UNIVERSITY OF ZAGREB
FACULTY OF MECHANICAL ENGINEERING
AND NAVAL ARCHITECTURE

Tino Stanković

GRAMMATICAL EVOLUTION OF TECHNICAL PROCESSES

DOCTORAL THESIS

Zagreb, 2010

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

Tino Stanković

GRAMATIČKA EVOLUCIJA TEHNIČKIH PROCESA

DOKTORSKI RAD

Zagreb, 2010

UNIVERSITY OF ZAGREB
FACULTY OF MECHANICAL ENGINEERING
AND NAVAL ARCHITECTURE

TINO STANKOVIĆ

GRAMMATICAL EVOLUTION OF TECHNICAL PROCESSES

DOCTORAL THESIS

SUPERVISOR:
Prof. dr. sc. Dorian Marjanović, dipl. ing.

Zagreb, 2010

BIBLIOGRAPHY DATA

UDC: 519.7:62-11

Keywords: Operand transformation variants, formal model of technical process, formal model of technical process synthesis, computational design synthesis, graph-grammar, and grammatical evolution.

Scientific area: TECHNICAL SCIENCES

Scientific field: Mechanical engineering

Institution: Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb

Thesis supervisor: Prof. dr. sc. Dorian Marjanović, dipl. ing.

Number of pages: 188

Number of figures: 42

Number of tables: 15

Number of references: 111

Date of oral examination: 13.01.2011.

Thesis defence commission:

Dr. sc. Mario Štorga, Assistant professor - chairman of the defence commission,

Dr. sc. Dorian Marjanović, Professor - PhD thesis supervisor,

Dr. Kristina Shea, Professor at Technical University of Munich - external member,

Archive: Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb

ACKNOWLEDGEMENTS

First of all I would like to thank Professor Dorian Marjanović for his mentorship, encouragement and guidance as well for that little talk we had six years ago about genetic algorithms that lead to accomplishment of this thesis. I would like to express my gratitude to Professor Kristina Shea for her hospitality while my stay at Technical University of Munich and for all of her remarks and suggestions that pointed this research towards graph grammar. A special word of thanks goes to Assistant professor Mario Štorga for our lengthy discussions about the Theory of Technical Systems and some other topics that provided useful insights and most of all a clarification on the matter.

This thesis was greatly influenced by all of the conversations and support provided by the people of Chair of Design and Product Development as well from the CADLab Laboratory. Many thanks go to Associate professor Nenad Bojčetić for resourceful advices on object-based programming. I would also like to thank a friend Damir Deković for many discussions over our coffee breaks, and most of all for all the useful books he managed to pull out of his hat precisely when it was necessary.

Hereby, I would like to thank Ms. Ksenija Ehrenfreund for correcting my English and Assistant professor Boris Ljubenkov for our talk about the working principles related to one of the examples shown within this thesis.

I would like to acknowledge members of the Board for the Postgraduate doctoral studies of the Faculty of Mechanical Engineering and Naval Architecture University of Zagreb who helped to improve this work even further.

Finally, I would like to thank my family and friends for their support and understanding during all of these years. Thank you all.

TABLE OF CONTENTS

BIBLIOGRAPHY DATA I

ACKNOWLEDGEMENTS II

FOREWORD VI

SUMMARY VII

SAŽETAK VIII

PROŠIRENI SAŽETAK IX

LIST OF FIGURES XXI

LIST OF TABLES XXIII

LIST OF SYMBOLS AND ABRIVIATIONS XXIV

1. INTRODUCTION 1

1.1 Motivation 3

1.2 Aim and objectives of the research 6

1.3 Hypothesis 8

1.4 Research methodology 10

1.5 Expected contribution and results 11

1.6 Thesis outline 12

2. ENGINEERING DESIGN SYNTHESIS 15

2.1 Design as a problem solving process 16

2.2 Design as an explanatory search 21

2.3 Engineering design synthesis according to the Domain Theory 24

2.4 Synthesis of technical processes 26

2.4.1 The Theory of Technical Systems 27

2.4.2 Decomposition of technical processes 29

2.4.3 Synthesis of technical process as a prerequisite for novel product design 32

2.5 Implications to this work 35

3. COMPUTATIONAL DESIGN SYNTHESIS 37

3.1 Generic model of CDS 38
3.2 State-of-the-art on CDS 41
3.2.1 Shape and spatial grammars 43
3.2.2 Graph grammars 46
3.2.3 Other approaches to early design support 49
3.2.4 Implications on this thesis 50
3.3 Implications on this thesis 52
4. FORMAL GRAMMARS AND LANGUAGES 54
4.1 Grammars, knowledge representation and engineering design 54
4.2 Production systems 57
4.3 Grammars as production systems 58
4.4 Classification of grammars and Chomsky's hierarchy 60
4.5 Backus-Naur Form 61
4.6 Implication to this work 63
5. GRAMMATICAL EVOLUTION 66
5.1 Recombination and Evolutionary design 67
5.2 Generic model of EA 69
5.2.1 Population 70
5.2.2 Fitness function 71
5.3 Evolutionary operators 72
5.4 Grammatical Evolution 74
5.5 Extension of fitness function 77
5.6 Implications to this thesis 78
6. GRAMMAR OF TECHNICAL PROCESSES 82
6.1 Method overview 83
6.2 Modelling of operand transformation system 85
6.3 Graph grammar of technical processes 90
6.4 Map between Chomsky's grammars and graph grammars 94
6.5 Transformation algorithm and connection procedure $\boldsymbol{\rho}$ 97

