Zadatak zadan:	Rok predaje rada:	Predviđeni datumi obrane:
12. ožujka 2015.	14. svibnja 2015.	20., 21. i 22. svibnja 2015.

Zadatak zadao: Predsjednica Povjerenstva:

Izv. prof. dr. sc. Nenad Bojčetić

Prof. dr. sc. Tanja Jurčević Lulić

TABLE OF CONTENTS

LIST OF FIGURES II
LIST OF TABLES IV
SAŽETAK V
SUMMARY VI
PRODUŽENI SAŽETAK VII
1. Introduction 1
2. Understanding system complexity and industry practices 4
3. System traceability 9
4. DSM (Data Structure Matrix) 11
4.1. System complexity 13
4.2. Visualization 16
4.3. DSM post-processing methods 19
5. Conceptual development of methods for extracting product assembly relations 22
5.1. Constraints 23
5.2. Proximity 28
5.3. Permanent joints 32
5.4. Non-permanent joints 33
5.5. Validation model for developed methods 34
6. PoC (Proof of Concept) 36
6.1. Eclipse IDE 36
6.1.1. NX Open Java API 38
6.1.2. NX Open documentation 38
6.1.3. Debugging and journals 39
6.2. MVC (Model-View-Controller) 40
6.3. Assembly-to-DSM PoC 41
6.3.1. Constraints 43
6.3.2. Proximity 43
6.3.3. Permanent joints 45
6.3.4. Output format 45
6.4. Result validation and visualization 46
7. Discussion 50
8. Conclusion 52
Bibliography 53
Additional documentation 56

LIST OF FIGURES

Figure 1: Traceability matrix - grey area represents the area of interest 1
Figure 2: Master thesis goal – developing methods for finding assembly component relations 2
Figure 3: How are the Light and Front Bumper related? 3
Figure 4: High level meta-model structure of sociotechnical system 5
Figure 5: Projects within matrix-organization [1] 5
Figure 6: Sample DSM 11
Figure 7: Basic DSM classification [16] [4] 11
Figure 8: a) Binary DSM, b) Weighted DSM 12
Figure 9: DSM created from process diagram 12
Figure 10: Diagonal cell logic 13
Figure 11: DSM dependencies 13
Figure 12: Aspects of complexity in product design [4] 15
Figure 13: Grayscale color mapping 16
Figure 14: Network graph and DSM created from the same data sample [27] 16
Figure 15: Broken cluster in DSM and the same representation in network graph [29] 17
Figure 16: The structure of a ballpoint pen represented in DSM matrix and non-directional force-directed graph [4] 18
Figure 17: DSM clustering – three overlapping clusters [31] 19
Figure 18: A simple bus and two modules [31] 20
Figure 19: DSM aggregation 21
Figure 20: Assembly-to-DSM 23
Figure 21: Siemens NX Assembly Constraints menu 24
Figure 22: Touch constraint examples 25
Figure 23: Concentric constraint example 25
Figure 24: Distance constraint examples 26
Figure 25: Fix constraint example 26
Figure 26: Perpendicular constraint example 27
Figure 27: Align/Lock constraint example 27
Figure 28: Fit constraint example 27
Figure 29: Center (1 to 2) constraint example 28
Figure 30: Angle constraint example 28
Figure 31: Non-constrained minimum distance 29
Figure 32: Comparator 29
Figure 33: Virtual box definition 29
Figure 34: Box method 30
Figure 35: Box method problem 30
Figure 36: Low body-box ratio – all relations to selected component from other components within the box should be excluded 31
Figure 37: Box method together with distance method 31
Figure 38: Validation model 34
Figure 39: Manually created aggregated DSM based on model from Figure 38 35
Figure 40: Luna release Eclipse IDE for Java EE Developers 37
Figure 41: Abstract of NX Open Java API role 38
Figure 42: NX Open Java API documentation 38
Figure 43: Siemens NX Java parameter to enable debugging 40
Figure 44: Additional Siemens NX Java parameter 40
Figure 45: MVC concept 41
Figure 46: Assembly-to-DSM project structure 42

Figure 47: Slice of result data - JSON format 47
Figure 48: Visualized results - separated based on the extraction method 48
Figure 49: Aggregated DSMs 48

LIST OF TABLES

Table 1: Relevant functionality in engineering tools to enable traceability [7] 8
Table 2: Daimler's suggested weld types 32
Table 3: Intro algorithm 42
Table 4: Constraints algorithm 43
Table 5: Proximity algorithm 44
Table 6: Permanent joints algorithm 45

SAŽETAK

Cilj ogova rada je identificirati metode koje pružaju mogućnost automatskog prepoznavanja postojećih veza između komponenata sklopa iz CAD modela. CAD alat korišten tijekom ovoga rada je Siemens NX. Neke od metoda su direktno bazirane na geometrijskim relacijama kreiranim prilikom konstruiranja, a ostale se koriste drugim dostupnim CAD podacima kojima se pristupa pomoću API-a. Sve identificirane metode su detaljno objašnjene kroz rad, a prepoznavanje veza je bitno za podršku sljedivosti.

Razvijen je odgovarajući algoritam za svaku od identificiranih metoda koja se pokazala kao pouzdana za detekciju veza između dvije ili više komponenata. Razvijeni algoritmi su zatim pretočeni u Java programski kod koristeći Eclipse razvojno sučelje. NX Open API je korišten za komunikaciju sa Siemens NX-om kako bi se pristupilo raspoloživim CAD podacima. Prikupljeni podaci su analizirani i preoblikovani u razumljiv format koji predstavlja vezu između komponenata.

Na temelju kreiranih veza iz CAD modela, kreirana je DSM matrica. DSM matrice imaju svojstvo pružanja jedinstvenog pogleda na arhitekturu sustava i kao takve su pogodne za vizualizaciju rezultata u ovome radu. Dodatno, metode za manipuliranje podataka koje su razvijene posebno za DSM matrice pružaju dodatne mogućnosti pri analizi sustava. Svaka od tih metoda ima specifičan cilj pa su stoga detaljnije objašnjene one koje su od posebnog značaja za podatke korištene u ovome radu.

Za potrebe testiranja razvijenog programa kreiran je CAD model na temelju kojega je nastala DSM matrica. Kreirana DSM matrica predstavlja rezultate algoritma koji sadrži sve identificirane metode za izvlačenje relevantnih veza. Rezultati su zatim uspoređeni s DSM matricom koja je nastala ručnim bilježenjem veza i koja služi kao pravovaljana referenca.

Ključne riječi: sljedivost, DSM, Siemens NX, CAD, kompleksni sustavi

SUMMARY

The main goal of this master thesis is to explore available methods for extracting component relations from the CAD model in order to support traceability. Siemens NX is used as a CAD tool because Daimler recently decided it will use that platform in the future product development process. At the time of writing this thesis, they are still in transition from CATIA. Some of the methods are based on Siemens NX features that are used while designing the product and others are developed based on new ideas that use information provided by the software API. The methods found are described in detail through the thesis.

A proper algorithm is developed for each individual method that has been identified as valid for extracting relations between two or more components. Developed algorithms are then put into proof of concept using Java in Eclipse IDE which uses NX Open API to communicate with the available Siemens NX CAD data. Collected data is used to extract relations.

Based on relations extracted from the CAD model, the DSM matrix is created. DSM matrices have the possibility to support traceability providing an additional way of looking at the system architecture and therefore are chosen as a tool in this thesis. Also, DSM post-processing methods enable an additional manipulation of data to get a better overview of the system architecture. Each post-processing method has certain goals and therefore the majority of them that are important for this thesis are explained in detail.

The final DSM is created based on the CAD model created for the purposes of this master thesis. Results are discussed and compared with a manually created DSM which is considered as a valid reference.

Keywords: traceability, DSM, Siemens NX, CAD, system complexity

PRODUŽENI SAŽETAK

U današnje vrijeme, kompleksnost proizvoda je sve viša. Prikupljeno znanje prilikom razvoja proizvoda je stoga vrijedno zbog toga što može biti ponovno korišteno, a to povisuje efikasnost i brzinu stvaranja varijanti ili potpuno novog proizvoda. Zato je važno osigurati što bolju sljedivost prilikom razvoja i ostalih procesa kroz koje proizvod prolazi.

Kuća kvalitete je jedna od metoda koja može biti iskorištena za poboljšanje sljedivosti jer povezuje zahtjeve za poboljšanje proizvoda s tehničkim funkcijama proizvoda. Same zahtjeve moguće je analizirati metodama koje analiziraju tekst ili veću količinu tekstualnih zapisa. Prisutne su i mnoge druge metode za poboljšanje sljedivosti, ali još uvijek postoji veliki prostor za unaprjeđenje sljedivosti korištenjem podataka iz CAD modela. Niti jedan trenutno prisutan PLM alat ne pruža mogućnost modeliranja procesa od zahtjeva do prodaje i održavanja, a da pri tome prati svaku akciju, analizira sustav i pomaže u obogaćivanju sljedivosti.

DSM matrice su jedan od alata koji nudi jednostavan način prikazivanja kompleksnih sustava, odnosno za potrebe ovoga rada, prikazivanja relacija između komponenata sklopa. Svaki element matrice može biti definiran tako da je vidljiva zavisnost s drugim elementima sustava. Tako se gradi struktura. Nakon definiranja sustava, grupiranjem elemenata ili spajanjem više matrica s istom vrstom i brojem elemenata, dobije se novi pogled na promatrani sustav.

Kreiranje veza između elemenata je iscrpan i dugotrajan posao. Vrijeme izrade DSM matrice ovisi o broju elemenata i kompleksnosti sustava, s time da ju moraju izraditi ljudi koji razumiju funkcije i veze koje postoje između komponenata proizvoda. Zadatak ovoga rada je pronaći metode koje će automatizirati i eliminirati ‘ručnu’ izradu DSM matrica.

Trenutno se autoindustrija suočava s mnogim izazovima od kojih su neki smanjenje CO_2 u ispušnim plinovima i ekonomska kriza. Budžeti za projekte se smanjeni, a razina kompleksnosti raste. Najbolji primjer su hibridna vozila koja jasno povisuju kompleksnost sustava zbog toga što se uvode nove tehnologije koje moraju raditi u skladu sa starima. Multidisciplinarnost timova otežava njihovo upravljanje.

Sociotehnički sustav je koncept koji nudi način za opisivanje takvih i sličnih sustava osnovnim funkcijama kao što su interakcija između čovjeka i stroja te analizom komunikacije između ljudi. Sociotehnički sustav je sačinjen od dvije osnovne grane: sociološka i tehnička.

Ovaj rad se bavi s tehničkom stranom sociotehničkog sustava koji uzima u obzir korištene alate i strojeve prilikom razvoja proizvoda. Rad podupire tehničku stranu sustava automatskim izvlačenjem važnih veza koje postoje između komponenata nekog sklopa. To je jedan od segmenata sociotehničkog sustava koji doprinosi boljem opisu cjelokupnog sustava.

Analiza autoindustrije osamdesetih godina je već upozoravala na visoku razinu kompleksnosti proizvoda, procesa i strategija za što je potrebno pronaći nove metode upravljanja. Bullinger očekuje daljnji i neprestani porast kompleksnosti sustava. Česta strategija, i često pogrešna je da se kompleksnost sustava pokušava izbjeći ili reducirati gdje god je moguće. Isto tako, Lindemann upozorava da se koncentracijom na samo jedan vid poboljšanja, primjerice 'Design for X', ne može pružiti visoka kvaliteta proizvoda. Uštede i ograničenja u jednom segmentu razvoja proizvoda idu na uštrp drugim segmentima. Nije nužno da kompleksnost sustava automatski predstavlja negativne posljedice za krajnji proizvod, već je određena doza kompleksnosti poželjna. Pitanje je kako upravljati kompleksnošću, a ne kako je izbjeći – sve u cilju kreiranja konkurentnog proizvoda koji ne narušava prvotne ciljeve i zahtjeve.

Sljedivost je koncept koji prilikom razvoja proizvoda pokušava objasniti na koje su sve načine elementi sustava povezani. Sljedivost nastoji dati odgovore na pitanja 'zašto?', 'kako?' i 'kada?' se nešto dogodilo, 'tko?' je odgovoran, na 'što?' se utjecalo i 'gdje?' se to dogodilo.

Tijekom razvoja proizvoda se uvijek teži ka boljoj sljedivosti, ali ako se informacijama za podršku sljedivosti ne upravlja na pravi način ili ako su one netočne, sljedivost negativno utječe projekt. Povisuje se cijene projekta, remeti se raspored, smanjuje se kvaliteta proizvoda i povećava se broj iteracija koje su potrebne da se projekt dovede do kraja.

Informacije za podršku sljedivosti dolaze u mnogo različitih formata – skice, tehnička dokumentacija, zabilješke sa sastanaka, bilješke i napomene radnika, proračuni, CAD modeli, razna izvješća i drugo. Štorga na temelju svojih istraživanja zaključuje da je kvalitetno upravljanje inženjerskim informacijama jedini način da se postigne efikasna sljedivost i predlaže načine kako to postići. Da bi podatak postao vrijedna informacija, u toku sljedivosti mora postojati nadogradnja koja ga u potpunosti opisuje. Sljedivost u tom slučaju pruža inženjerima, upravljačkoj strukturi i ostalim odgovornim ljudima bolje razumijevanje i priliku za kvalitetnije odluke temeljene na prije prikupljenim informacijama. Podatak može biti vrijedan bez obzira odakle potječe i u kojem je formatu. Bolja sljedivost znači bolji proizvod i siguran rast. Projekt TRENIN i SysMT su neki od projekata koji pokušavaju stvoriti

platformu koja pomaže u poboljšanju sljedivosti na temelju zapažanja njihovih autora i metoda koje su razvili.

DSM matrice su jednostavan, kompaktan i moćan alat za opisivanje relacija između elemenata sustava. Elementi mogu predstavljati proizvode, procese i ostale entitete između kojih postoji relacija. Mogućnost jednostavnog opisivanja sustava i naknadna analiza su prednosti DSM matrica koje su ih dovele do intenzivnijeg korištenja u raznim kontekstima. Postoje dvije glavne kategorije DSM matrica: statičke i vremenske. Zbog teme koju ovaj rad obrađuje, detaljnije su opisane kvadratne statičke matrice. Statičke DSM matrice predstavljaju sustav elemenata koji postoje istovremeno pa su zbog toga dobre za opisivanje arhitekture proizvoda. Elementi predstavljaju komponente, a relacije zapisane u matrici predstavljaju veze između komponenata. Identične komponente se prema istom rasporedu nalaze u zaglavlju i na lijevoj strani tablice. Veza između elemenata se bilježi kao točka na sjecištu tih elemenata u tablici ili se na istome mjestu stavlja broj koji predstavlja težinski faktor. Dijagonalna polja DSM matrice koja kreću iz gornjeg lijevog kuta i protežu se do desnog donjeg kuta nemaju značenje nego se mogu koristiti kao pomoć pri čitanju DSM matrice.

Standard korišten pri kreiranju DSM matrica u ovome radu je IC/FBD što se tumači kao da su ulazni podaci smješteni u stupce, a izlazni u retke. Simbol u lijevom gornjem kutu matrice ukazuje na odabrani standard. U nesimetričnoj matrici, elementi u redovima se mogu smatrati kao onima koji utječu na elemente u stupcima, a shodno tome elementi u stupcima se mogu smatrati kao oni na koje će utjecati elementi iz redaka. Glavna zadaća DSM matrice izrađene na bazi kvalitetnih informacija je da pruži bolji pregled cjelokupnog sustava i omogućiti vizualno uočavanje važnih područja. Statičke DSM matrice su često analizirane algoritmima za klasteriranje koji se temelje na reorganizaciji položaja elemenata u matrici i time grupiraju elemente koji su usko povezani. Tako se izlučuju strukturne jedinice koje čine podsustave većih sustava. Kombiniranjem više matrica iste domene se zove agregacija. Elementi u svim matricama se moraju referencirati na identične objekte koji se promatraju i samo tada je dozvoljeno vršiti agregaciju matrica, a time se objedinjuju različite veze u jednu matricu.

Sklop koji je pretvoren u CAD model je virtualna preslika modela iz stvarnosti te stoga sadrži sve potrebne veze između komponenti. U ovome radu su korištene informacije dostupne iz Siemens NX programskog paketa kako bi se prepoznale željene veze, ali na temelju ovoga istraživanja koje generalizira navedene metode, lako ih je primijeniti unutar drugih CAD

paketa. Zbir pronađenih metoda je kasnije implementiran u prototip koji je nazvan 'Assembly-to-DSM'.

'Constraint' je vrsta veze u Siemens NX-u koja predstavlja geometrijski odnos između dvije ili više komponenata, odnosno točnije – geometrijsko ograničenje. Sklop se gradi tako da svaki novi dio dobije svoja geometrijska ograničenja u odnosu na dio koji je već postavljen u virtualnom prostoru. Nakon što su umetnuti svi dijelovi i nakon što su za svaki od njih odabrana geometrijska ograničenja, dobije se konačni sklop. Vrijedi napomenuti da se odabirom vrste veze može utjecati na broj stupnjeva slobode gibanja svake pojedine komponente. Svaki puta se odabire prikladna geometrijska restrikcija od njih jedanaest ponuđenih.