6.6 Implications to the knowledge formalisation 100
7. KNOWLEDGE FORMALISATION AND EXAMPLE 102
7.1 Formalization of the knowledge about technical process 102
7.2 Taxonomy and ontology 103
7.3 Foundations for the knowledge formalisation 105
7.3.1 WordNet 106
7.3.2 SUMO 109
7.3.3 Functional basis for engineering design 111
7.4 Examples of methods application 115
7.4.1 Tea-brewing process 115
7.4.2 Design of stiffened panel assembly line 119
7.5 Implications to this thesis 123
8. COMPUTATIONAL TOOL'S ARCHITECTURE 126
8.1 Architecture of computational tool 127
8.1.1 Architectures of Preparation and Execution modules 130
8.2 Class diagrams 132
8.3 GUI 138
8.4 Implications to this thesis 140
9. CONCLUSIONS AND FURTHER WORK 142
9.1 Research summary 142
9.2 Results discussion 144
9.3 Further work 147
10. REFERENCES 148
CURICUULUM VITAE 158
ŽIVOTOPIS 159

FOREWORD

This work is funded by Ministry of Science, Education and Sports of Republic of Croatia as a part of the research project 120-1201829-1828 "Model and Methods of Knowledge Management in Product Development". Provided with a grant funded by MZOS RH a part of this research was conducted at Technical University of Munich where the author spent six months as a guest researcher during academic year 2008/2009. It is to expect that computational approach to generation of operand transformation variants build on a graph-grammar based model of technical process synthesis can efficiently provide support to designers at the beginning of conceptual design phase.

SUMMARY

Keywords: Operand transformation variants, formal model of technical process, formal model of technical process synthesis, computational design synthesis, graph-grammar, and grammatical evolution.

The aim of this thesis is to provide a support to the beginning of conceptual development phase by offering designers the possibility to computationally explore operand transformation variants in technical processes. To accomplish the postulated aim it was proposed that the following objectives have to be met: theoretical objective focused on development of a method for generation of operand transformation variants based on different technological (working) principles, and empirical objective as implementation of the method as a computational tool build to a stage that allows verification of the research results. First, it was necessary to understand the phenomenon of problem solving and cognitive aspects of solution synthesis as a part of problem solving activity. Also, a state-of-the-art review on the computational design synthesis (CDS) [2] was conducted with a purpose to determine theoretical and methodological fundaments of the current research projects and to compare and systematize those in order to focus this research. The efforts where then turned to exploration of the existent mathematical concepts that could be used for the modelling of technical processes and related synthesis method. Based on the findings from the field of CDS it was concluded to conceive the method as knowledge-based with the solution emerging as a result of successive application of a set of production rules in which the knowledge about technical processes and working principles is formalised. The theoretical objective concluded with main scientific contributions of this thesis: (1) the creation of graph based formal model of technical process, (2) the definition of graph-grammar based model of technical process synthesis, (3) addition of stochastic optimisation technical process synthesis by applying grammatical evolution [3]. Within empirical objective of this research, a computational tool was realised on the foundations of developed method for generation of operand transformation variants. During the research it was found out that knowledge about technical processes still does not exists in the accessible open taxonomies or ontologies as per se, which required to propose (4) knowledge formalisation suggestions when defining graph-grammars.

SAŽETAK

Ključne riječi: Varijante transformacije operanda, formalni model tehničkoga procesa, formalni model sinteze tehničkoga procesa, računalom podržana sinteza proizvoda, graf gramatike, gramatička evolucija.

Teorija tehničkih sustava objašnjava tehničku evoluciju, konstruiranje i razvoj proizvoda kao odgovor na potrebe društva koje se mogu ostvariti tehničkim procesima. Takvo teleološko shvaćanje nalaže da je početni korak u razvoju koncepta novog proizvoda utvrđivanje tehničkog procesa kao procesa upotrebe tehničkoga proizvoda tijekom kojega se ostvaruju efekti potrebni za svrhovitu transformaciju operanada obzirom na poznate radne principe na kojima se tehnički proces temelji. Cilj istraživanja u okviru izrade doktorskog rada jest kreiranje računalne podrške upravo za taj početni korak konceptualne faze razvoja proizvoda. Generiranje varijanti transformacije operanada računalnom mogu stvoriti osnovu koja će poslužiti za temeljitije razmatranje mogućnosti za definiranje tehničkog procesa. Sukladno znanstveno-istraživačkoj metodologiji prisutnoj unutar područja Znanosti o konstruiranju, istraživanje u okviru ovoga rada provedeno je unutar dvije faze: teoretska faza koja obuhvaća definiranje metode za generiranje varijanti transformacije operanda temeljem poznatih radnih principa, i praktična faza koja obuhvaća razvitak računalnog alata na osnovu definirane metode do razine koja će omogućiti potvrđivanje rezultata istraživanja. Teoretska faza istraživanja zaključena je sa glavnim znanstvenim doprinosima ove disertacije: (1) definiran je formalni model tehničkog procesa, (2) definiran je formalni model sinteze tehničkih procesa temeljen na graf-gramatikama, (3) uvedena je mogućnost generiranja optimalnih varijanti transformacije koristeći se algoritmom gramatičke evolucije [3]. Praktična faza ovoga istraživanja rezultirala je računalnom implementacijom definirane metode za generiranje varijanti transformacije operanada u okruženju za tu svrhu osmišljenog i razvijenoga računalnoga alata. Tijekom istraživanja utvrđeno je da generalizirano i sistematizirano znanje o tehničkim procesima i radnim principima unutar područja još uvijek nije dostupno u obliku dovoljno detaljne taksonomije ili ontologije za razinu koju zahtijeva definirana metoda. Iz tog razloga predložene su smjernice potrebne za graf-gramatičku formalizaciju znanja o tehničkim procesima i radnim principima (4).