'Proximity' metoda se temelji na analizi udaljenosti komponenata. Prednost ove metode je u tome što ne mora postajati već prije definirana bila kakva veza između komponenata, nego se na temelju minimalne udaljenosti određuje postoji li valjan razlog da se dvije komponente promatra kao da između njih postoji određena povezanost. Za to postoji varijabla s kojom se uspoređuje izračunata minimalna udaljenost, a vrijednošću navedene varijable upravlja korisnik. Ako je vrijednost mala, u obzir će se uzimati sve komponente koje su blizu jedna drugoj, a ako je vrijednost veća, u obzir će se uzimati komponente koje su blizu jedna drugoj, ali i one koje su udaljenije. Vrijednost ne smije biti prevelika ako se žele postići kvalitetni rezultati. To znači da se govori o rangu od 1-30 mm, ali vrijednost ovisi o vrsti proizvoda. Ako je proizvod zbit i komponente su jako blizu jedna drugoj, bolje je da je odabrana vrijednost što manja.

'Permanent joints' ili nerastavljivi spojevi predstavljaju veze koje se temelje na zavarima. Bilo koji tip zavara povezuje najmanje dvije komponente i time se kvalificira kao važna veza. Problem se pojavljuje kod segmentiranih i točkastih zavara jer se izlučuje puno veza koje su od istog značaja; svaka veza povezuje identične komponente. Jedno od rješenja je grupirati mnogo istoznačnih veza u jednu.

'Non-permanent joints' ili rastavljivi spojevi su tipovi veza koje stvaraju vijci i slične komponente koje nisu trajno čvrsto vezane za bilo koju od komponenata u sklopu. S obzirom na to da vijak dodiruje ili je vrlo blizu komponentama koje spaja, uzima se kao da prijašnje metode koje analiziraju udaljenost između komponenata već prepoznaju ovaj tip veze.

Kako bi se ispitalo navedene metode napravljen je model koji sadrži sve navedene tipove veza. Na temelju tog modela, 'ručno' je napravljena DSM matrica koja će u kasnijim ispitivanjima poslužiti kao referenca za provjeru vjerodostojnosti rješenja koje će generirati prototip. Važno je napomenuti da 'ručna' izrada DSM matrica iziskuje puno vremena i stručne ljude koji poznaju sustav. Jako kompleksne sustave postaje gotovo nemoguće modelirati pomoću DSM matrica.

Prototip rješenja je izrađen zbirom alata koji su u konačnici omogućili prikaz rješenja u obliku DSM matrice. CAD model je napravljen u Siemens NX programskom paketu, a informacije iz modela se dohvaćalo koristeći NX Open API funkcije koje su pozivane iz Java programskog jezika. Eclipse IDE je korišten kao razvojno okruženje za Javu jer je besplatan i pruža dovoljno napredne alate za provjeru ispravnosti programskog koda uz kvalitetnu dokumentaciju koja je neophodna za ovakve pothvate.

Prototip je izrađen s MVC konceptom programiranja na umu. MVC je način programiranja koji predlaže odvajanje ključnih dijelova programa u zasebne grupe kako bi u kasnijem razvoju i dopunama bilo lakše upravljati promjenama. Osnovne skupine su 'Model', 'View' i 'Controller' gdje 'Model' predstavlja format podataka koji će služiti kao rješenje algoritama, 'View' definira način prikaza tih podataka, a 'Controller' je dio koji poznaje načine prikupljanja podataka. Ovakav pristup programiranju je važan u ovom slučaju jer je izrađeno rješenje jedinstveno za Siemens NX. Budući da je opisane metode za izvlačenje relevantnih veza moguće promatrati kao generalno primjenjive i u drugim CAD programskim paketima, onda je važno da je moguće te metode prilagoditi drugim načinima za upravljanje podacima koje nude primjerice SolidWorks ili CATIA. MVC pojednostavljuje implementaciju novih modula koji podupiru ostatak CAD programskih paketa zadržavajući neke od postojećih dijelova programskog koda.

Dobiveni rezultati analize CAD modela dolaze u JSON formatu koji sadrži sve prepoznate veze. Zatim su JavaScript, d3.js, HTML i CSS tehnologije iskorištene kako bi se 'sirovi' podaci vizualizirali u formi DSM matrice.

Analizom dobivenih rezultata i usporedbom s 'ručno' izrađenom DSM matricom došlo se je do zaključka da je metoda automatskog prepoznavanja veza iz CAD modela bila preciznija. Pronađena je jedna veza koja je slučajno bila zanemarena u 'ručno' izrađenoj DSM matrici.

1. Introduction

With today's increased level of product complexity [1] during product development, traceability is an approach which naturally fits into an environment where future decisions need to be made in a fast and precise way while at the same time considering the context surrounding the product. Knowing the decision-making process and how conflicts inside issues were resolved helps with future versioning and product variants. Knowledge gathered during the development process can be reused and it helps to achieve better efficiency while shortening the time for finalizing similar future projects [2]. New members of the team are able to learn about the history of the product and it can help them to easily integrate into the team. This is why it is important to have good product traceability.

While developing mechatronic systems, every module of the system with a specific function should be applicable to a different context of another system. This enables future development to take an existing project and use it in another as a module which performs its function regardless of a new context [3]. Unfortunately, there are often new requirements that change existing modules and the new changes can affect other parts of the system. Knowing which parts of the system will be affected by the change facilitates foreseeing which resources will be required to achieve a well-integrated solution.

There are existing methods that support traceability when dealing with product requirements which are then translated into technical functions, e.g. House of Quality. House of Quality supports traceability based on the knowledge about the relations between functions and requirements [4]. Requirements alone can be analyzed with existing data mining and text mining techniques [5]; however, the effective analysis of product component relations remains a weak link in the product traceability path.. There is a large space for improvement in traceability by the use of component relations in product assembly (Figure 1).

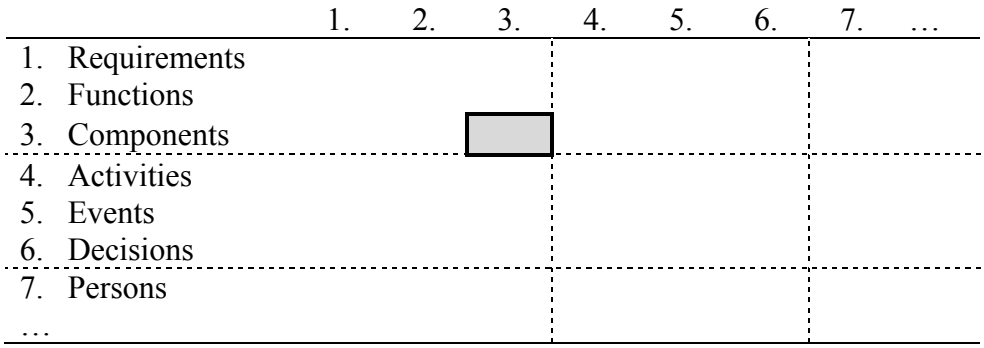


Figure 1: Traceability matrix - grey area represents the area of interest

The existing tools and methods for product development are not adapted to deal with today's complex dependencies in vehicle development [6]. Therefore, the effort to manually model Engineering Object Relations (EORs) is the main obstacle for using traceability tools in practice. None of the existing commercial PLM tools support the modeling of EORs to the full extent [7].

The Design Structure Matrix (DSM) offers simple way to present the view of assembly component relations (Figure 2). Every DSM element can be defined with regard to the relations to many different elements of the system. In that way, the system structure is built [4]. After the matrix has been created, post-processing methods enable efficient use of the matrix for analysis and further use such as search, visualization, modularization etc. For example, it is possible to detect groups of closely related components by clustering them [8] and this could later lead to building a single component with all the necessary functions, thus optimizing the number of system elements. DSM matrices serve as a tool to better understand system structure so it is relevant to have good data for creating the matrix.

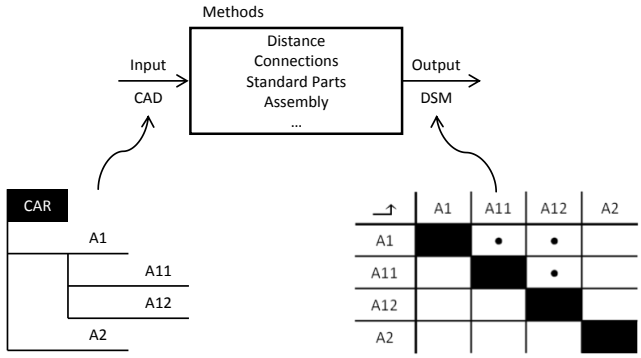


Figure 2: Master thesis goal – developing methods for finding assembly component relations

A complete structural representation of the product model with described EORs between Engineering Objects (EOs) should enable engineers to recognize which EOs of the complete system will be affected by the change of the desired EO. Creating EORs manually consumes a lot of time and involves a lot of people familiar with the system design [9], thus new methods identifying the relevant parts as well as the relevant relations of complex mechatronic systems have to be developed.

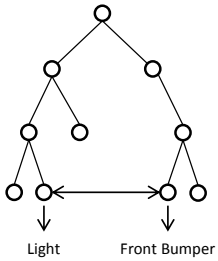


Figure 3: How are the Light and Front Bumper related?

This thesis describes ways in which traceability can be supported by finding new methods for automated and semi-automated extraction of EORs between given EOs (Figure 3) to eliminate manual work which is nowadays needed for extracting these relations. This contribution helps to better understand the product assembly structure and produces data that can consequently be used for better product management.

2. Understanding system complexity and industry practices

Recently, automotive industry faced few important challenges. Significant CO_2 restrictions, market drifts and economic crisis which demanded sudden changes in project management. Economically, budgets for individual projects were reduced and in the other hand system complexity rose due to the demand for fuel efficient vehicles. Fuel efficiency is trying to be achieved by looking into green energy sources and building hybrid vehicles. Hybrid vehicles made system complexity harder to handle because new approaches had to be introduced to achieve combination of multiple power trains working together seamlessly. Old well established and well known methods had to be modified. Now, different fields of science are coming together and therefore new model had to be developed to manage this complexity of all newly introduced artifacts in intensified cooperation, joint venture and network structure in manufacturing [6].

One of the methods for understanding system complexity is introduced as sociotechnical system approach. It defines basic system functions as interaction between human and machine and as communication between humans. Product development process can then be modeled based on those two functions. With this approach it is possible to model the system on lower level in contrast to what was possible before, therefore it gives better overview of the system. Methods and tools like CAx and PLM technologies had to be embraced and put into the everyday process of product development. In the end, all of this should come together in a model describing how to deal with the issues related to product development under consideration of social behavior, design methodologies and IT services [6].

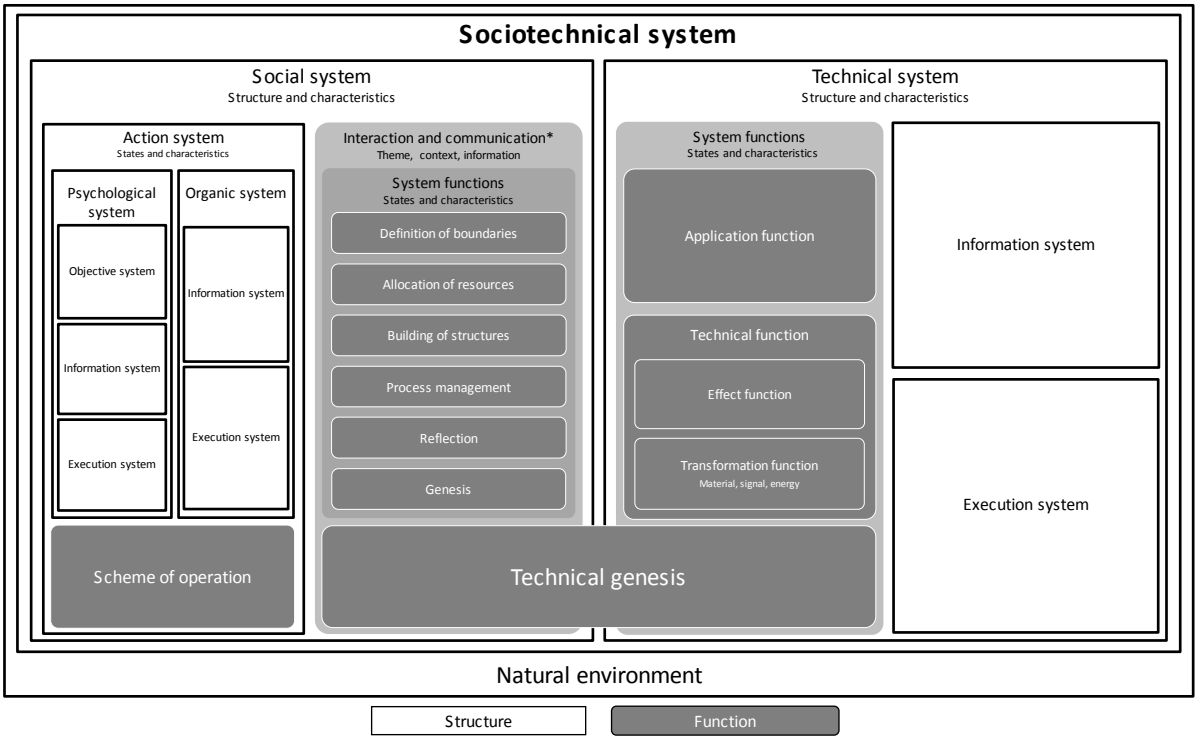


Figure 4: High level meta-model structure of sociotechnical system

As humans interact with the machines (for example computers) which often provide us with tools for solving different scenarios, it's important that those tools are efficient and produce quality results [6]. If produced results were not correct, it's not possible to support traceability. Efficiency should be always taken into account as efficiency of the whole project depends on the efficiency of partial steps in the project development.

This thesis supports technical side of sociotechnical model by providing methods and tools for extracting relevant relations between components in complex system architecture.

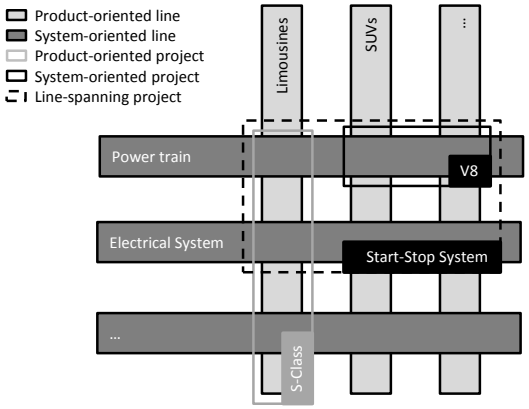


Figure 5: Projects within matrix-organization [1]

A multitude of new technical dependencies emerged due to increased integration of mechatronic components in modern automobiles. Consequently, not only product complexity but also process complexity increased.

The number of engineers involved in the development of complex technical systems has been rising significantly over last hundred years [1]. In 1885, Karl Benz alone built the first vehicle powered by a gasoline engine. Today, the development of a modern car involves several hundred people. Rising product complexity is directly proportional to rising complexity within the organization [6]. Organization is responsible for partitioning complex projects in smaller ones and for rejoining them afterwards. It has the responsibility to deliver and verify the solution.

When trying to understand company and development complexity, it's necessary to understand its organizational structure. Hierarchy often regulates amount of power and responsibility within development projects. Right balance of power is important for the success of collaborative development.

Matrix-organization (Figure 5) is a common way of arranging departments and roles inside companies where two types of lines are facing each other. Figure 5 suggests vertical product-oriented and system-oriented as horizontal lines. Vertical product oriented program teams have the responsibility to integrate different systems into specific products whereas the system-oriented departments focus on the integration of specific systems into different products [1]. Projects span through multiple lines intersecting with one or few product- and system-oriented lines.

Process and development control are one other aspect of organization. Processes define the sequence, timeline, expected results and responsibilities of teams. The gateway processes [1] in automotive industry can be seen as a validation door for merged sub-processes. Those gateways are used to validate the synchronized results from different teams, to track the status of the project and to define corrective actions. Today, processes in the automotive industry are rather product-, not system-oriented.

The development of a new system is initiated either top-down or bottom-up within the organizational structure. Top-down initialization comes after certain management attention and sets a mandate for development. It is often a quick and urgent response to new customer demands. Special project teams are under time pressure and they are working in close

cooperation with production development to ensure fast integration [1]. Figure 5 illustrates the need for engineers from different departments in case of developing Start-Stop system which came as a demand for reducing full consumption. Standard processes for this king of cross-department project usually don't exist and have to be defined on the fly. It is obvious that this kind of development practice causes deficiencies in development robustness. Bottom-up on the other hand is initiated by ideas within the departments. New specification comes as a result of customer feedback, supplier experience, repair and servicing shops and possibly from other sources that base new ideas on their own experience. There is usually less time pressure but projects are not mandatory to be integrated into existing systems. Projects gain attention only by achieving good results and then they might be converted into top-down system development project. Top-down and bottom-up system approaches are both dealing with same problems regarding collaborative work and cross-linked information within those processes. Problems lay in inconsistent, hard to retrieve or outdated information across departments, low transparency about changes and low transparency about impact of changes [1].

People tend to search for new tools and methods for accomplishing the task they have in front of them. This happens separately and simultaneously in different areas in the company, leading to inhomogeneous set of applied tools and methods.

The current gateway processes mentioned in previous paragraph define goals, timelines and responsibilities across different teams but do not assist the coordination of information in between the gateway points.

Current practice from a tool perspective in German automotive industry suggests that Microsoft Office suite, MATLAB- and Simulink-product family tools have become de facto the standard [1]. Furthermore, there is a wide diversity of tools used by specialist departments which help them to achieve better results but stresses an ease of integration and systematic coordination of engineering data across the company. Non-coordinated data contributes to unnecessary iterations which manifest as rising product cost and possibly can have an impact on product quality. Inevitably, the number of tools and amount of generated data will persist to increase (Table 1). Companies will struggle more and more with more complex data and organization management; therefore there is a strong need to support the aspects of interaction and communication within the development process and across organization structures.

Table 1: Relevant functionality in engineering tools to enable traceability [7]

PDM	CAD	Office tools
Project management Document versioning Workflow mechanism Engineering change management Search/Query engine Report generator	Feature tree (structure of the CAD model) Associatively links between assemblies and parts File versioning 3D model characteristic management	Changes tracking mechanism Document properties management

Analyses of the automotive industry of the 1980s already showed a high complexity concerning processes and products, and numerous strategies for its systematic management have since been designed [10]. Bullinger mentioned that the trend towards increasing variant numbers and product complexity will continue in the automotive industry [11]. An approach which potentially addresses the mentioned problems was introduced in the late 1970s. It focuses on the management of interdependencies between software requirements and other artefacts in software engineering and is called traceability [1].

3. System traceability

System traceability is the power of knowing how all things done in process of creating a product relate to each other. Traceability should be able to provide answers to 'why?', 'how?' and 'when?' specific events occurred, 'who?' is responsible, 'what?' was affected by those events and 'where?' did it happen.

When talking about supporting traceability, there are some bad examples to have in mind. If traceability is not well implemented or the information entering traceability support system is not correct, it can impact project cost and schedule. Decrease in system quality, increase in the number of changes and iterations in design, loss of knowledge due to misunderstanding and misleading information are some of the common problems that can occur [7].

During product design process, information is recorded and evaluated. Information occurs in variety of formats such as sketches, drawings, notes, meeting recordings, CAD models, production drawings, calculations, reports and other [7]. Storga therefore argues that the effective traceability is highly dependent on the effective utilization of existing engineering information and records.

In order to fully understand an instance of information, it is important to know the circumstances in which it has been developed and recorded. Traceability then allows engineers to better understand and make better judgments about their future decisions based on the previously collected and now well known facts. This is why it is important to leverage all relevant information no matter where it originated, no matter of its format and no matter where it resides [7] in order to help the company provide better services, produce better products and therefore ensure healthy growth.

Relations that exist within product development lifecycle help anyone who may be concerned to better understand the rationale behind previously made decisions [7]. In order to build quality network of relations, different research groups approach traceability issues from different perspectives. They state that it's important to address knowledge integration [12], communication, handling complex dependencies between requirements and components [13], ontological retrieval of unstructured documents [14] and other areas for effective traceability to be ensured. Storga suggests four main areas to focus on in order to understand what is necessary to have complete traceability support. Those are requirements, changes, characteristics and decision traceability issues.

Project TRENIN [7] (Traceability of Engineering Information) is one of the approaches trying to support system traceability. TRENIN architecture proposes four key elements that should provide sufficient knowledge on how to describe complex system traceability: traceability-point, -record, -engine and –framework. Traceability point should be seen as an external event related to the product development process. Traceability record should be dynamic container of links between system elements or any information in the database. Links should be enriched with properties and structure in order to provide the context for every relation. Traceability engine should be another layer on top of traceability records which should enrich them with ontology and vocabulary in order to provide additional context to engineering information for it to become even more useful. TRENIN traceability framework should be complete architecture of elements mentioned before which is independent but later integrated in PLM systems. In general, project TRENIN addresses shortcomings related to traceability functionalities in existing engineering tools and wants to provide new framework for integration to deal with this problem.

System modeling and management tool (SysMT) is developed by Daimler and is trying to deal with a multitude of mechatronic systems that are used for different vehicle versions with minimal adaptions. Its goal is to enhance traceability and to better describe EOs and EORs by intercepting management, concept, application and properties during the product design [15]. Qualitative analysis of this traceability approach shows that SysMT implements most of the required traceability functions for modeling and monitoring design process in comparison with Teamcenter by Siemens or Catia V6 by Dassault Systems and some other products. It's worth noticing that there is no comparison with PTC Windchill PLM system [1].

To support traceability in technical aspect of sociotechnical model and to ensure possibility to show the relations that are important for describing system architecture, this thesis will embrace DSMs as a tool to facilitate extracted dependencies between system components. DSMs post-processing methods will allow to further dive into system architecture analysis and therefore provide more insight about the product structure itself.

4. DSM (Data Structure Matrix)

This chapter will present only the segments of DSMs that are important for understanding, creating and manipulating DSM matrices for the purpose of this thesis. There is a broad spectrum of methods and applications for DSMs that can be applied in other cases.

DSM matrix is a simple, compact and powerful tool for representing relations between objects of any type. Objects can represent products, processes or organization related information, but to explain the possibilities of DSMs, objects don't have to differentiate. They are always part of a complex system. The advantages of DSMs have led to their increasing use in a variety of contexts, also becoming widely used tool in engineering because of the ability to represent and analyze system structure.

	A	B	C	D	E	F	G	H	I	J
Element A		•	•							
Element B			•						•	
Element C					•	•				
Element D								•		
Element E		•					•	•	•	
Element F		•		•						
Element G	•		•		•					
Element H										•
Element I		•			•					•
Element J								•		

Figure 6: Sample DSM

There are two main categories of DSMs: static and time-based. Point of interest here are the static ones. Static DSMs represents system elements existing simultaneously and therefore are good for modeling product architecture. Product architecture is the arrangement of functional elements into physical chunks [16]. Every interaction between chunks should be well defined and chunks should implement one or more functions entirely. DSMs serve as a tool to represent those interactions within system architecture.

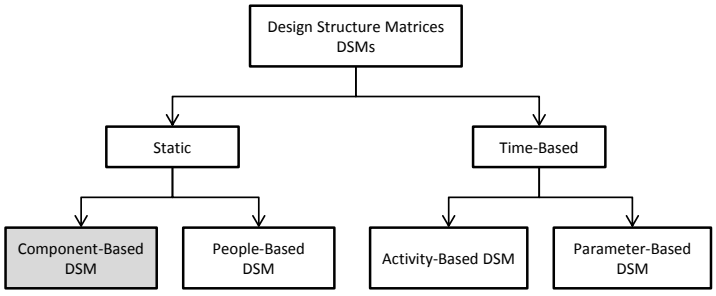


Figure 7: Basic DSM classification [16] [4]

Static intra-domain DSM is essentially the square matrix. Intra-domain matrix is defined if the elements in rows and columns are from the same domain/type [4]. System elements are placed down the side of the matrix as row headings and across the top as column headings in the same order. Elements can be represented in headings as names or numbers that indicate the connection with the element. Intersections between nodes (rows *i*, columns *j*) represent unity (*ij*) marked with a symbol in binary matrix or number in weighted matrix (Figure 8). Weight numbers can describe anything that is decided to be important, for example connection strength of unity, number of different types of connections between nodes or there could be the unity cell that is split into a table for more complex representation of relation [17].

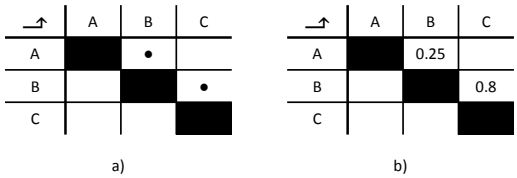


Figure 8: a) Binary DSM, b) Weighted DSM

The diagonal elements of the matrix don't have any interpretation in describing the system. They are usually either left empty or blacked out, although many find it intuitive to think of these diagonal cells as representative of the nodes themselves. Thinking about the diagonal nodes as representatives of themselves, it is possible to develop useful logic that helps us quickly interpret element interactions in directional (non-symmetrical) matrix that is modeled based on a process diagram (Figure 9). This logic doesn't apply to symmetric DSM matrix.

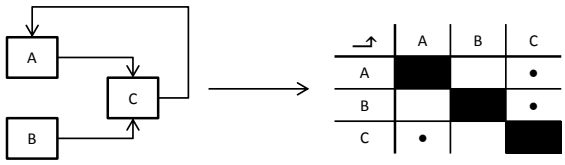


Figure 9: DSM created from process diagram

Before the explanation on how to use DSM diagonal cells, it should be mentioned that IC/FBD convention for reading matrix is chosen. IC/FBD [8] convention is used for DSM matrix reading which means that DSM has inputs shown in columns, outputs in rows; hence, any feedback marks will appear below the diagonal. Symbol in the top left corner indicates the chosen convention. IR/FAD convention is inverted.

Logic that helps us better understand an element relation by looking at one of the diagonal cells is the following (Figure 10):

- one diagonal cell represents one element ($i=j$),
- depending on the chosen convention for reading DSMs, IC/FBD in this case, cell's column (j) represents the element's inputs and cell's row (i) represents the element's outputs,
- for example in Figure 9, bottom right diagonal cell represents element C which gets inputs from element A and B while providing output to element A.

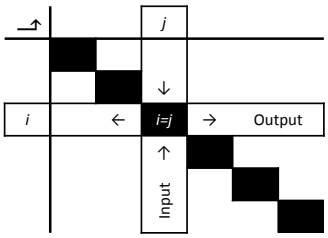


Figure 10: Diagonal cell logic

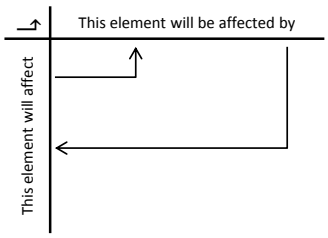


Figure 11: DSM dependencies

In directional non-symmetrical matrix rows can be considered as elements that will affect elements in columns and elements in columns can be considered as elements that will be affected by the elements in rows (Figure 11). As it may seem very simple, it's important to have a good understanding on how to read DSMs because as they become bigger and more complex those simple principles help to better understand the relations between elements.

The classic approach towards better understanding of the complex system is to model it. Systems are modeled typically by decomposing actual structures into subsystems we know relatively more about and by noting the relationships between them [16]. Dependencies of a system form structures, such as a sequential chain of dependencies, a loop, or a hierarchical tree. Thus, if system structures can be identified, it is possible to predict system behavior [4].

What if the system architecture is very large and complex? DSMs are still a great tool to represent the relations between elements, visualize it and offer methods for further system analysis, but it would be a great advancement if the decomposing and noting the relations wasn't manual.

4.1. System complexity

In almost all relevant sections of engineering, a steady increase of complexity can be observed [4]. Often, complexity management is understood as the management of product

variants only. Structural complexity management exceeds this, as further disciplines and aspects of product design can be considered simultaneously. Every system, a technical product composed of different parts or a project consisting of process steps, people and documents, is characterized by dependencies between the system's parts. Complexity can arise from every aspects of development process.

The core idea of mass customization is an optimal combination of mass production with customized product specifications. Companies try to achieve this by turning low internal complexity into high external complexity for specific customer requests. The better companies can control their existent complexity, the more customization becomes possible to market quickly and at reasonable costs. The better system complexity can be controlled, the better the structure can be adjusted in order to serve the desired functional objectives.

The structural complexity management simultaneously takes in consideration multiple aspects of dependencies. Geometric and functional dependencies between technical components can be processed together in order to describe system behavior. This is referred to as Design for X approaches in product design [18]. The X stands for a large variety of possible optimization targets. The Design for Assembly or the Design for Modularity is especially well established. Methods summarized by the name Design for X only aim at one single optimization.

A common strategy for facing complex problems is to avoid or reduce complexity whenever possible and one can assume that complexity must be prevented by any means. Lindemann argues that focusing only on one specific objective, e.g. cost or assembly, cannot provide comprehensive system improvements, because one system dependency adaptation can spread through multitude of further system elements [19]. Complexity does not automatically represent negative characteristics in product design. A specific level of complexity can be useful to permit the flexibility; if, for example, the implied complexity refers to the quantity of product variants offered, an increased product variety can better match different customer requests that arise [20] and therefore provide competitive advantages [21] [22]. Also, complexity reduction may decrease competitiveness. Controlling complexity stands for the ability to handle the complexity of processes and their effects without jeopardizing their targets [21].

Figure 12: Aspects of complexity in product design [4]

Improper simplification or the extraction of single aspects must be avoided. Users can draw incorrect conclusions that may result in an unfavorable impact to system domains that had not been considered (Figure 12). In contrast, all aspects must be excluded from considerations that are irrelevant for the specific question. There are four fields of complexity in product development: market, product, process, and organization (Figure 12). Mutual connectivity of those four fields is the reason why considering only one (isolated) aspect of complexity is often misleading.

In general, the complexity of each system can be reduced if it is possible to eliminate elements and relations while keeping the existing system's functionality. Furthermore, it is possible that the reduction of complexity in one domain will increase complexity in another one. This would be the case, for example, if a simplification of product components is accompanied by a more complex production process [4].

Often an optimized product structure can help reduce unnecessary product complexity. Related approaches aim at a modular product design [23] [24]. The objective is to design subsystems that are generally independent from each other.

Users must always be able to have access to the overview of the considered system and the acquired content, even if the data acquisition is split up in several workshops. For example, assembly model can be built through several departments. Users need representations that focus on case-specific aspects, where all relevant dependencies are integrated. Extracting relevant information form assemblies can help support this claim.

Consequently, the avoidance of complexity cannot represent the only strategy for addressing problems of complexity. If complexity can be controlled, it does not necessarily imply negative aspects but can provide competitive advantages in product design. It turns out that it

is not about how complex the system is but how well the complexity can be managed for the customer's good.

4.2. Visualization

Visualization is a form of knowledge compression [25] because a seemingly simple image can take vast amounts of structured or unstructured data and compress it into a few lines together with colors that communicate the meaning of all that data quickly and efficiently.

Grayscale (Figure 13) allows us to see much less notable difference between different data levels in comparison with using pseudocolor imaging. Pseudocolor image is derived from a grayscale image by mapping each intensity value to a color according to a table or function describing color values for different intensities [26]. We can only differentiate a few dozen different grayscale intensity levels but we can differentiate thousands of different colors. There is no finite answer on how to choose the right set of colors but common associations among humans is that red and blue very clearly highlight high and low values [26]. Learning about those principles is helpful in creating better DSMs that will be more appealing to the user. The point is to highlight differences so the user can easily see things that relate to the issue.

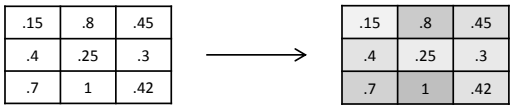


Figure 13: Grayscale color mapping

Structural consideration of complex systems requires suitable possibilities of information visualization, for example by graphs or matrices, and efficient computational approaches. For the interaction with complex systems in product design, different methodologies provide possibilities of system modeling, visualization techniques and computational approaches. DSMs provide simple and easy solutions that address those needs.

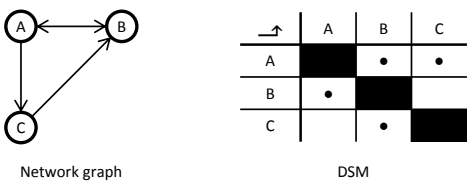


Figure 14: Network graph and DSM created from the same data sample [27]

Generally, matrices and/or graphs are applied for visualizing dependency information in product design. Even though the mutual transferability of both forms is mathematically formulated, only a few applications make use of their combination in order to benefit from the advantage of both visualizations. Despite the possibility of mutually switching between matrix and graph representations, information losses can still occur [28]. This thesis is focused on DSM matrices because of the available computational methods and fair visualization possibilities.

There are some disadvantages regarding the visualization of data through matrices but DSMs are still commonly used in engineering. Some of the disadvantages are stated below and are mainly related to the ease of recognizing important elements in comparison with force-directed graph (Figure 15) [4].