PROŠIRENI SAŽETAK

Uvod

Motivacija za razvoj računalne podrške za konceptualnu fazu procesa razvoja proizvoda temeljena je na pretpostavci da će biti moguće, kada se takva podrška razvije, generirati upotrebljive ili potpuno nove koncepte. Vrlo često se za potrebe generiranje rješenja računalom pribjegava oponašanju kognitivnih procesa u čovjeka, pa se tako algoritmi pretrage vrlo često temelje na slučajnosti, heuristici, učenju na temelju vrednovanja, te izmjeni konačnog broja gradivnih elemenata rješenja koji posloženi sistematski i na novi način mogu rezultirati kreativnim rješenjem problema. Generiranje, vrednovanje i odabir varijante koncepta temeljeni su na konstruktorovu znanju, iskustvu u području, te informacijama prikupljenim iz raznih drugih izvora koje su potrebne za rješavanje danog konstrukcijskog zadatka. Računalni sustav koji bi bio u stanju procesirati informacije na način da kvalitetno generira koncepcijska rješenja značajno bi utjecao na proces konstruiranja i na tehnički proizvod koji se konstruira. Kao rezultat efikasne pretrage prostora koncepcijskih rješenja računalom, trebalo bi se omogućiti stvaranje racionalne osnove kojom se konstruktori mogu voditi prilikom donošenja odluka u ranim fazama razvoja proizvoda. Razumno je očekivati da bi računani sustavi za podršku konstruiranju trebali moći generirati rješenja brzo i da bi producirana rješenja trebala biti korisna.

Na samome početku konceptualne faze, konstrukcijski zadatak se definira sukladno prepoznatim potrebama u društvu i stanju na tržištu. Razmatranje sudjelovanja tehničkog proizvoda u procesu njegovoga korištenja kojime bi se udovoljio potrebama društva i tržišta polazište su za razvoj koncepta proizvoda. Razvoj računalne podrške upravo za tu polaznu točku u razvoju proizvoda odabrana je kao predmet i tema ovog istraživanja. Ovisno o odabranim radnim principima na kojima se će se temeljiti proces realizacije postojećih potreba u društvu koristeći tehnički proizvod, definirati će se zahtjevi i ograničenja kojima mora udovoljiti tehnički proizvod. Sve odluke donošene unutar konceptualne faze uvelike predodređuju kasnije faze procesa razvoja proizvoda, čime se još više naglašava potreba za razvitak i uvođenje računalne podrške.

Odabir teoretskog polazišta istraživanja i ciljevi istraživanja

Kao teorijsko polazište ovoga istraživanja odabrana je Teorija tehničkih sustava. Teorija tehničkih sustava modelira tehničke procese kao sustave transformacija kojima se postiže potrebno stanje operanada materije, signala i energije, te proizvode kao tehničke sustave čije je sudjelovanje neophodno da bi se procesi transformacije izvršili. Takvo teleološko shvaćanje predlaže kao početni korak u razvoju koncepta novog proizvoda utvrđivanje tehničkog procesa kao procesa korištenja proizvoda.

Cilj predloženog istraživanja u okviru izrade doktorskog rada izrečena sukladno Teoriji tehničkih sustava jest kreiranje računalne podrške upravo za taj početni korak konceptualne faze razvoja proizvoda kako bi se konstruktorima omogućilo da bolje i efikasnije razmotre različite mogućnosti za realizaciju proizvoda obzirom na poznate radne principe, potrebe i zahtjeve tržišta, te ograničenja. Pojam radnog principa treba shvatiti kao svaki propisani postupak transformacije definiran operacijama koje je nužno provesti određenim slijedom kako bi se postiglo odgovarajuće konačno stanje operanda. Svaki radni princip specificira i potrebne efekte proizašle kao rezultat aktivnosti čovjeka i/ili tehničkog sustava koji sudjeluju u tehničkom procesu osiguravajući pri tome potrebnu energiju, dodatni materijal, regulaciju i kontrolu. Bez odgovarajućih efekata proces transformacije operanda ne bi bio moguć. Generiranje varijanti dekompozicijom kreće od potrebe opisane sa jednom operacijom, skupom operanada sa poznatim ulaznim i izlaznim stanjima, a završava konačnim skupom varijanti razrađenog procesa transformacije. Iako je iz literature poznato da se analiza tehničkog procesa obično provodi kod konstruiranja potpuno novih proizvoda razrađeni tehnički proces će konstruktoru ipak zorno predočiti kako i u kojoj mjeri čovjek i tehnički sustav sudjeluju u procesu transformacije, te se na taj način precizno određuje funkcija proizvoda. Drugim riječima, ako se sposobnost isporuke odgovarajućih efekata shvati kao funkcija proizvoda u tehničkom procesu, tada je posve jasno da varijacija transformacije na razini tehničkog procesa uzrokuje varijaciju funkcijske strukture proizvoda.