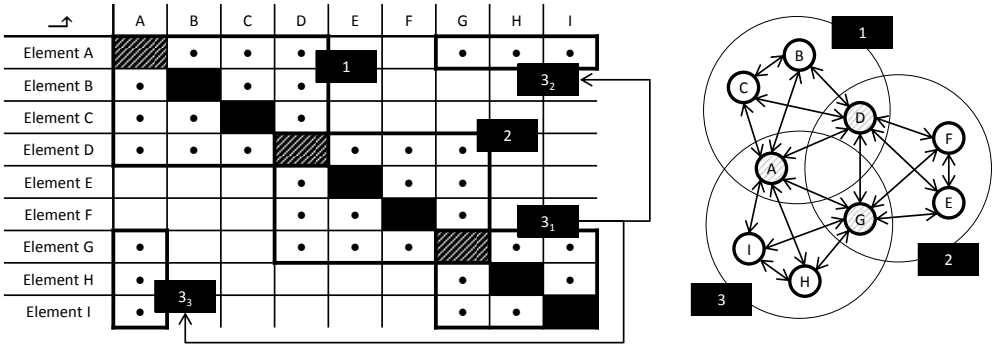


Figure 15: Broken cluster in DSM and the same representation in network graph [29]

Clusters indicate that the group of elements has a strong interaction between them. Figure 15 shows that the matrix on the left has two complete clusters containing four elements (indicated as 1 and 2) and a third one containing three elements (indicated as 3_1). The clusters 1 and 2 as well as 2 and 3_1 mutually overlap by one element (D and G). However, the system's structure overextends the matrix's capabilities of representation. This can be seen from the graph depiction at the right side of Figure 15, visualizing the same structure exactly. The network graph clearly illustrates that the system includes three clusters (1, 2 and 3), each containing four elements and overlapping with the other ones in one element. This specific constellation results in a fourth cluster comprising three elements (A, D and G) and overlapping with all other clusters in two elements. This constellation cannot be displayed intuitively in a matrix form. Only one cluster (2) can be aligned with and connected to two others (1 and 3_1). As the cluster in the lower right corner of the matrix has to be linked to the clusters 1 and 2, it becomes visually split up. The attempt to align all four elements (A, G, H and I) belonging to cluster 3 side by side would split up cluster 1. This shows that structures

comprising highly interrelated subsets may require more possibilities for their representation than available in common matrices [4].

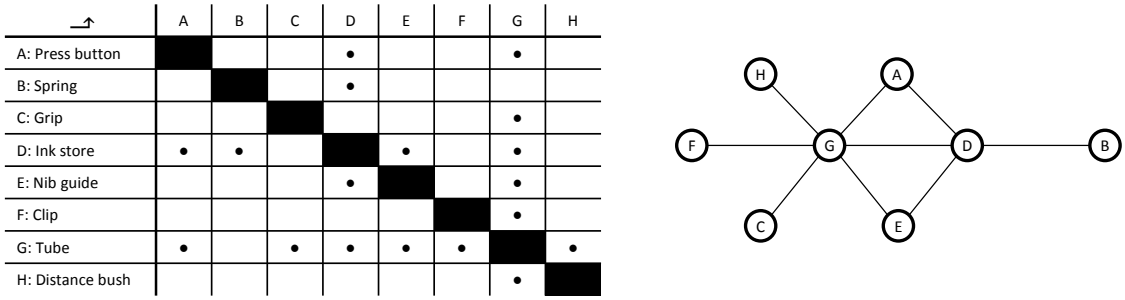


Figure 16: The structure of a ballpoint pen represented in DSM matrix and non-directional force-directed graph [4]

A force-directed graph is applied with non-directional dependencies between the system components (Figure 9). These dependencies are represented in the matrix as bi-directional dependencies, symmetrically aligned to the matrix diagonal – for example the Tube links to the Distance bush and the Distance bush links to the Tube. It must be mentioned that representing the dependencies at one side of the matrix diagonal would be sufficient for mediating the information included in the force-directed graph. However, due to the applied reading direction of matrices (Figure 11), crosses at only one side of the matrix are interpreted as unidirectional linkages.

Even if both representations contain the same information, the implied structure is easier to understand by the force-directed graph: The Tube represents the core element, as almost all other elements are linked to this. In the graph representation, the Tube is located in the center, which makes its structural relevance intuitive.

People can also extract this information directly from the matrix, if they are used to this depiction – the row and column associated with the Tube are the ones most filled with dependencies.

The more elements exist in a structure and the more interlinked these elements become, the less appropriate the matrix depictions seem to be. In fact, the depiction of one constellation can hide other ones. The preceding examples suggest that graphs outmatch matrix depictions and qualify for the mediation of system structures for the user. Whereas matrices are suitable for purposes of information acquisition, in structure representation they only seem to possess advantages for specific constellations, such as isolated clusters. Sometimes it's better to

visualize sections of relations with force-directed graph because it can more clearly show the important intersections of relations (Figure 16).

The general objective of DSMs is to permit users a global system overview as well as to provide focused views on specific aspects in order to obtain a better system understanding. Visualized data helps engineers better understand product structure.

4.3. DSM post-processing methods

A high quality of captured data is the key factor for the accuracy and significance of all further structure interactions. Deficiencies in the data acquisition can hardly be corrected later on and often result in data that is useless for analysis and interpretation. In particular, if mistakes made during data acquisition remain undiscovered, the resulting low quality of information can become critical. Even correctly executed network analyses can lead to misleading findings and unsuitable actions (Garbage-In-Garbage-Out[1]).

Beyond approximately 30 elements in the DSM matrix, manual analysis becomes almost impossible [4]. In that case, depending on the type of matrix in possession, there are different methods that are applicable for data manipulation.

Several matrix-based algorithms applied in engineering originally emerge from algorithms developed in graph theory [22]. They provide the mathematical basics for analyzing dependencies between system elements and in fact matrices only represent graphs in another form (Figure 14, Figure 15 and Figure 16). The field of graph theory provides the fundamentals for many methods applied in product design [30].

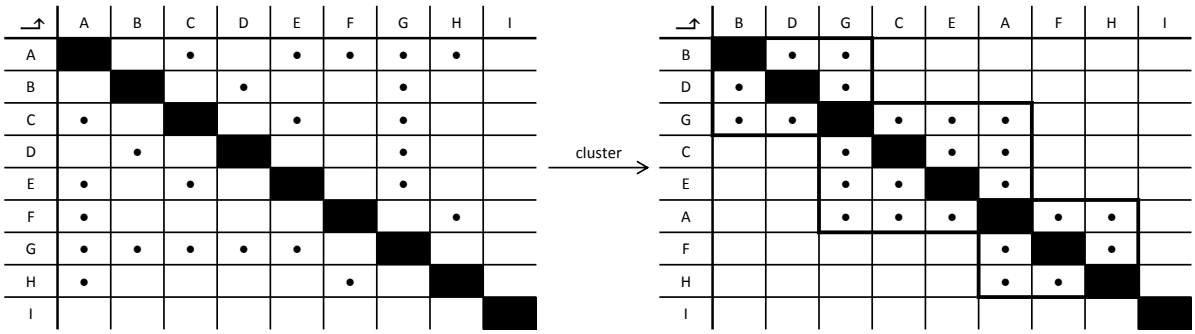


Figure 17: DSM clustering – three overlapping clusters [31]

[1] Pertaining to the concept that, if meaningless or erroneous data, i.e., "garbage," are entered into a data processing system and are processed by that system, the output will also be meaningless or erroneous, i.e., meaningful or correct information cannot be obtained from "garbage" no matter how the "garbage" is processed. Common abbreviation: GIGO.

Static DSMs are usually analyzed with clustering algorithms [16] which are based on the realignment of elements in order to closely visualize related groups of elements. By appropriate realignment of the element's rows and columns, DSMs support the identification of structural clusters (Figure 17).

As clustering is a mathematical model for realigning rows and columns in the matrix, there are multiple algorithms and approaches and not all of them address important problems that arise when trying to apply clustering algorithm. Authors Yu, Yassine and Goldberg [31] conclude their paper with few important guidelines which point out the issues that should be addressed when developing clustering algorithms:

- bus modules (Figure 18),
- overlapping modules (Figure 17),
- designed to overcome DSM manual/human clustering problems,
- tuning capability to mimic human expert clustering…

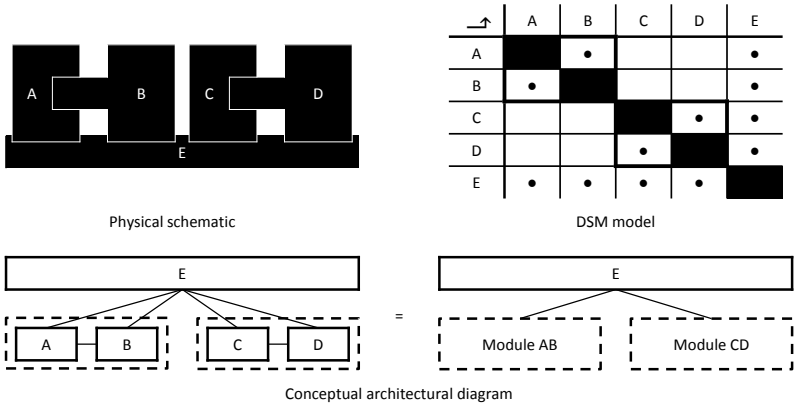


Figure 18: A simple bus and two modules [31]

Combining multiple DSMs into one matrix is called aggregation [4]. Aggregation is possible only when all matrices contain the same number of elements while every element references to the same base object in the system architecture. DSM needs to be Component-Based (Figure 7) and symmetric. Since all the matrices should be the same in terms of basic template, differences are in the relations that they represent. Different points of view on the same system architecture will create different relations for the same unions. For example, one system architecture can be observed from functional, geometric and feature point of view (Figure 19). Combining matrices can give clustering algorithms opportunity to create clusters based on multiple relation types and therefore provide better overview of the system architecture as a whole.

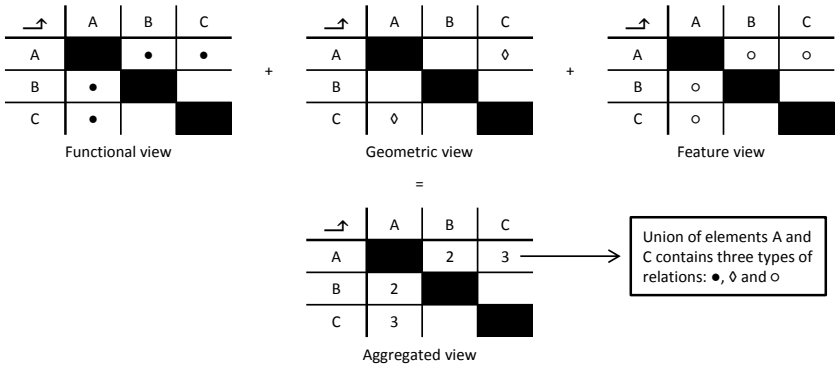


Figure 19: DSM aggregation

Aggregation and clustering are DSM post-processing methods that create better visual representation of the system architecture by combining and rearranging DSM elements.

5. Conceptual development of methods for extracting product assembly relations

Methods for extracting valid relations between components from finished product assembly are developed to support project traceability. Finished product assembly is virtual representation of a real product therefore it is created in a way that represents real system structure. System structure consists of multiple components that are related to each other. There are multiple types of relations that exist but the point is to extract all of them automatically.

For this to be able to develop, existing design methodology and features for creating relations need to be explored. In this case, features used for creating an assembly in Siemens NX will be used to find important component relations. Even though the concept of extracting relevant relations is based on features and possibilities of Siemens NX, it can easily be generalized and applied in different contexts where components are in some way geometrically related to each other and when there is a virtual representation of them available in form of virtual system structure. Virtual system structure in this case is represented with CAD assembly model.

Why to develop methods for extracting relations between system components if those relations are already defined in drawings and somewhere in project documentation? The key word is from the previous sentence is ‘somewhere’ and this is the word on which the answer is based on. When there is a virtual representation of system structure available, all the relations are centralized! If the DSM matrix based on this system structure has to be created, the easiest and most logical way to extract the relations is from the source that contains all of them in one place. This is where the power of well-developed methods for extracting important relations from product model comes into place. Assembly-to-DSM (Figure 20) is the common name for all the methods described in this thesis and therefore it represents the process of getting DSM matrix from product assembly.

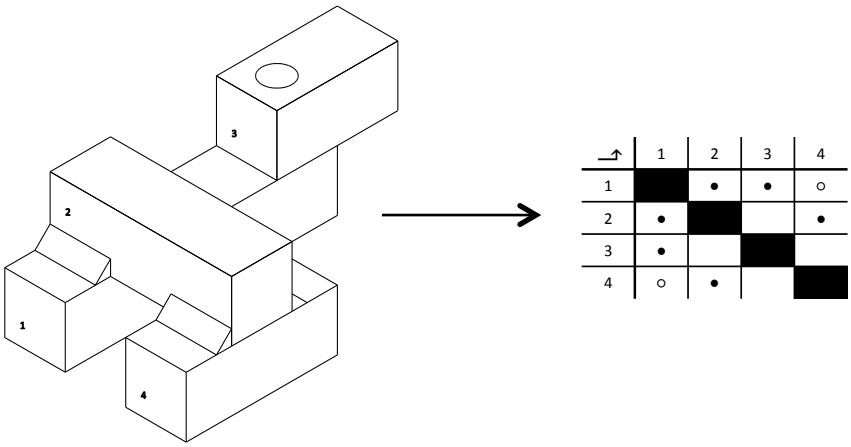


Figure 20: Assembly-to-DSM

Next few headings will present the methods for extracting important and valid relations between system components. Concepts for extracting important relations will we explored and explained together with the problems encountered during the development. Problems are addressed and adequate solutions are suggested. Defined conceptual methods will be later used for creating the prototype application in Java.

5.1. Constraints

Constraint is a Siemens NX feature that is a synonym for geometric relation. By definition it is a restriction and here it is geometric restriction. Product assembly is created by defining one or multiple constraints between inserted components, therefore creating unique structure that relates to physical world.

Common practice is to start building the assembly with base component. Base component serves as a foundation for other components to be constrained to. As each component is added to an assembly, specific constraints are used in order to correctly position the component in the assembly. Not all components should be fully constrained though. Difference between fully and not-fully constrained components is in the degrees of freedom to move. Some components such as gears need to rotate and other may need to slide in one or more directions.

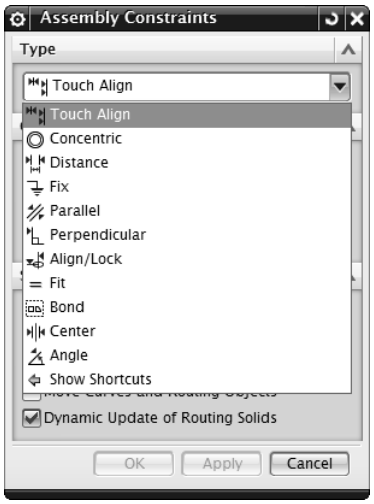


Figure 21: Siemens NX Assembly Constraints menu

There are eleven types of constraints in Siemens NX (Figure 21). Very often they are combined to achieve expected component behavior inside the assembly. It is important to note that constraints create relations between two or more components and it is not possible to create constraint based on only one component. Constraint features take points, edges, axes, faces, datum planes or solid bodies as references and every constraint is defined in its own specific way. Some can take in an account all mentioned references and some are based on only one or few possible references.

Touch constraint is the first on the list of constraints. Two references from two different objects need to be selected. It accepts all types of references and this constraint alone does not fix the component in place but allows linear and rotational movement until more constraints are added. Free linear and rotational movement depends on reference type used in creating the constraint. Some of the examples are shown in Figure 22. One can notice that selected references must touch in one common plane or axis.

First three illustrations on the left in Figure 22 represent touch constraint based on face references. After the constraint is defined, components are only allowed to slide with their referenced faces on the common touch plane whereas roller on the far right is allowed to rotate as long as its face touches common touch plane. Roller is allowed to slide also.

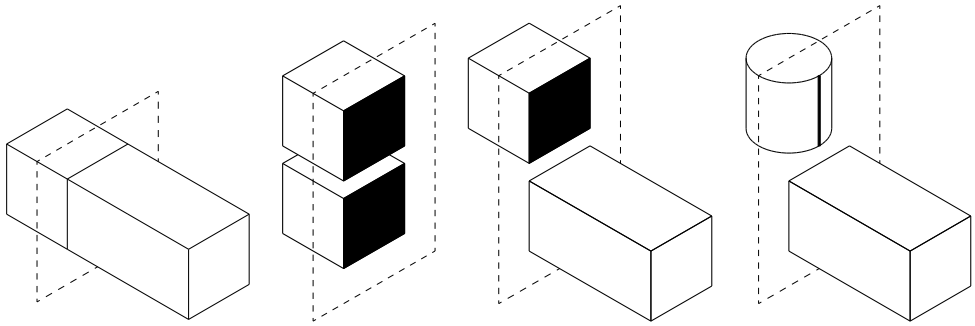


Figure 22: Touch constraint examples

Concentric constraints are uses mainly with rounded objects. In Figure 23, circle edge is selected as a reference on both components. Components are merged together in a way that both circles end up in the common plane created as a reference to one of the circles and the center of circles always move to the same spot. Component that hasn't been constrained and is floating freely in the assembly space is always the one that will move towards another component that is fixed (Figure 25) or in some other way constrained to the rest of assembly. In contrast to fit constraint example (Figure 28), components can only rotate around the center of the circle and are not allowed to move in linear motion parallel to circle center axis (up and down in this case).

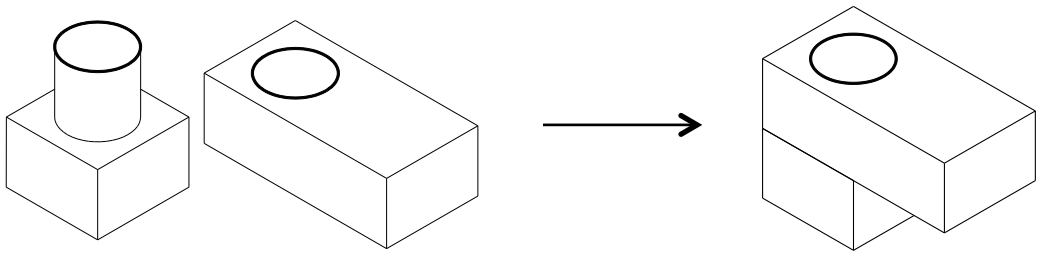


Figure 23: Concentric constraint example

Distance constraint defines fixed distance between referenced objects (Figure 24). Even though components don't touch in given examples, it's important for further research to know that distance constraint can be considered as a relation between two components because it's one of theirs mutual attributes. Once distance constraint is defined, components can move the same way as in case of touch constraint with the only difference of having fixed distance between them.

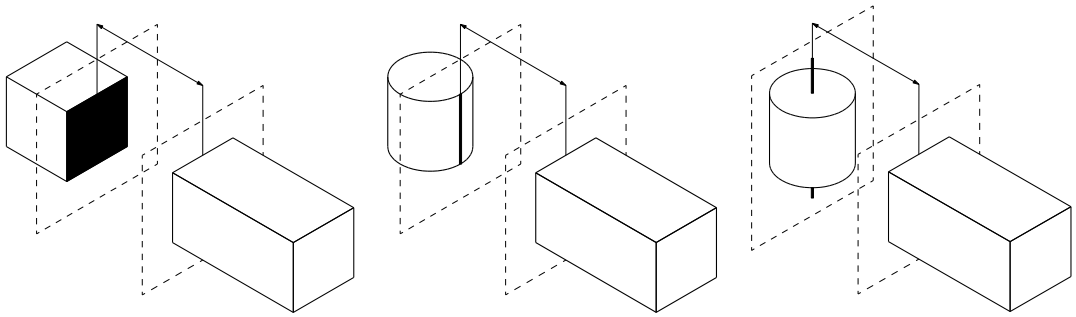


Figure 24: Distance constraint examples

Fix constraint is used when component shouldn't move at all (Figure 25). First component inserted into assembly is often fixed because it serves as a foundation for the rest of the assembly. Any component can be fixed at any place in the assembly and it's the only constraint that requires only one reference body. Because there are no two references that are required for constrained definition, this constraint doesn't create relation between to objects. This is important for further method development which will deal with extracting relations between components because it's obvious that this constraint can be excluded from analysis.

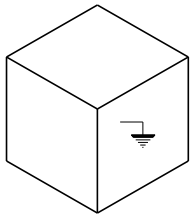


Figure 25: Fix constraint example

Parallel constraint is similar compared to distance constraint (Figure 24) with the exception that there is no fixed distance between referenced components. Selected referenced types are parallel. Please refer to paragraph describing distance constraint for more information.

Perpendicular constraint defines 90° angle relation (Figure 26) between two references from two different components. Reference types that are supported by this constraint are axes, edges, faces and datum planes. Constrained components are able to move as long as the perpendicular condition of two referenced types is met.

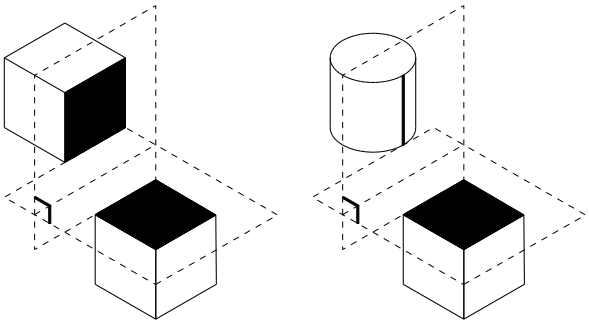


Figure 26: Perpendicular constraint example

Lock constraints are only able to use edges and axes as a reference types for defining the relation (Figure 27) between two components. Reference types are aligned parallel to each other without any distance between them. Linear and rotational movements are allowed where linear means sliding along the referenced edge or axis and rotational means that component can rotate around the referenced edge or axis.

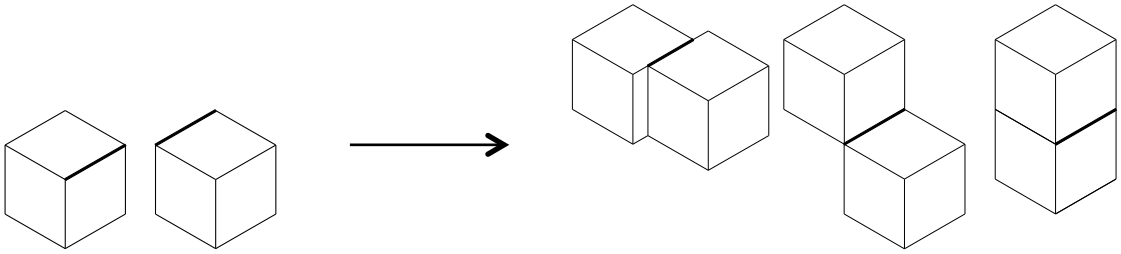


Figure 27: Align/Lock constraint example

Fit constraint (Figure 28) is similar to concentric constraint but uses faces as additional geometric reference type. It doesn't fix two components in place as concentric constraint does when circle edge reference is used (Figure 23). Rotational and linear movements are allowed relative to center of face or circle edge.

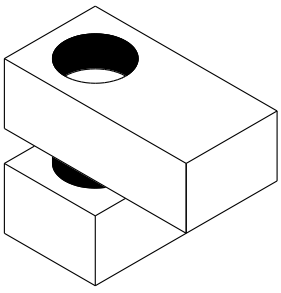


Figure 28: Fit constraint example

Bond constraint is similar to fix constrain (Figure 25) with the exception of being able to select multiple bodies which will define group of components that can't move. Two or more components have to be selected. Fix constraint is used if only one component needs to be fixed.

Center constraint is the only one defined with three or four referenced objects. Figure 29 demonstrates '1 to 2' definition model for defining the constraint. In this example, black colored face is selected first and after that two grey faces. Center constraint defines equal distance between all selected objects by placing first selected one in the middle. Two other ways of defining constrains are '2 to 1' and '2 to 2'. Process of defining the constraint is similar to the one described at the beginning of this paragraph with the difference in order and number of selected objects.

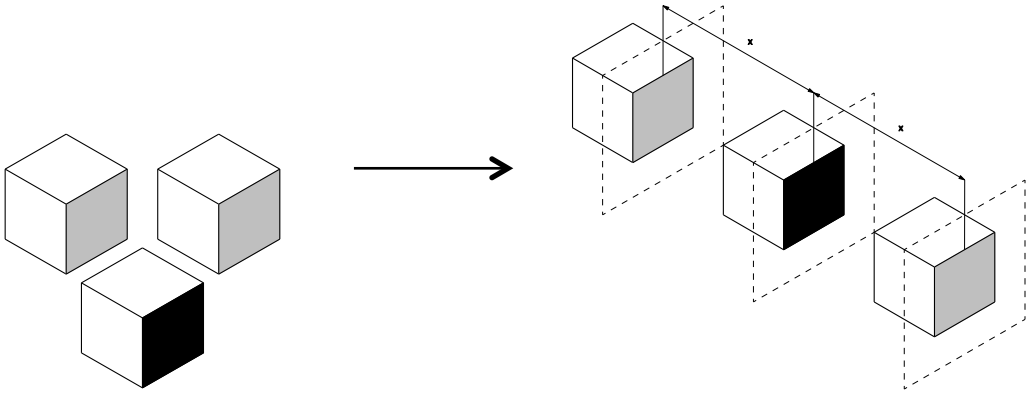


Figure 29: Center (1 to 2) constraint example

In contrast to perpendicular constraint (Figure 26) which has angle fixed at 90°, angle constraint (Figure 30) offers arbitrary angle value. For more details, please refer to paragraph that describes perpendicular constraint.

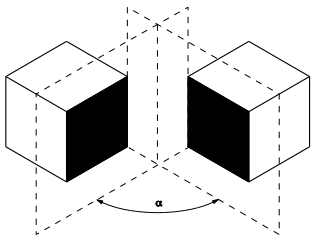


Figure 30: Angle constraint example

Those types of constrains and their names are specific for Siemens NX but other CAD (Computer Aided Design) software have similar, if not the same constraint types and names. All of them define basic geometric relations between components of the assembly.

5.2. Proximity

Proximity method for extracting component relations will consist of basic methods which take in the account distance between the components. No relation between components has to be defined prior to applying these methods.

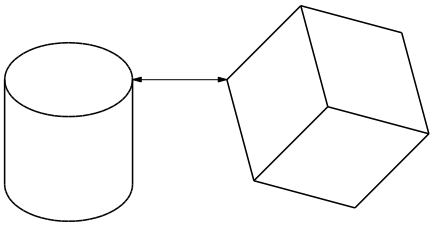


Figure 31: Non-constrained minimum distance

Minimum distance method has nothing to do with distance constraint method (Figure 24). It simply takes in the account two components and calculates minimum distance between them (Figure 31). Distance is calculated from the component surface and orientation doesn't matter. It is obvious that the user will have to choose the threshold which will then be used to determine if the relation between two components is valid.

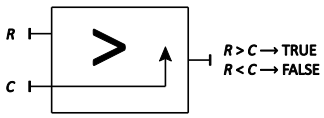


Figure 32: Comparator

Figure 32 describes threshold distance method logic. User chooses referenced value (*R*) which is then put against compared value (*C*). Result is TRUE or FALSE depending on the comparator method which is 'larger than' (>) in this case. TRUE comes as a result if referenced value is larger than compared value and it means that the relation for chosen distance is valid. FALSE comes as a result if compared distance is larger than referenced one and it means that the relation shouldn't be taken into account because the components are too far apart.

Box method derives from the Siemens NX feature that is able to create a virtual box around any type of geometric shape. In Siemens NX, this feature is referred to as Box Zone or Proximity Zone.

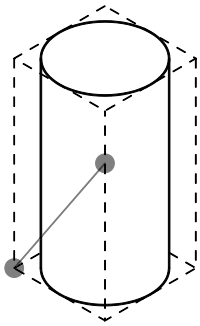


Figure 33: Virtual box definition

Virtual box is defined by two points: Component center point and the furthest point that will act as a box corner (Figure 33). Selected component will always fall inside the virtual box and the size of the box can only grow from there. By varying virtual box size, box method can detect if other components are within this virtual box volume (Figure 34). There are two conditions that can be used for validation if the relation between two components exists: virtual box intersect with only a part of other component or virtual box completely surrounds other component. In further development, first condition will be used as it will be important to get as many threats that can alert the engineer on possible collisions when the parts are moved or if their size changes.

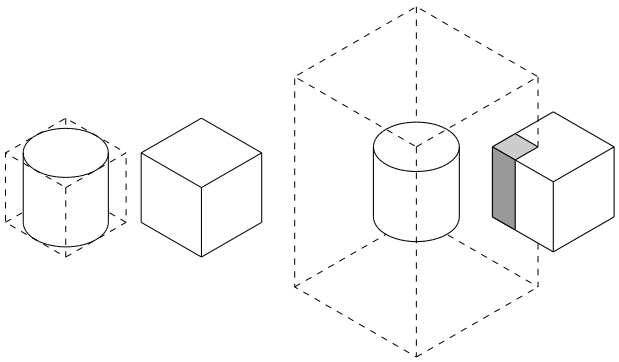


Figure 34: Box method

Left illustration in Figure 34 shows a non-valid component relation because virtual box is not touching other component. Right picture illustrates that for the different box size, there is a relation between two parts because virtual box intersects with part of other component. Same as with non-constrained minimum distance (Figure 31), user should define which box size is used for validation of relation.

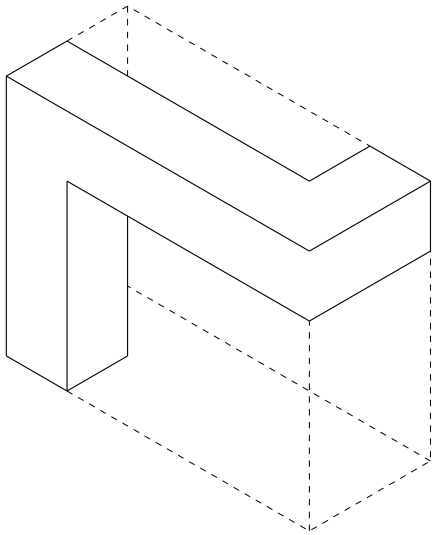


Figure 35: Box method problem

Because of the way the virtual box is created, there is a potential problem of getting non-important relations while using this method. Figure 35 illustrates the large space captured by the virtual box, especially in lower right corner. The question is if the component that will potentially sit there has enough important relation with the component from which the virtual box is created from because it is very far from it? Figure 36 suggests one of the potential solutions for getting rid of those non-important relations.

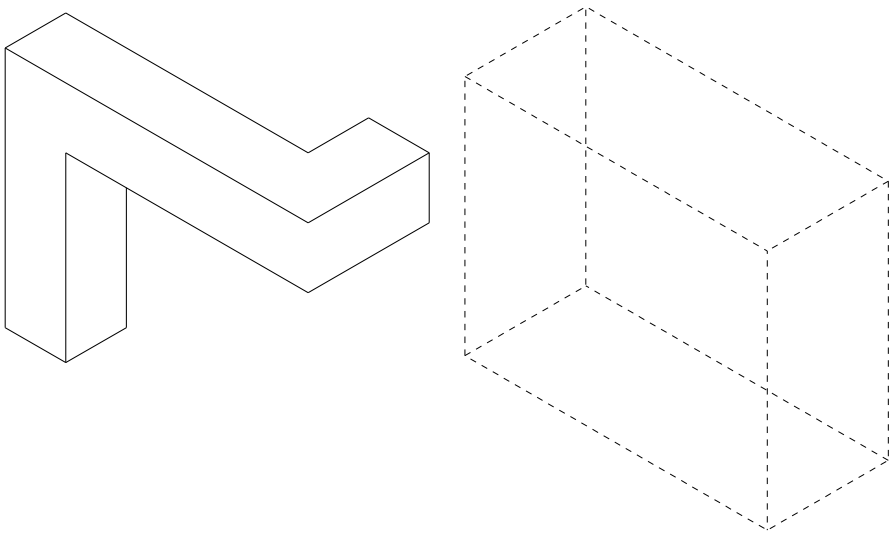


Figure 36: Low body-box ratio – all relations to selected component from other components within the box should be excluded

Left illustration in Figure 36 represents the volume of the selected component. Right illustration represents the volume of the virtual box for this component. Both illustrations are created from simple cube boxes, therefore number of boxes that created left illustration is 9 and 20 for the right illustration. Ratio between those two volumes is 0.45. This value can then be used to compare with referenced value (Figure 32) to validate if the relation should be taken into consideration. Idea is to completely discard any relation produced by this method if the ratio is lower than the given reference value.

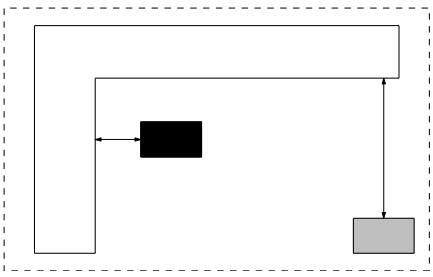


Figure 37: Box method together with distance method

Figure 37 illustrates what will happen when virtual box around selected component surrounds two other small components but the volume ratio between selected component and the virtual

box is low. Taking only box method into consideration, both components that fall into the virtual box would be excluded from extracting the relation because of the small volume ratio. Minimum distance method (Figure 31) will still include the relation of one small component (Figure 37, black) even though it is discarded with box method. If we assume that the minimum distance value to reach the other small component (Figure 37, grey) is larger than the referenced value (Figure 32), then the validation of the relations from sample model in Figure 37 is acceptable.

5.3. Permanent joints

Permanent joints will be represented by welds. Any type of weld is sure to connect at least two components and this is what qualifies it as a relation reference.

Daimler's design training literature (Table 2) suggests many weld types but they all have the same purpose of connecting components, therefore it is irrelevant to distinguish them at this point. As it was stated in previous paragraph, every weld is a valid relation between two or more components. Attributes describing the weld will later be used to extract valuable meta-data for describing the relation itself, therefore distinguishing them.