Kako bi se ostvario zadani cilj istraživanja potrebno je ostvariti sljedeće:

1) definirati metodu za generiranje optimalnih varijanti transformacije operanada temeljem poznatih radnih principa, i
2) razviti računalni alat do razine koja će omogućiti verifikaciju rezultata.

Sukladno znanstveno-istraživačkoj metodologiji prisutnoj unutar područja Znanosti o konstruiranju definicija metode za generiranje optimalnih varijanti transformacije operanada temeljem poznatih radnih principa može se shvatiti kao teoretski cilj ovoga istraživanja. Implementacija te metode unutar računalnog alata kako bi se provjerila njena valjanost i na taj način se potvrdili rezultati istraživanja može se shvatiti kao praktični cilj ove disertacije.

Očekivani dugoročni ciljevi istraživanja usmjereni su ka daljnjem razvoju, unapređenju i proširivanju metode i računalnog alata za generiranje optimalnih varijanti transformacije operanada temeljem poznatih radnih principa. Razmatrati će se i moguća integracija u postojeća okruženja za Računalom podržanu sintezu proizvoda (*eng. computational design synthesis, CDS*). Kada bi se prevladale razlike nastale kao posljedica drugačijih teoretskih osnova na kojima se temelje formalni modeli tehničkog sustava i procesa konstruiranja, dodavanjem razine tehničkog procesa u neko od postojećih okruženja omogućilo bi se proširenje područja pretrage uzimajući u obzir tehnički proces i sudjelovanje tehničkog sustava u transformaciji na taj način definirajući funkciju tehničkoga sustava.

Hipoteza rada i istraživačka pitanja

Sukladno definiranim teoretskim polazištima i ciljevima, hipoteza ovoga rada glasi:

Koristeći skup produkcijskih pravila kojima je formalizirano inženjersko znanje o tehničkim procesima, radim principima i potrebnim efektima mogu se generirati varijante transformacije operanada kao polazište za daljnji razvoj koncepta tehničkog proizvoda.

Hipoteza ovoga rada ipak je ponajviše umjerena ga teoretskim ciljevima ovoga istraživanje. Definiranje formalnog modela tehničkoga procesa, kao i definiranje formalnog modela sinteze tehničkoga procesa imaju stoga najveće težište unutar ovoga istraživanja. Za predvidjeti je da će se empirijski dio istraživanja moći provesti korektno samo unutar okvira stvarne inženjerske prakse. Da bi se to ostvarilo, prethodno se trebaju realizirati dugoročni ciljevi ovoga istraživanja.

Obzirom na definiranu hipotezu istraživačka pitanja postavljena su kako slijedi:

- Kako interpretirati tehnički proces na računalu razumljiv način? Cjelokupna upotrebljivost metode za generiranje varijanti transformacije operanda ovisiti će o načinu na koji će se formalno modelirati tehnički proces.

- Kako generirati varijante transformacije operanada unutar tehničkog procesa? Koji matematički koncept upotrijebiti za definiranje formalnog modela sinteze tehničkoga procesa? Da li će razvijeni model težiti više ka emuliranju kognitivnih procesa u čovjeka ili će biti temeljen na heuristici?
- Da li je moguće uvesti optimizacijske metode na razinu tehničkog procesa? Po kojim kriterijima je moguće provesti optimizaciju?
- Kako je predloženo da se metoda temelji na formalizaciji znanja o tehničkim procesima i poznatim radnim principima koristeći produkcijska pravila, koje su tada osnove za formalizaciju takvog znanja?

Očekivani znanstveni doprinos

Sukladno svemu navedenome očekivani znanstveni doprinos istraživanja se može sažeti na sljedeći način:

- Formalni model tehničkoga procesa. Uvođenjem razine tehničkoga procesa postignut je doprinos unutar područja Računalom podržane sinteze proizvoda koja je do sada razmatra samo funkcijsku razinu kao najvišu razinu apstrakcije.
- Formalni model procesa sinteze tehničkoga procesa. Rezultat sinteze bi trebale biti varijante transformacije operanada temeljene na poznatim radim principima i potrebnim efektima.
- Implementacijom metode unutar računalnoga alata omogućeno je generiranje varijanti transformacije operanda koji mogu poslužiti kao osnova za dalji razvoj koncepta proizvoda.

Metodologija istraživanja

Metodologija predloženog istraživanja slijedi deskriptivnu DRM metodologiju (eng. *design research methodology*). Plan rada predloženog istraživanja sukladno DRM metodologiji može se objasniti sljedećim koracima:

1) Analiza koja obuhvaća pregled postojeće stručne i znanstvene literature unutar područja kako bi se opravdali i razjasnili ciljevi i svrha istraživanja. Prikupljenim činjenicama se utvrđuje postojeće stanje računalne podrške za rane faze procesa razvoja proizvoda, tj. područja koje se predloženim istraživanjem želi unaprijediti. Cilj predloženog istraživanja definira se kao dvostruki, jer razvoj računalne podrške

namijenjene ranim fazama razvoja proizvoda nedvojbeno povlači i prethodno definiranje računalne metode.