Table 2: Daimler's suggested weld types

ID	Description	ID	Description
Weld			
21	Spot welding	211	Spot welding (VAN/TRUCK); Indirect spot welding (CAR)
212	Spot welding (VAN/TRUCK); Direct spot welding (CAR)	23	Projection welding
231	Indirect projection welding	232	Direct projection welding
24	Flash welding	25	Upset welding
4	Welding with pressure	42	Friction welding
H611	Laser-MIG hybrid welding	H612	Laser-MIG hybrid welding
75	Laser welding	78	Stud welding
78	Stud welding (VAN/TRUCK) A0009847719 M8x13	78	Stud welding A0019840819 M6x16 Threaded bolt with painted groove
78	Stud welding A0009902913 T5x14.2 Xmas tree stud	78	Stud welding A0009910103 M6 Grounding stud

78	Stud welding A0009914903 M4x11.1	23	Resistance stud welding
783	Drawn arc stud welding	784	Short-cycle drawn arc stud welding
785	Capacitor discharge drawn arc stud welding	423	Friction stud welding
Mechanic			
52LN	Laser knob	52PN	Punching operation knob
52MN	Mould knob	C1	Clinching
C110	Clinching without cutting	C140	Clinching with cutting
C170	Clinching with prehole	D110	Flow drill screwing with prehole
D140	Flow drill screwing without prehole	E110	ImpAcT (impulse-type linear driving)
S2	Self-piercing rivets (all materials)	S210	Self-piercing semitubular rivet
S240	Self-piercing full rivet	N3	Rivets (all materials)
N310	Blind rivet (all materials)	N360	Riveting with tubular rivet
N380	Riveting with huck bolt		
Robscan			
L524	Robscan Remote welding		

Problems can arise from welds that are partitioned, but belong to the same group. For example, spot welds have many spots that connect two or more components and the problem is that every spot will be recognized as one relation. Those relations would be duplicates. One of the solutions to this problem is to gather all the relations and in post-processing methods combine the spots in one group, therefore one relation. One relation is logically the correct answer even though one weld has many partitioned welds.

5.4. Non-permanent joints

Non-permanent joints should represent screws and remote forces that cause interaction between two or more components. In the very beginning, this thesis eliminates the possibility of remote force existence such as magnetic field because during my research I did not stumble upon such or similar feature in Siemens NX assemblies. Therefore, screws are most common non-permanent type of connection that needs to be addressed.

There is one main assumption that guarantees extraction of screw relation: Screw is touching or is very close to components that are connected by the screw. Based on this claim, minimum distance (Figure 31) and box (Figure 34) method ensure extraction of relation. For example,

component 11 is very close or touching components 8 and 9 in Figure 38. Therefore there is relation between components 8 and 9 through 11. It confirms that basic principles of previous methods applied on non-permanent relations considered in this thesis are sufficient to extract important relations. No other method needs to be developed. In future, when there will be assemblies containing magnetic fields or similar remote forces, new methods for non-permanent relations will need to be developed to find related relations.

5.5. Validation model for developed methods

Validation model is simplified real life representation of an assembly. Assembly components are arranged in a way that is challenging for prototype relation extraction algorithm to recognize all the important relations.

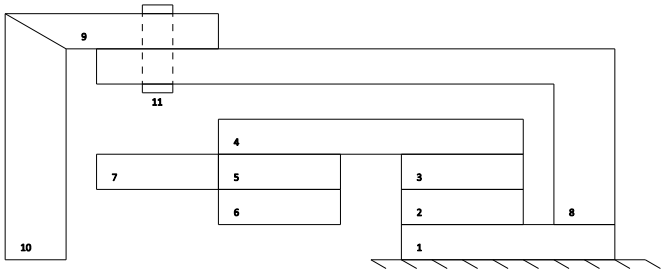


Figure 38: Validation model

Figure 39 is manually created DSM matrix based on model illustrated in Figure 38. It is important to note that the amount of time is significant for describing even the simple model like this. If complex technical systems are taken into account, it becomes almost impossible and completely inefficient to create DSMs manually. Again, this confirms the need for automatic relation extraction approach. Such approach is beneficial in saving man power, working hours and in many other activities related to the process of manually creating DSMs. Some may consider using DSMs even if it seemed impossible to do so before.

Aggregated DSM from Figure 39 serves as a reference for validation of results that the prototype algorithm will produce. Grey cells are most important thing to focus on. It is only extra beneficial if grey cell contains multiple types of relations that are recognized, but it is enough to detect even one type of relation to be able to tell that there is a relation between two components. Results will be compared by overlapping DSM from Figure 39 and the one created by prototype. Ideally, grey areas will match.

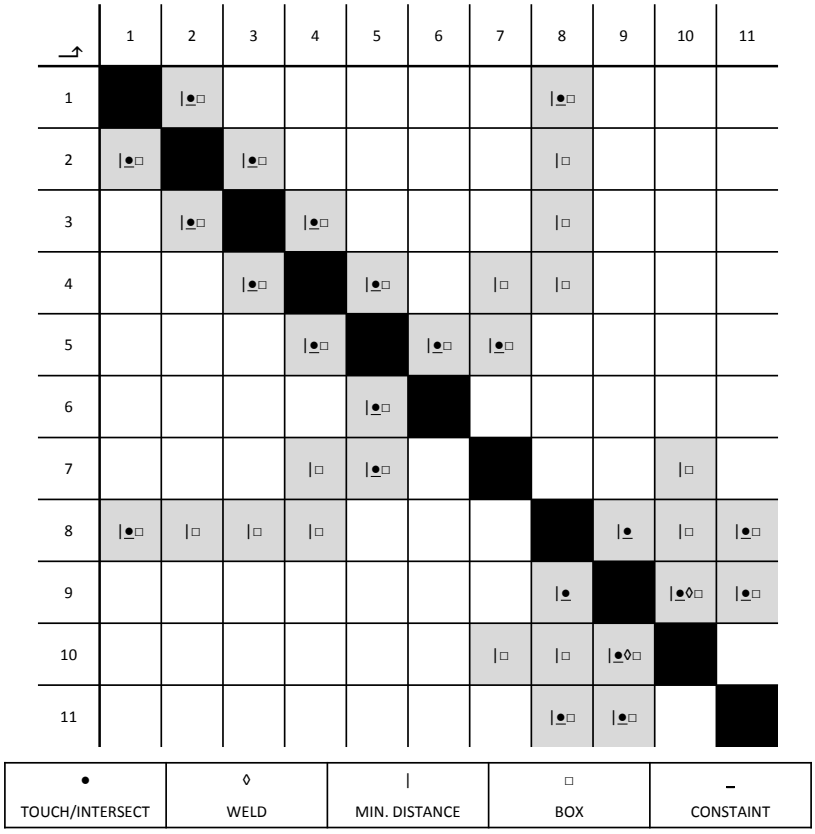


Figure 39: Manually created aggregated DSM based on model from Figure 38

6. PoC (Proof of Concept)

Task of prototyping software solutions firstly requires the selection of the right tool that suits the needs the prototype has to address. Ideally, selected IDE (Integrated Development Environment) allows intuitive, simple, fast, expandable and powerful enough environment. Since this thesis is dealing with specific topic and therefore used tools, this is also what needs to be taken into consideration.

Siemens NX provides NX Open API (Application Programming Interface) to interact with data produced in NX environment. NX Open API's documentation discloses possible programing languages that can be used for interacting with the API. Those are: C/C++, Java, .NET, GRIP and CAE. GRIP and CAE are the programing languages that support manufacturing and simulation and therefore can be ignored for the use case this thesis is dealing with. It leaves me with C/C++, Java and .NET to choose from. Considering my background and interest in web technologies and therefore inevitably coming across JavaScript programing language, it is clear that I should choose Java. I will leave the discussion about advantages and disadvantages of every programing language aside.

Eclipse IDE is free, supports Java and is packed with necessary features to build a prototype. It is also powerful enough for production stage of software development if needed. Specifically, Eclipse IDE for Java EE (Enterprise Edition) Developers is used. Basic MVC (Model-View-Controller) software architecture approach will be applied for this PoC.

Next few paragraphs will describe the way Eclipse IDE was set in combination with Siemens NX to enable feature like debugging, brief introduction to NX Open API and following documentation, brief overview of Siemens NX Journals while explaining why are they useful and finally, description of implemented methods mentioned in paragraph **5**.

This thesis will not go in detail explaining programming logic and practices. It provides an overview of used tools, documents important steps taken to make a prototype solution and describes algorithms in plain language. Application source is included in additional prints coupled with this thesis.

6.1. Eclipse IDE

Eclipse offers pre-defined packages based on the type of development in place. Luna release Eclipse IDE for Java EE Developers package is the one used for the purpose of this thesis.

There are many different releases but to make sure that important libraries and features are included, Luna is chosen.

Figure 40 illustrates basic layout of Eclipse. List on the far left side is referred to as Package explorer. Essentially, it shows all files related to the project and serves as a navigator through these files. Large middle section is used for code editing. Code syntax is colored and therefore it is easier to find the point of interest. Bottom of Eclipse layout holds multi-tab section which includes Javadoc (documentation references to code methods, types and classes), Problems (Eclipse-identified possible problems) and Console (simple input-output canvas) to name a few. Information provided from those tabs helps developers during project development process. Two far right lists help developers to identify created methods, functions and variables together with the possibility to create task list in process of managing the project. Top toolbar provides shortcuts to most used functions. Java and Debug buttons on the right are worth mentioning because they switch the layout depending on what is done with the application. Figure 40 shows Java layout which is explained in this paragraph but Debug layout is always used while debugging the project. Debugging layout emphasizes variables and their content together with new functions in toolbar that allow the developer to go through the code step-by-step in order to provide better overview of what is happening inside application as it runs.

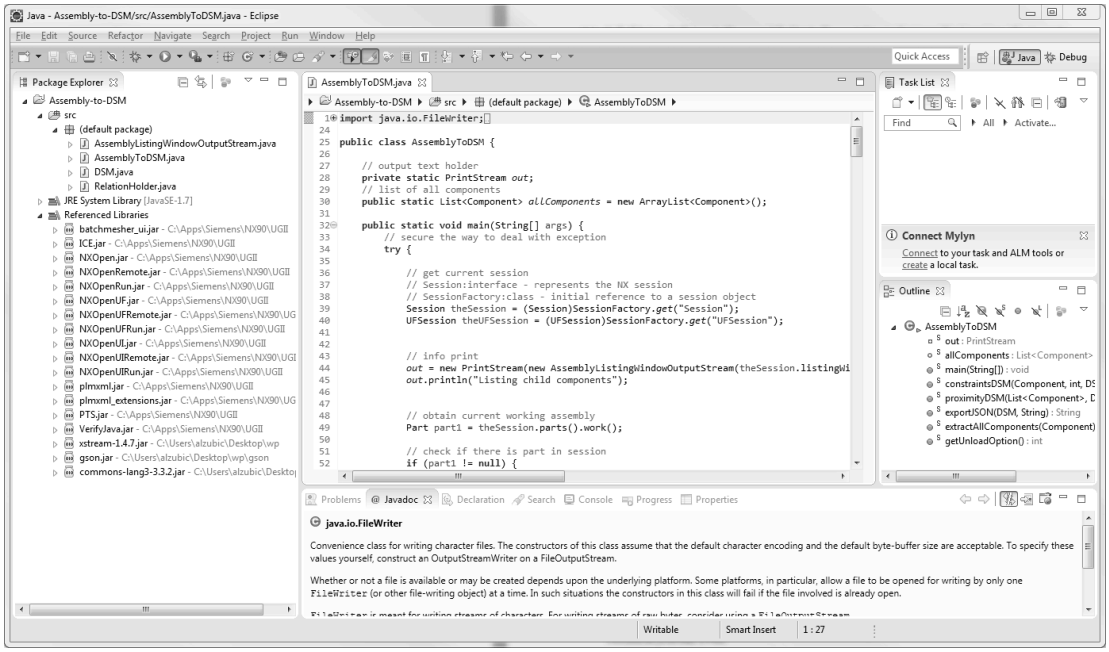


Figure 40: Luna release Eclipse IDE for Java EE Developers

Eclipse Marketplace is module inside Eclipse that allows developers to search for new libraries and install them if needed. To build a prototype for this thesis, Eclipse Marketplace was not used.

6.1.1. NX Open Java API

NX Open API is an interface through which Siemens NX can be manipulated and managed with code. Figure 41 illustrates how the NX Open Java API enables Java code to interact with Siemens NX. Direct connection between Java application and Siemens NX is not possible.

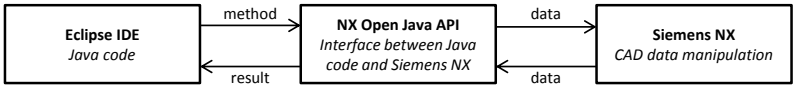


Figure 41: Abstract of NX Open Java API role

NX Open Java API is essentially a container that holds Siemens NX specific methods, functions and types which can be used in the programing language of choice, this time in Java. Since the mentioned API library is compiled, source cannot be seen. Therefore Siemens created a documentation explaining all existing methods, functions and types held inside NX Open API.

6.1.2. NX Open documentation

As previous paragraph stated, NX open API has to have a documentation explaining the APIs. Without it, it would be hard (if even possible) to use the API.

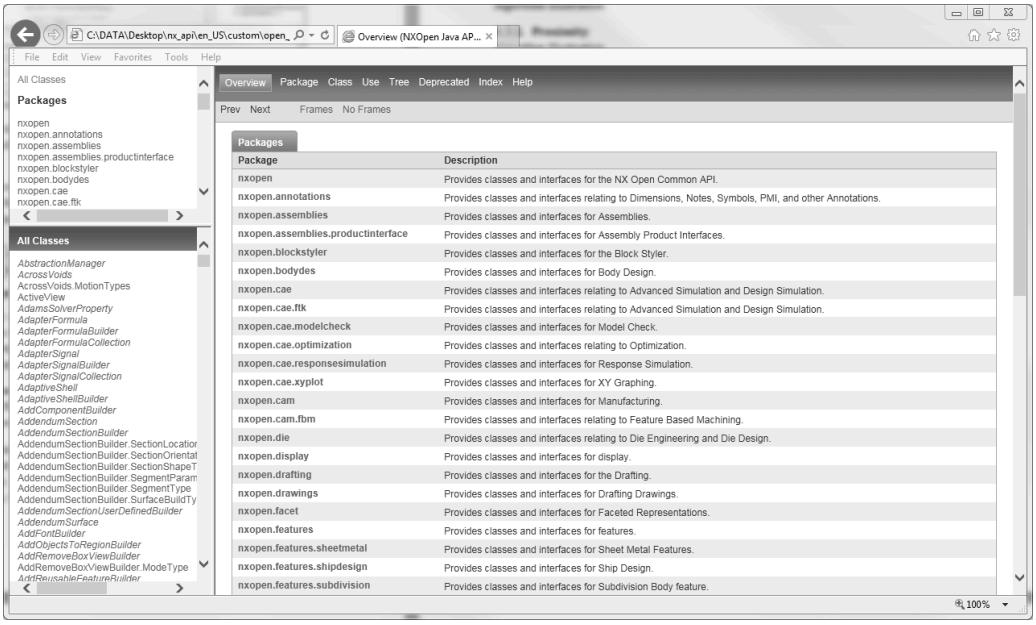


Figure 42: NX Open Java API documentation

NX Open Java API documentation (Figure 42) is standardized in regard to Java practices. Basic structure of documentation consists of a list of packages, classes and interfaces inside those packages and methods together with types inside selected class or interface. Methods contain brief explanation about what they do, which parameters are taken in account and what are the results of certain method.

It is obvious that detailed API documentation is necessary to understand the library. Unfortunately, NX Open APIs are not that well documented. Mastering methods to understand what they do and what the necessary inputs are comes with the experience. Available descriptions are often not enough to understand the method at first. There is also the lack of examples. Siemens GTAC Solution Center and PLM Community forums can provide valuable solutions for some of the practical problems.

6.1.3. Debugging and journals

Debugging plays important role in application development. It allows the developer to run the application step-by-step through the code. Variables used by the application can be monitored at the same time. It is important because it validates that the code manipulates them as it should.

Remote debugging has to be enabled to debug Siemens NX application inside Eclipse. Following steps have to be taken:

1. Step: Create Remote Java Application
    a. Open project,
    b. Menu: 'Run' – 'Debug Configurations…',
    c. Select 'Remote Java Application' and click 'New launch configuration' button from the upper menu. New Remote Java Application item is created,
    d. Select new item and change 'Connection Type' to 'Standard (Socket Listen)',
    e. Remember 'Port' number from 'Connection Properties',
    f. Click 'Debug' button at dialog bottom to confirm configuration.

Those steps ensure that Eclipse is ready to receive debugging data. It is important to note that debugging has to be started every time project is reopened. It is not necessary to start remote debugging again after every application test run.