2) Određivanje teoretskih osnova potrebnih za izradu računalne metode pomoću koje će se generirati varijante procesa korištenja proizvoda. Pri tome posebno će se razmatrati različiti fenomenološki modeli procesa konstruiranja i razvoja proizvoda kako bi se odabrala odgovarajuća teoretska podloga potrebna za razvoj računalne podrške. Istraživanja koja za cilj imaju razvoj računalne podrške procesu konstruiranja zbog svoga opsega gotovo uvijek nalažu uključivanje multidisciplinarnog pristupa u istraživanje, pa je iz tih razloga nužno proširiti razumijevanje problema koji se stiče uvidom stručne i znanstvene literature u strogom području istraživanja. Analizirati će se rezultati istraživanja iz raznih znanstvenih disciplina čije bi spoznaje mogle pridonijeti ovome istraživanju. Neke od znanstvenih disciplina koje će se razmotriti jesu: evolucijsko računarstvo, više-ciljna optimizacija, formalni jezici, teorija sustava i dr.
3) Sinteza koja uključuje izradu računalne podrške za razinu utvrđivanja tehničkog procesa temeljem teoretskih osnova utvrđenih u prošlom koraku. Prvo će se izraditi matematički i informacijski modeli metode, a zatim će se definirati arhitektura računalnog alata.
4) Verifikacija sadrži vrednovanje rezultata istraživanja empirijskom analizom mjere u kojoj razvijeni alat utječe na povećanje efikasnosti i kvalitete proizvoda. Pristupiti će se izradi računalnog modela alata u nekom od postojećih programskih jezika ili razvojnih okruženja. Biti će dan i osvrt na primjenjivost metode i alata za sintezu tehničkog procesa obzirom na vrstu inženjerskog zadataka, odnosno tehničkog sustava koji se razvija. To je posebno važno obzirom da se generiranje varijanti tehničkog procesa preporučuje kod razvoja potpuno novih ili kod razvoja vrlo složenih tehničkih sustava, jer kod njih zbog velikog broja elemenata i relacija među njima broj mogućih alternativa na samome početku konceptualne faze razvoja proizvoda može biti vrlo velik.

Razmatranje sinteze kao aktivnosti u procesu konstruiranja

Unutar ovog poglavlja analizirao se proces konstruiranja kao proces rješavanja problema, s težištem postavljenim na sintezu kao jednu od aktivnosti unutar procesa konstruiranja. Rezultatima provedene analize utvrdili su se principi koji su potrebni za razvoj računalne

podrške za definiranje i sintezu tehničkih procesa. Rješavanje problema obuhvaća razne kognitivne procese i aktivnosti prisutne i kod rješavanja konstrukcijskih problema, pa se analizom utvrdilo da su osnovni mehanizmi rješavanja konstrukcijskih problema sljedeći:

- analiza i formulacija problema,
- naizmjenično generiranje i vrednovanje rješenja,
- apstrahiranje i specificiranje,
- dekompozicija sa ciljem utvrđivanja elemenata sustava i njihovih međuovisnostima,
- odlučivanje i odabir varijante rješenja,
- heuristika i metode pretrage temeljene na znanju.

Sa stanovišta utvrđeno je da je proces konstruiranja proces pretrage koji za cilj ima objašnjavanje rješenja. Abduktivno promišljanje tijekom procesa konstruiranja rezultira generiranjem tehničkoga opisa proizvoda kao objašnjenja postojećih društvenih potreba i potraživanja na tržištu. Konstrukcijsko rješenje nastaje kao rezultat ko-evolutivnog iterativnog procesa između predloženog rješenja problema, te razine znanja i razumijevanja o problemu koji je zadan. Drugim riječima, definicija problema i rješenje problema se u procesu konstruiranja sistematski vrednuju i redefiniraju jedno u odnosu na drugo. Zbog postojanja takve međupovezanosti ne može se ustvrditi da li je generirano rješenje konstrukcijskog problema i konačno, tj. te da li ono stvarno najbolje rješenje koje udovoljava stvarnim potrebama u društvu i na tržištu.

Konačno, dekompozicija tehničkih procesa razmatrana je obzirom na Teoriju domena i Teoriju tehničkih sustava. Zaključeno je da konstruktor modelira strukturu tehničkog proces da se transformacijom operanda postigne potrebno stanje. Sinteza tehničkoga procesa dio je stoga faze utvrđivanja konstrukcijskoga zadataka, obzirom da sadrži operande u početnom i završnom stanju koji odgovaraju postojećim potrebama, odnosno zadovoljenju tih potreba u društvu. Analizom Teorije domena utvrđeno je da opće odlike tehničkoga procesa (*eng. universal virtues*) ukoliko se mogu mjeriti, mogu poslužiti kao funkcije cilja prilikom optimizacije. Konačno, sukladno Teoriji domena i Teoriji tehničkih sustava utvrđeno je sinteza tehničkog procesa nužna samo kod razvoja potpuno novog proizvoda.