2. Step: Set Siemens NX Java parameters
    a. Ensure fresh start of Siemens NX,

b. Menu: ‘File’ – ‘Execute’ – ‘Override Java Parameters…’,
c. Copy line from Figure 43 into ‘UGII_JVM_OPTIONS’,
d. Confirm changes with ‘OK’ button.

```
-Xdebug -Xrunjdwp:transport=dt_socket,address=127.0.0.1:8000,suspend=y
```

Figure 43: Siemens NX Java parameter to enable debugging

Now every time application is run from Siemens NX (Menu: ‘File’ – ‘Execute’ – ‘NX Open…’), if there is a breaking point in the code, Eclipse will switch to debugging layout.

Additional parameters have to be set in 2. Step – b. part for this prototype to work properly. It’s not related to debugging issue. Please copy line from Figure 44 into ‘UGII_CLASSPATH’. *path_to_gson* and *path_to_lang* have to be modified to match file path on current computer. GSON is a Java library for converting JSON to Java objects and vice-versa and Apache Commons Lang is Apache’s library that supports additional methods for array manipulation.

```
*path_to_gson*\gson.jar;*path_to_lang*\commons-lang.jar
```

Figure 44: Additional Siemens NX Java parameter

Issue form previous paragraph should be resolved in the future and therefore it won’t be necessary to take that step.

Siemens NX Journal is a useful tool for extracting relevant code to achieve certain actions. Prior to diving into NX Open API documentation and searching for relevant methods, it is wise to record a journal. It contains Java code from events that happened in the process of recording. Not all of the events are recorded but often times Journals can help steer the developer in the right direction.

6.2. MVC (Model-View-Controller)

MVC is well known software architecture concept (not rules) used for creating modern system structures. Basic idea is to have specific modules that are independent and have specific tasks. Therefore, it is easier to plan the development, manage all the resources and update the system with new features. MVC separates application in three main modules: Model, View and Controller.

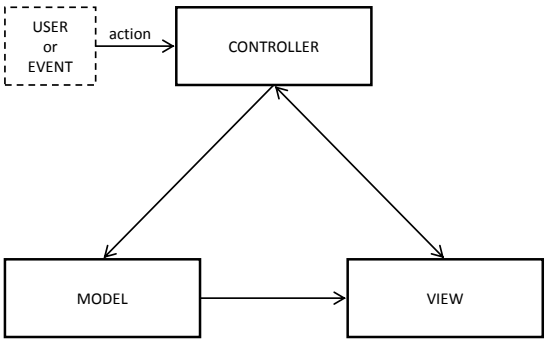


Figure 45: MVC concept

Model stores and manipulates state of data within the system. It can be referred to as a 'skeleton' of the system. Data to the Model is either passed by the user or it is pulled from existing databases. Output of the Model is s subset of data and sometimes instructions recognizable by the View module.

View can be referred to as a 'skin' for the subset of data and instruction from the Model or the user. It is visual representation of action done by the user and the Model. View module displays the data and therefore has to know the semantics and layout of presented data from other modules in MVC architecture.

Controller takes the user input and controls the interaction between Model and View modules. It can be referred to as a 'brain' of the system because user input changes Model which means View also. User requests are routed through the system by the handlers which are build to coordinate specific user actions.

It is obvious that the user is also part of the MVC concept but it is never discussed specifically because user only commands the system. System structure is therefore of grater importance to successfully perform tasks set by the user.

Following Assembly-to-DSM files can be fitted into MVC concept like this:

- Model – DSM.java, RelationHolder.java, DSM output JSON files
- View – JSONtoDSM.html
- Controller – AssemblyToDSM.java, AssemblyListingWindowOutputStream.java

6.3. Assembly-to-DSM PoC

Assembly-to-DSM is the name of prototype project. Java code contains algorithms that will extract important relations from the assembly based on the methods described in Conceptual

development of methods for extracting product assembly relations paragraph. Methods through which code interacts with Siemens NX are learned from NX Open API Java documentation. Resulting data is visualized using JavaScript d3.js library to show the results as a webpage.

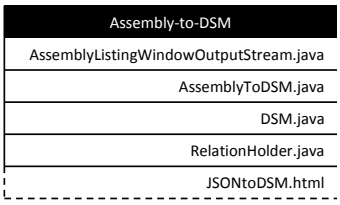


Figure 46: Assembly-to-DSM project structure

Figure 46 shows basic project class structure. In addition, JSONtoDSM.html is a HTML file that is not part of Eclipse project. It uses produced results put in *.json files to plot resulting DSM. Here is the summary of the roles of each file:

- AssemblyListingWindowOutputStream.java – serves as an interface through which any data can be outputted as a string. Outputted information in this project informs user about the status of application execution process,
- AssemblyToDSM.java – contains `main()` method, therefore it runs first and defines the execution steps of the application. It contains methods for extracting component relations,
- DSM.java – defines how the final object describing DSM relations. This object is exported as a final result of the application,
- RelationHolder.java – when application finds a relation between two or more components, RelationHolder class serves as an object that describes this relation. After relation is fully described based on the data extracted from Siemens NX, it is then put in related instance of DSM class that contains all the relations,
- JSONtoDSM.html – Results gained from Java algorithm are exported to JSON file format. Created files are then loaded into web interface which uses d3.js visualization library to display data as DSM.

Table 3 describes `main()` method from AssemblyToDSM.java.

Table 3: Intro algorithm

```
get current Session
get current Part from Session
if Part
·       create constraintRelations
```

```
·       create proximityRelations
·       create permanentJointRelations
·       create nonpermanentJointRelations
·       extractConstraintRelations(Part)
·       extractProximityRelations(Part)
·       extractPermanentJointRelations(Part)
·       extractNonpermanentJointRelations(Part)
end if
export *Relations objects into JSON files
```

After `extract*` methods are called, `*Relations` objects will contain all extracted relations. It is worth mentioning that prototype code uses Google's GSON library to export `*Relations` objects into JSON files as a final output. It is expected that exported files then serve as a base for creating DSMs.

Next few paragraphs describe generic algorithms for extracting relations and do not contain exact NX Open API method names. Exact method names and related attributes that are necessary for them to work can be found in additional prints coupled with this thesis.

6.3.1. Constraints

Constraints are explicitly defined in the assembly. Algorithm described in Table 4 has to go through all the components and check if there is a constraint related to any of those components. If there is any, data is extracted from this constraint and passed to the collection that holds all the relations from this extraction method.

Table 4: Constraints algorithm

```
for i = 1 → total # of components
·       current component constraints
·       for j = 0 → total # of constraints
·       ·       if constraint[j]
·       ·       ·       currentConstraint
·       ·       ·       ·       += component file path
·       ·       ·       ·       += component tag
·       ·       ·       ·       += relation summary
·       ·       ·       ·       += relation type
·       ·       ·       ·       += constraint alignment
·       ·       ·       ·       += # of related components
·       ·       ·       ·       += constraint reference type
·       ·       ·       ·       += constraint related components
·       ·       ·       ·       += constraint tag
·       ·       ·       add currentConstraint to constraintRelations
·       ·       end if
·       next j
next i
```

6.3.2. Proximity

In contrast to constraints which are explicitly defined in the assembly, proximity finds relations based on the methods from paragraph **5.2** which are then compared to the threshold that is set by the user. Threshold filters important relations from those which are not. It is obvious that the number of extracted relations depends on how strict are the user parameters.

`boxVolume` and `minDistanceTreshold` are two variables that are set by the user. `boxVolume` defines distance between corner box coordinate and center point coordinate as shown in Figure 33. Default component virtual box will grow. `minDistanceThreshold` is compared with minimum distance between two components. If the `minDistanceThreshold` is larger than calculated minimum distance between components, relation is considered important.

`volumeRatio` is a parameter calculated as described in Figure 36 and can be additionally used for excluding non-important relations. It is ratio between component and box volume.

Table 5: Proximity algorithm

```
for i = 1 → total # of components
·       // box
·       for j = 1 → total # of components
·       ·       create boxSize
·       ·       create inBox
·       ·       if inBox && component[i] != component[j] && !assembly
·       ·       ·       create boxVolume
·       ·       ·       create componentVolume
·       ·       ·       create volumeRatio =  componentVolume / boxVolume
·       ·       ·       currentBoxRelation
·       ·       ·       ·       += component file path
·       ·       ·       ·       += component tag
·       ·       ·       ·       += relation summary
·       ·       ·       ·       += relation type
·       ·       ·       ·       += # of related components
·       ·       ·       ·       += constraint reference type
·       ·       ·       ·       += constraint related components
·       ·       ·       ·       += constraint tag
·       ·       ·       ·       += volumeRatio
·       ·       ·       add currentBoxRelation to proximityRelations
·       ·       end if
·       next j
·       // distance
·       for k = i → total # of components
·       ·       create minDistThreshold
·       ·       create minDistMeasured
·       ·       // threshold distance is smaller than minimum measured between i and k component
·       ·       if minDistThreshold <= minDistMeasured && minDistanceThreshold != 0
·       ·       && component[i] != component[k] && !assembly
·       ·       ·       currentDistanceRelation
·       ·       ·       ·       += component file path
·       ·       ·       ·       += component tag
·       ·       ·       ·       += relation summary
·       ·       ·       ·       += relation type
·       ·       ·       ·       += # of related components
·       ·       ·       ·       += proximity reference type
·       ·       ·       ·       += proximity related components
·       ·       ·       ·       += proximity tag
·       ·       ·       add currentDistanceRelation to proximityRelations
·       ·       end if
·       ·       // components touch or intrude one another
·       ·       if minDistMeasured = 0 && component[i] != component[k] && !assembly
·       ·       ·       currentDistanceRelation
·       ·       ·       ·       += component file path
·       ·       ·       ·       += component tag
·       ·       ·       ·       += relation summary
·       ·       ·       ·       += relation type
·       ·       ·       ·       += # of related components
·       ·       ·       ·       += proximity reference type
·       ·       ·       ·       += proximity related components
```

```
·       ·       ·       ·       += proximity tag
·       ·       ·       add currentDistanceRelation to proximityRelations
·       ·       end if
·       next k
next i
```

6.3.3. Permanent joints

Permanent joints (welds) are explicitly defined features inside the assembly, therefore it is necessary to go through all the components and check if there is a feature with certain attribute that indicates the weld exists. This prototype checks for `Weld_Type` attribute inside the feature and if it exists, `isWeld` is set to `TRUE`. Additional info regarding the weld is extracted and put as meta-data in current relation holder object.

Table 6: Permanent joints algorithm

```
for i = 1 → total # of components
·       set current component as working part
·       extract all features from working part
·       for j = 0 → total # of features
·       ·       create isWeld
·       ·       if isWeld
·       ·       ·       currentPJoint
·       ·       ·       ·       += component file path
·       ·       ·       ·       += component tag
·       ·       ·       ·       += relation summary
·       ·       ·       ·       += relation type
·       ·       ·       ·       += # of related components
·       ·       ·       ·       += joint reference type
·       ·       ·       ·       += joint related components
·       ·       ·       ·       += procedure code
·       ·       ·       add currentPJoint to permanentJointRelations
·       ·       end if
·       next j
next i
```

6.3.4. Output format

RelationHolder.java is the class that holds all relevant data extracted by the algorithm. Objects within the given class are described as follows:

- summary – brief description of the relation. Contains name of the constraint, names of components welded together or names of components that are close to each other,
- type – describes type of the relation which is extracted from the CAD variables or custom name if the relation is not directly extracted from the CAD feature,
- path – exact path on hard drive to the file of selected component. Since relation is defined between two components or more, path is related to the component on top of which the relation is created,
- componentTag – component's unique ID. Siemens NX tags components with numbers,

- alignment – only exist in constrained type of relation. Describes the way components are aligned based on information extracted from available CAD variables,
- referenceType – custom created array stating the references based on which the relation is created (for example planes) or strings that describe the threshold parameters from the algorithm (for example 15mm BOX which means that the relation is extracted with proximity method building a 15mm long, deep and tall square box around the component described in section **5.2**),
- realName – name of the component displayed in CAD,
- referenceName – any additional identifications of the component's reference that is not the realName or componentTag,
- referenceTag – array containing component's relation reference ID. When constraint is defined, each component gets additional ID for the constraint. When proximity relation is extracted, array fields contain componentTag of each component,
- numberOfRelatedComponents – now it is always two, but for possible future improvements of the program this variable is already introduced,
- volumeRatio – calculated only in proximity algorithm. Detailed explanation is given in section **5.2**, Figure 35,
- procedureCode – relevant only for welded joints. Procedure code is the number that references to specific type of weld given in section **5.3**, Table 2. This is related to Daimler internal codes that are implemented in CAD metadata therefore not relevant to any other company.

Gson (official name is google-gson) Java library is later used to convert given objects from Java environment into JSON format so that JSONtoDSM.html web interface can use the data for visualization. Gson library is able to convert Java objects to JSON and vice-versa.

6.4. Result validation and visualization

As stated in section **5.5**, results of this thesis are validated based on designed model (Figure 38) which is then put through the algorithm developed using methods explained in section **5**. Figure 39 summarizes validation model into DSM matrix. DSM matrix is manually created by the author of this thesis and will serve as a reference and for comparison with computer generated results.

Design of the validation model (Figure 38) is simple, therefore time needed for algorithm to process it was short. For the reference, algorithm was put thought some more complex geometries available at the time spent in Daimler and it took few minutes to process (approximately 50 components under 3 minutes). Processing time primarily depends on the number of components and not on the complexity of the components involved. Larger quantity of components need more time to be processed.


```
…
{
            "summary": "block-8 catches box of block-2-7",
            "type": "INSIDE_BOX",
            "path": "C:\\Users\\alzubic\\Desktop\\tmp NX files\\validation\\block-8.prt",
            "componentTag": [39690],
            "referenceType": ["15.0mm BOX"],
            "realName": ["8","3"],
            "referenceName": ["block-8","block-2-7"],
            "referenceTag": [39690,39660],
            "numberOfRelatedComponents": 2,
            "volumeRatio": 0.2912846327236437
}
…
```


Figure 47: Slice of result data - JSON format

Figure 47 shows part of the final result produced by developed algorithm. Results are in JSON file format containing extracted information from product assembly. As shown in Figure 47, this slice of result data shows one of the relation recognized between component number 3 and 8 as labeled in Figure 38. In detail, this slice tells us that the component number 3 is inside the virtual box (Figure 37) created around component number 8. Also, volume ratio (Figure 35) of the component number 8 body volume in relation to virtual box volume around its body is calculated. It is obvious that the ratio is low and since it is suggested as a problem in section **5.2**, this particular relation is ignored in final DSM plot. All relations where volume ratio is below 0.5 are ignored.

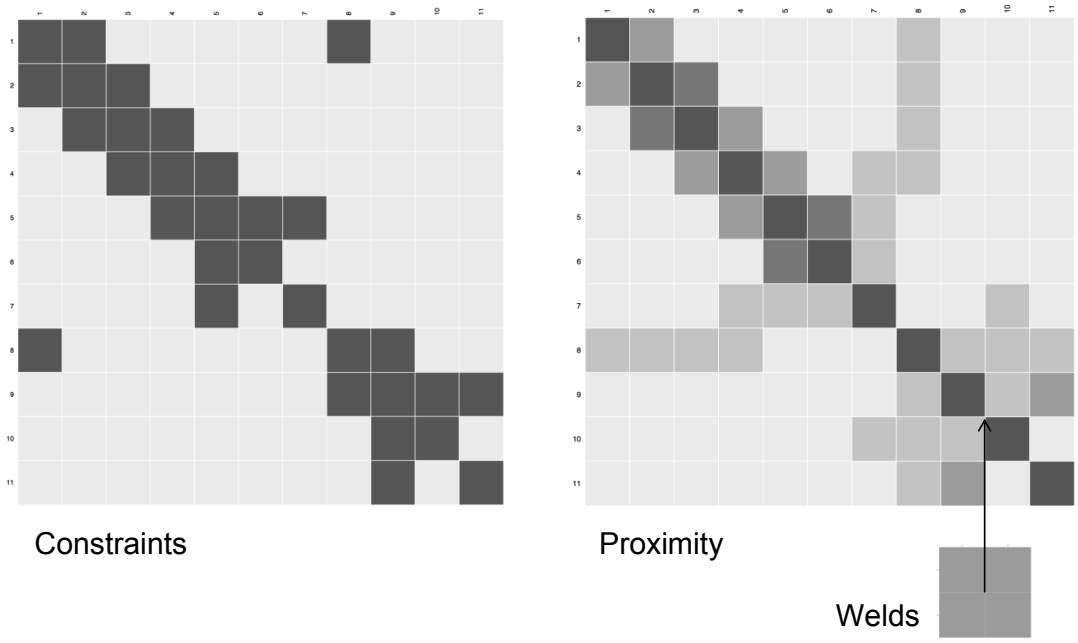


Figure 48: Visualized results - separated based on the extraction method

Final results are visualized using d3.js graph library with some help from JavaScript, JQuery, HTML and CSS. As seen in Figure 48, results are put in DSM style table where darker grey color represents existing relation between the components. Necessarily, there is not only one type of relation between two components, but for the validation purposes is determined that if there is at least one relation detected, the field is darker grey colored. All the other detected relations between the same two components are still in JSON files even though the number of them is not shown in the visual output.