Računalom podržana sinteza proizvoda

Analizirane su postojeće metode i alati u području Računalom podržane sinteze proizvoda (CDS) kako bi se utvrdile teoretske osnove i principi koje se koriste u svrhu podrške pojedinim fazama razvoja proizvoda. Zaključci koji su proizašli iz te analize poslužili su za određivanje doprinosa području CDS-a, te principa na kojima će se definirati metoda za generiranje varijanti operanada u tehničkome procesu. Računalom podržana sinteza proizvoda implicira sistematski pristup definiranju algoritama i metodološkog modeliranja tehničkoga sustava i procesa konstruiranja sa ciljem kreiranja konstrukcijskih rješenja upotrebom računala. CDS jest složeno multidisciplinarno istraživačko područje koje obuhvaća napredne računalne tehnike i algoritme pretrage, te znanje o procesu konstruiranja i tehničkim sustavima. Utvrđeno je da se CDS metode vrlo često temelje na oponašanju kognitivnih procesa čovjeka prisutnih prilikom rješavanja problema, odmičući se tako od standardne inženjerske optimizacije. namjena postojećih CDS metoda i alata može se formulirati kao pružanje podrške konstruktoru u situacijama kada pronalaženje potrebnog rješenja problema iziskuje generiranje prevelikoga broja varijanata da bi se ono moglo riješiti u potrebnom vremenu. Pregled područja dan je u nastavku.

Osnovni pristupi. Dobar primjer uporabe evolucijskog računarstva za ostvarivanje računalne podrške u konceptualnoj fazi razvoja proizvoda jest korištenje genetičkog algoritma za generiranja konceptualnih varijanti temeljem morfološke matrice koja sadrži različita tehnička rješenja parcijalnih problema. Vrednovanje konceptualnih varijanti rješenja provodi se provjeravanjem kompatibilnosti tokova energije, dok se optimalno rješenje traži temeljem definirane funkcije cilja. U istraživanju je korišten Računalni generator koncepta (*eng. Concept generator*) Matrična algebra korištena je za određivanje komponenti kojima je moguće realizirati odgovarajuće funkcije iz zadane funkcijske strukture. Drugačiji pristup računalnoj podršci namijenjenoj konceptualnoj fazi razvoja proizvoda donosi metoda A-Design. Okosnica podrške jest skup softverskih agenata – računalnih programa kojima ugrađeno znanje omogućava i propisuje izvršavanje točno određenih zadaća potrebnih za kombiniranje komponenti u smisleno tehničko rješenje. Kompatibilnost međusobnog spajanja komponenti provjerava se na ulazno/izlaznim sučeljima.

Graf-gramatičke metode temeljene na formalnim jezicima. Pristup inspiriran pionirskim radovima Chomskog i Minskog na polju formalnih jezika, koristi formalne gramatike kako bi se na lingvistički način opisala pojedina konstrukcijska rješenja, te tako definirao jezik

konstruiranja. Korištenjem gramatika se znanje o konstruiranju, koje postoji u specifičnom području primjene, formalizira skupom pravila. Izvršavanjem svih dopuštenih kombinacija pravila stvara se velik, ali ipak konačan broj varijanti rješenja čime su definirane granice prostora pretraživanja, odnosno jezik područja. Iz literature su poznate gramatike oblika (eng. *shape grammars*) koje se zajedno sa stohastičko-optimizacijskim algoritmom simuliranog popuštanja (*eng. simulated annealing*) primjenjuju za potrebe strukturne i topološke optimizacije. Na sličnim principima se temelji razvoj računalne podrške za konceptualnu fazu razvoja proizvoda gdje se za formalizaciju znanja u području najčešće primjenjuju graf gramatike (*eng. graph grammars*). Pribjegavanje graf gramatikama može se opravdati činjenicom da gotovo sve teorije koje se bave proučavanjem procesa konstruiranja u konceptualnoj fazi modeliraju proizvod kao transformacijski sustav koji se formalno i vizualno opisuje grafom. Graf gramatike pri tome opisuju skup pravila kojima se može transformirati graf, te uvjeta koji određuju kada se započinje i završava izvršavanje pravila. Pregled istraživanja u području pokazao je da su postojeće metode koje koriste graf gramatike orijentirane na dekompoziciju na razinama funkcija i komponenti proizvoda. Optimizacija se najčešće provodi samo za razinu komponenti.

Dobar primjer kako se računalom može simulirati proces generiranja koncepata proizvoda prikazan je unutra HiCED okruženja. Pristup se temelji na ko-evolutivnom razvoju proizvoda na različitim razinama apstrakcije. Postupak započinje generiranjem populacije funkcijskih dekompozicija zadanog proizvoda koristeći formalizirano znanje u obliku pravila. Nakon toga se pomoću genetičkog programiranja i genetičkoga algoritma istodobno evoluiraju funkcijska struktura i komponente koje su u stanju realizirati pojedine funkcije. Funkcija cilja formulirana je uz korištenje težinskih faktora. Ideja ko-evolucije na različitim razinama apstrakcije proizvoda objašnjena je u okviru Opće teorije konstruiranja (*eng. General Design Theory*) i FBS (*eng. Function-Behaviour-State*) pristupa modeliranju proizvoda.