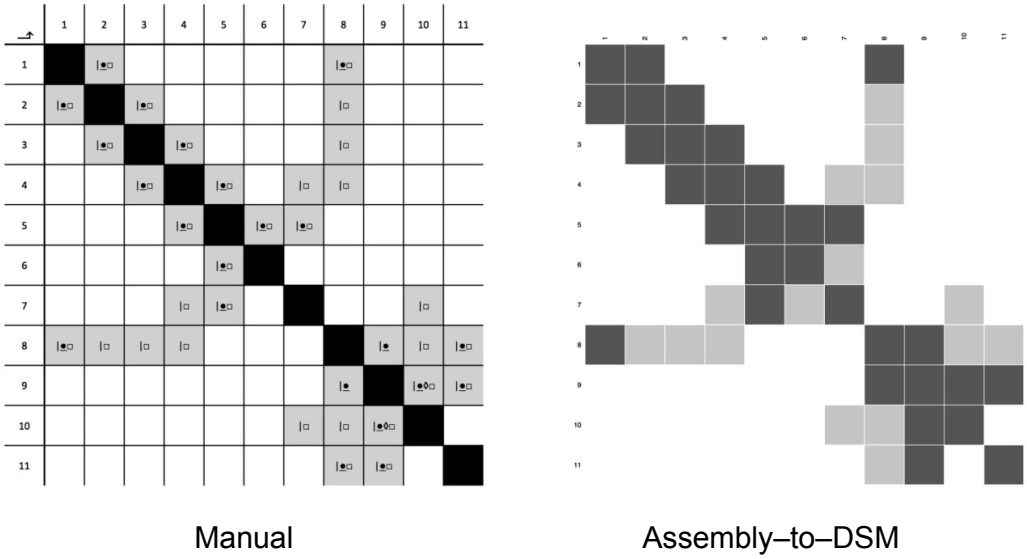


Figure 49: Aggregated DSMs

Figure 49 presents the aggregated view of the separate results shown in Figure 48 on the right together with manually created DSM (Figure 39) on the left. Those are two DSMs which need to be compared to validate the results produced by the algorithm. Rows and columns contain numbers from 1 to 11 that represents the number of the components labeled in Figure 38.

When Manual and Assembly-to-DSM matrices (Figure 49) are overlapped, it can be stated that developed algorithms are well designed. Algorithms extract proper relations from given CAD model and results are successfully validated in comparison with manually created DSM.

There is one important lesson to be learned here and this is that machines never miss their goal if well tuned. Proof to support this claim comes from Figure 49. Take a closer look at coordinates 6-7 and 7-6! This is symmetric DSM and therefore both coordinates indicate the same relation between components 6 and 7 (proximity), but you can notice that this relation was not recognized in manually created DSM! This is not done on purpose and since result is valid, credit goes to developed algorithms. Statistical probability of human error in recognizing relevant component relations rises with the product complexity whereas machine needs less time and is much more precise.

7. Discussion

Please note that developed methods are applicable on an existing assembly while creating the product or after the product is finished. It means that it compliments traceability with new data after the product is completed or during iterative design process. Methods are especially helpful in case of creating new version of existing product because of an existing overview of system architecture.

Daimler has certain design rules for creating CAD assemblies. With those rules in place, standardization of process is achieved. While writing this thesis, it is noticed that there is some room for improvement. If Daimler adopted following design rule for creating assemblies, other methods (not just clustering, Figure 17) would be applicable: while creating constraints between components, first select the one on which the next component depends on. Also, start from the root component - component from which the real-life assembly starts. It primarily means that sequencing methods [16] could be helpful in creating step-by-step component mounting diagrams because the order of assembling could be extracted. If Daimler doesn't implement constraints in its design process, it is virtually impossible to create base ground for sequencing methods without user input. Currently, component assembly sequence is identified by the user. Decomposing that kind of an assembly with the suggested rule applied in the process, it would give non-symmetric DSM matrix which could be used to remove 'PowerPoint Engineering'[2].

Methods for visualization other than DSMs are welcome to be considered. For example, graph theory diagrams which are mentioned in section **4.3**. Those diagrams contain simple circles as nodes and lines representing relations which are connecting them. Depending on the application and the needs of engineering team, different type of visualization can be beneficial for the specific application. Mentioned graph theory diagrams are specially beneficial if the components which are the 'hub of many relations' are of interest. It is easy to see and detect them.

It would be beneficial to 'directly' connect DSM matrices with assembly components in CAD tools. It means that selecting certain group of components in DSM matrix would offer direct interaction with this group in CAD environment. Users can easily select specific subsets for

[2] Step-by-step schetches of product assembly sequence is nicknamed 'PowerPoint Engineering'. Assembly sketches in exploded views are produced by CAD tools and then put into Microsoft PowerPoint where engineers put numbers that represent the sequence for putting parts together.

closer consideration or navigate through the structure step-by-step. Group can then be hidden, weight of the group can be calculated and any other CAD action can be applied. Integrating DSM and CAD view into one interface offers new perspective on the system architecture.

Matrix representation of system structures possess deficiencies that have been clarified in section **4.3**. Because of those deficiencies, the structural complexity management asks for a supplementary representation by graphs, in particular force directed graphs. Those graphs fulfill the requirements for the intuitive comprehension of visualized structures, enhance the capabilities of matrices concerning the mediation of structural subsets, and allow for extensive possibilities of structure interaction. Thus, force-directed graphs are also appropriate for users without a technical background and can help enhance team work on complex systems. Graphs and matrices can easily be transformed into each other and therefore open advanced possibilities for the representation of complex systems.

Further research is suggested to eliminate the need of user input for setting up threshold values of the box and distance methods. Suggested solution is to select fixed value through the case study or automate the calculation of the value based on available CAD data. For example based on volume of the components and other parameters that could suggest the right threshold.

8. Conclusion

This thesis successfully identified four methods for extracting relations that exist between components in the Siemens NX product assembly model. All methods are described in detail after which the Java PoC is built. Extracted relations are then stored in JSON format which is convinient for usage in any application that aims to improve product traceability. Methods are derived from four main relation types – constraints between components, component proximity, permanent and non-permanent joints.

DSM matrices are chosen as a tool to visualize relations because of their ability to simply visualize a complex system architecture. Additionally, post-processing methods to manipulate DSM data are described for further system analysis.

PoC demonstrated the way it is possible to automate the extraction of relations from the CAD model. Results are successfully verified by comparing the DSM matrix produced by the PoC algorithm with a manually created matrix based on a validation CAD model created for the purposes of this thesis.

Bibliography

[1] S. F. Königs, G. Beier, A. Figge, and R. Stark, *Traceability in Systems Engineering - Review of industrial practices, state-of-the-art technologies and new research solutions*. Berlin: Advanced Engineering Informatics, 2012.

[2] Rob Bracewell et al., *DRED 2.0: A method and tool for capture and communication of design knowledge deliberated in the creation of technical products*. Stanford: ICED, 2009.

[3] Ron Sanchez and Joseph T. Mahoney, *Modularity, flexibility, and knowledge management in product and organization design*. Nedlands/Campaign: John Wiley & Sons, 1996.

[4] Udo Lindemann, Maik Maurer, and Thomas Braun, *Structural Complexity Management: An Approach for the Field of Product Design*. Berlin: Springer, 2009.

[5] W. Chaovalitwongse, H. Pham, S. Hwang, Z. Liang, and C.H. Pham, *Recent Advances in Reliability and Quality in Design - Chapter 21: Recent Advances in Data Mining for Categorizing Text Records*. Piscataway,: Springer, 2008.

[6] T. Naumann, S. Königs, O. Kallenborn, and I. Tuttass, *Social Systems Engineering - An Approach for Efficient Systems Developement*. Copenhagen: Proceedings od the 18th International Conference on Engineering Design, 2011.

[7] M. Storga, N. Bojcetic, N. Pavkovic, and T. Stankovic, *Traceability of Engineering information Development in PLM Framework*. Eindhoven: Proceedings of the 8th International Conference on PLM, 2011.

[8] Steven D. Eppinger and Tyson R. Browning, *Design Structure Matrix Methods and Applications*. Cambridge: MIT Press, 2012.

[9] Boris Tarnovski, *"DSM" razvojnog projekta električnog sportskog automobila*. Zagreb: FSB, Master-Thesis, 2011.

[10] K.B. Clark and T. Fujimoto, *Product Development Performance: Strategy, Organization and Management in the World Auto Industry*. Boston: Harvard Business School Press,

1991.

[11] H.J. Bullinger, E. Kiss-Preussinger, and D. Spath, *Automobilentwicklung in Deutschland – wie sicher in die Zukunft? Chancen, Potenziale und Handlungsempfehlungen für 30 Prozent mehr Effizienz*. Stuttgart: Fraunhofer-IRB, 2003.

[12] C.E. Shannon and W. Weaver, *A mathematical theory of communication*. New York: Bell Systems Technical Journal, 1948.

[13] N.P. Suh, *The Principles of Design*. Oxford: Oxford University Press, 1990.

[14] D.V. Steward, *The design structure system: a method for managing the design of complex systems*. Sacramento: IEEE Transactions on Engineering Management, 1981.

[15] N. Koehler, T. Naumann, and S. Vajna, *Suporting the Modeling of Traceability Information*. Dubrovnik: International DESIGN Conference - DESIGN 2014, 2014.

[16] Tyson R. Browning, *Applying the Design Structure Matrix to System Decomposition and Integration Problem: A Review and New Directions*. Fort Worth: IEEE Transastions on Engineering Management, Vol. 48, No. 3, 2001.

[17] Steven D. Eppinger and Tyson R. Browning, *Design Structure Matrix Methods and Applications*.: MIT Press, 2012.

[18] G.Q. Huang, *Design for X: Concurrent engineering imperatives*. Netherlands: Springer Science+Business Media Dordrecht, 1996.

[19] U. Lindemann, *A vision to overcome "chaotic" Design for X processes in early phases*. Paris: ICED 07, 2007.

[20] G. Schuh and U. Schwenk, *Produktkomplexität managen*. München: Hanser, 2001.

[21] H. Puhl, *Komplexitätsmanagement*. Kaiserslautern: Univ. Kaiserslautern, 1999.

[22] M. Maurer and U. Lindemann, *Facing Multi-Domain Complexity in Product Development*. München: Cidad Working Paper Series 3, 2007.

[23] A. Ericsson and G. Erixon, *Controlling Design Variants – Modular Product Platforms*.

New York: ASME Press, 1999.

[24] C. Y. Baldwin and K. B. Clark, *Design Rules - The Power of Modularity*. Cambridge: MIT Press, 2000.

[25] David Feinleib, *Big Data Bootcamp*. New York: Apress, 2014.

[26] B. Fortner, *Number by Colors*. New York: Springer-Verlag, 1997.

[27] Michael C. Pasqual and Olivier L. de Weck, *Multilayer network model for analysis and management of change propagation*. London: Springer-Verlag, 2011.

[28] M. Broy, *Informatik – Systemstrukturen und theoretische Informatik 2 (2nd edition)*. Berlin: Springer, 1998.

[29] D. M. Sharman and A. Yassine, *Characterizing Complex Product Architectures*. Cambridge: Systems Engineering, Vol. 7, 2004.

[30] J. L. Gross and J. Yellen, *Graph Theory and its Applications, 2nd edition*. London: Chapman and Hall/CRC, 2006.

[31] Yu Tian-Li, Ali A. Yassine, and David E. Goldberg, *An information theoretic method for developing modular architectures using genetic algorithms*. London : Springer-Verlag Limited, 2007.

[32] J. M. Usher, U. Roy, and H. R. Parsaei, *Integrated Product and Process Development – Methods, Tools and Technologies*. New York: John Wiley & Sons, 1998.

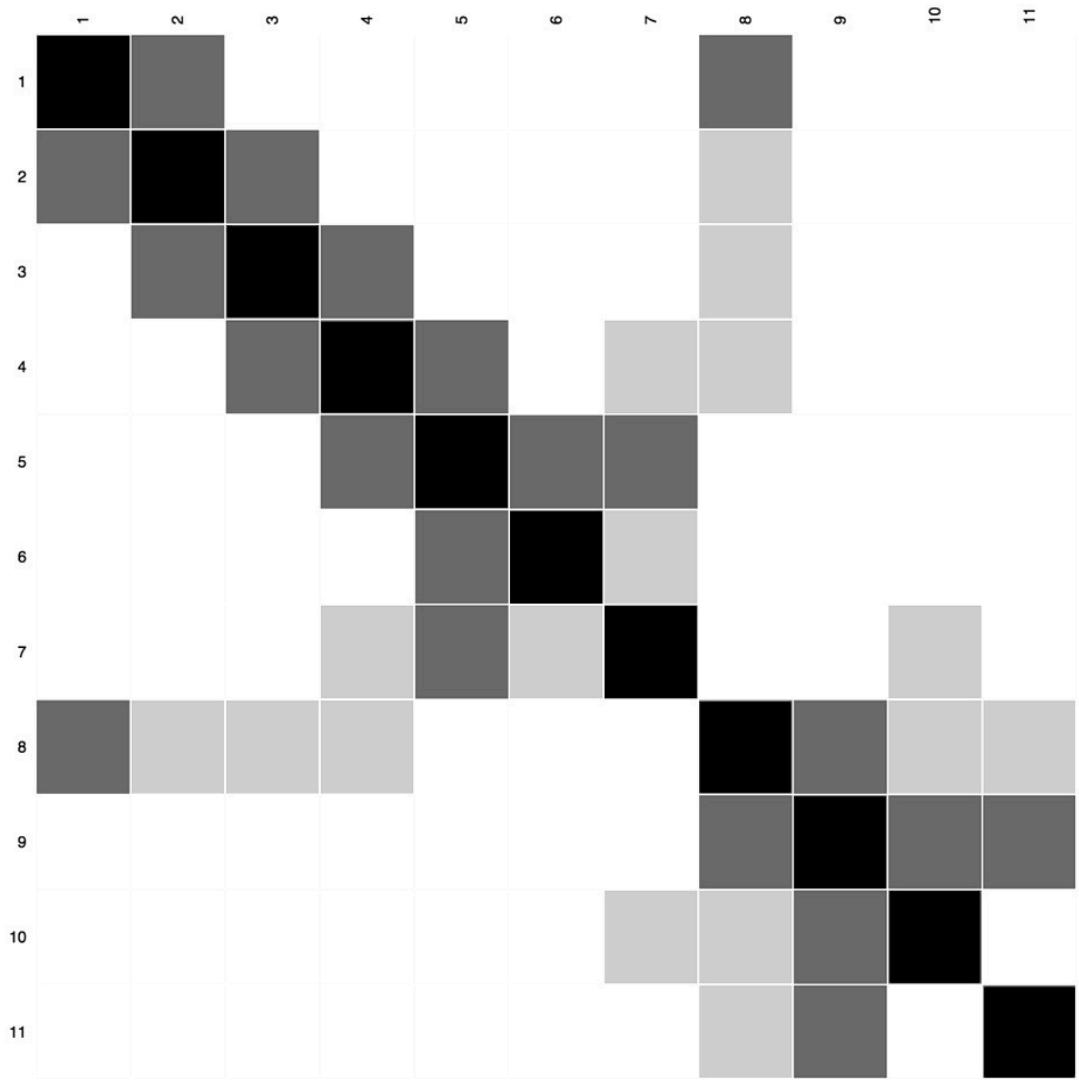
Additional documentation

I. Aggregated DSM comparison

II. CD-R disc

Additional documentation

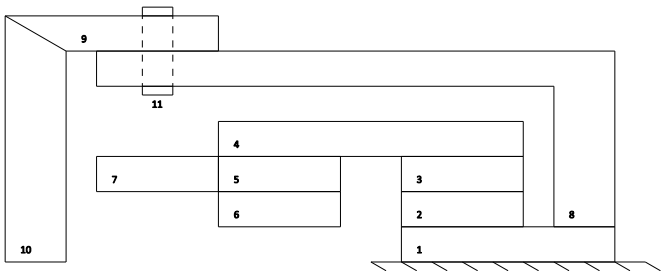
I. Aggregated DSM comparison



Assembly-to-DSM

	1	2	3	4	5	6	7	8	9	10	11
1		|●□						|●□			
2	|●□		|●□					|□			
3		|●□		|●□				|□			
4			|●□		|●□		|□	|□			
5				|●□		|●□	|●□				
6					|●□						
7				|□	|●□					|□	
8	|●□	|□	|□	|□					|●	|□	|●□
9								|●		|●◊□	|●□
10							|□	|□	|●◊□		
11								|●□	|●□		

Manual



Validation model