Novije istraživanje temeljeno na FBS modelu pokušava iskoristiti i objediniti dostupne računalne alate kako bi se kreiralo cjelokupno okruženje za računalom podržanu sintezu mehatroničkih proizvoda. Za provedbu graf-gramatičkih transformacija koristi se GrGen, vizualizacija grafova riješena je unutar okruženja TULIP, a za modeliranje tehničkoga proizvoda pokušava se iskoristiti SysML. Računalno okruženje omogućava korisniku da se pravila definiraju unutar vizualnog sučelja pomoću skriptnog programskog jezika koji je integralni dio GrGen-a. Generiranje varijanti konceptualnog rješenja provodi se unutar posebnog modula. Za sada okruženje ne podržava optimizacijske metode.

Ostali pristupi. Nešto drugačiji pristup kreiranju računalne podrške konceptualnoj fazi razvoja proizvoda dostupan je unutar računalnog okruženja *Cambridge Advanced Modeller* (CAM) razvijenog na temelju *P3-Signposting* alata. Računalna podrška namijenjena je sintezi arhitekture proizvoda koja se unutar CAM-a opisuje pomoću mreže komponenata. Korisnik kroz vizualno sučelje koristeći grafički jezik za modeliranje definira ulaznu shemu koja može sadržavati komponente različitog tipa, više tipova relacija te ograničenja temeljena na predikatnoj logici prvoga reda. Varijante arhitekture generiraju se računalno analizom zadane sheme.

Temeljem istraženoga utvrđeno da se razvitak predložene metode temelji na formalizaciji znanja o tehničkim procesima i radim principima u obliku produkcijskih pravila čijom će se sukcesivnom provedbom moći generirati varijante transformacije operanada. U svrhu modeliranja tehničkih procesa odabran je koncept grafa, generiranje varijanti transformacije operanada provesti će se upotrebom graf-gramatika, dok će se optimizacija implementirati kroz algoritam gramatičke evolucije.

Rezultati istraživanja

Sukladno motivaciji koja je potaknula na ovo istraživanje, definiranim teoretskim polazištima i ciljevima istraživanja, te istraživačkim pitanjima postignuto je sljedeće:

- Osmišljena je i definirana graf-gramatička metoda za generiranje varijanata transformacije operanda. Većina metoda unutar Računalom podržane sinteze proizvoda se za kreiranje podrške ranim fazama konstruiranja oslanja se na isti princip pretrage i generiranja rješenja koristeći formalizirano znanje. Područje primjene i kvaliteta generiranih rješenja tako postaju i funkcija formaliziranoga znanja, a ne samo algoritma provedbe pretrage u užem smislu. Pokazano je da se upotrebom graf-gramatike koristeći formalizirano znanje o tehničkim procesima i radim principima u obliku produkcijskih pravila može provesti dekompozicija tehničkih procesa do sinteze varijanti. Pri tome su se koristile transformacije implementirane na razini za pojedinačnog čvora grafa odnosno na razini pojedinačne operacije tehničkoga procesa. Za svaku transformaciju definirana su pravila umetanja i spajanja novih pod-struktura u postojeću strukturu grafa, odnosno dekompozicije pojedinog pod-procesa u niz operacija, te njihova integracija u postojeću strukturu tehničkoga procesa. Algoritam dekompozicije tehničkog procesa, kao i pravila umetanja i spajanja definirana su šestom poglavlju ove disertacije.

- Za potrebe kreiranja formalnog modela tehničkog procesa osmišljen je usmjereni multigraf sa operacijama, operandima i efektima (šesto poglavlje disertacije). Nužno je bilo uvesti takav model tehničkoga procesa kako bi bilo moguće opisati sve tokove operanada koji mogu postojati između dvije operacije unutar tehničkog procesa. Jednako tako, definirani formalni model tehničkoga prosca u stanjuje opisati tokove koji ulaze i izlaze iz transformacijskoga sustava, te efekte. Sukladno objektnom-pristupu operacije, operandi i efekti definirani su kao entiteti tehničkoga procesa, pridruženi su čvorovima i lukovima grafa nosioca čime nastala struktura opisuje tehnički proces. Usmjereni multi graf invarijantan je obzirom na složenost entiteta tehničkoga procesa i njihove međusobne relacije koje se mogu uspostaviti izvan okvira tehničkoga procesa, tako ostavljajući prostor za daljnje unapređenje metode na razini entiteta. Slično je definirana metoda dekompozicije na način da ovisi samo o tipu grafa nad kojim je definirana. Entiteti tehničkoga procesa zajedno čine vokabular tehničkih procesa.
- Varijante transformacije operanda mogu se kreirati na način da se zadani aksiom, odnosno odabranu operaciju sa operandima i njihovim stanjima specificiranim prije i poslije transformacije, generiraju sve moguće varijante upotrebom skupa produkcijskih pravila, na taj način prikazujući jezik tehničkih procesa za tu operaciju u okvirima znanja koje je formalizirano. Kako bi se generirale samo optimalne varijante unutar jezika tehničkoga procesa koristi se algoritam gramatičke evolucije (peto poglavlje disertacije). Kako se na trenutnoj razini istraživanja entiteti tehničkog procesa još uvijek ne dopuštaju postojanje atributa koji bih pobliže i detaljnije opisali, obzirom da se za definiranje istih prvo zahtijeva generalizacija i sistematizacija znanja o tehničkim procesima, trenutno nije moguće uspostaviti složene i realne kriterije vrednovanja tehničkih procesa koji bi poslužili za formulaciju funkcija cilja. Iako je u okviru računalnog alata prikazanog unutar ove disertacije razvijena i podrška za više-ciljnu inženjersku optimizaciju koristeći algoritam gramatičke evolucije, trenutno nije moguće opisati pretragu složenije od jednociljnog optimiranja sa ograničenjima.
- Kako se definirana metoda za generiranje varijanti transformacije operanada u tehničkom procesu temelji na znanju, potrebno je istražiti zahtjeve koji moraju biti zadovoljeni kako bi se uspješno formaliziralo znanje o tehničkim procesima i poznatim radnim principima koristeći produkcijska pravila. Potrebno je naglasiti da ova disertacija nije imala za cilj definiranje generalizacije i sistematizacije znanja o

tehničkim procesima, već je namjera bila da se samo kreiraju mehanizmi koji će omogućiti formalizaciju znanja i njegovu upotrebu za generiranje varijanti rješenja koristeći skup produkcijskih pravila. Tijekom istraživanja utvrđeno je da generalizirano i sistematizirano znanje o tehničkim procesima i radnim principima još uvijek nije dostupno u obliku dovoljno detaljne taksonomije ili ontologije za razinu koju zahtijeva definirana metoda kako bi se bilo u stanju detaljnije opisati tehnički proces. Iz tog razloga predložene su smjernice potrebne za graf-gramatičku formalizaciju znanja o tehničkim procesima i radnim principima (sedmo poglavlje disertacije). Predložene smjernice uključuju formalizaciju na temelju dostupnih jezičnih leksikona (*WordNet*), inženjerskih ontologija (*SUMO*) i drugih istraživanja u području kako bi se znanje o tehničkim proscima definiralo koliko je god to moguće konzistentno. Na kraju, kao potvrda valjanosti koncepta definirane metode pokazana su dva primjera.

Ovo istraživanje temelji se na uspješnosti definiranja metode za generiranje transformacije operanda tehničkoga procesa uključivši formalni model tehničkoga procesa i formalni model dekompozicije tehničkoga procesa, te njihovu implementaciju u okviru računalnog alata. Formalni modeli koji uključuju usmjereni multigraf sa operacijama, operandima i efektima i njegovu graf-gramatičku dekompoziciju kako su definirani unutar ove disertacije mogu se smatrati doprinosim znanstveno-istraživačkom području Računalom podržane sinteze proizvoda. Doista je teško usporediti metodu razvijenu unutar ove disertacije sa ostalim metodama i alatima unutar područja računalom podržane sinteze proizvoda obzirom da su proizašle iz potpuno različitih teoretskih osnova. Namjera ovoga istraživanja bila je i stvoriti uvijete za kreiranje cjelokupnoga okruženja za podršku ranim fazama razvoja proizvoda, koje bi se tada temeljile na sličnim formalnim modelima.

Gramatička evolucija jest robustan stohastički optimizator koji je moguće primijeniti za sve razine ranih faza razvoja proizvoda pod uvjetom da se za formalni modeli uključuju modeliranje graf-gramatikama, odnosno zapis znanja pravilima. Gotovo sve teorije konstruiranja za modeliranje ranih faza temeljene model tehničkog proizvoda su na Teoriji sustava. Iz toga razloga bilo je prirodno odabrati osnove za modeliranje tehničkoga procesa i njegove dekompozicije sukladno teoretskim polazištima. Za pretpostaviti je da će pravo vrednovanje ovoga pa i srodnih alata i metoda biti moguće kada se one implementiraju u stvarnoj inženjerskoj praksi kao dijelovi cjelovitih okruženja za računalom podržanu sintezu

proizvoda. Konačno, temeljem postignutih rezultata istraživanja sukladno postavljenim ciljevima i istraživačkim pitanjima može se zaključiti da je hipoteza ovoga rada potvrđena.

LIST OF FIGURES

Figure 1.1 General model of transformation system according to TTS 5
Figure 2.1 Causality in design's degrees of freedom [8], [24] 16
Figure 2.2 Design as an explanatory search according to GDT [30].....22
Figure 2.3 Domain Theory applied to design object [8].....24
Figure 2.4 General model of transformation process [1].....28
Figure 2.5 Structure of technical process according to TTS [1].....30
Figure 2.6 Decomposition of technical process.....31
Figure 2.7 Horizontal and vertical action chain [1], [8] 34
Figure 3.1 A generic model of CDS [2].....38
Figure 3.2 Parametric synthesis model [38] 38
Figure 3.3 Planar truss grammar (f - free line, black dot - fixed point, white dot - free point) [43].....44
Figure 3.4 Example of rule applications [43] 44
Figure 3.5 Cantilever truss designs for minimum mass using shape annealing [43].....44
Figure 3.6. Packing bounding box and shaft and gear [9] 45
Figure 3.7 Component vs. graph representations [9].....45
Figure 3.8 Rule examples [9].....45
Figure 3.9 Design process of HiCED [54].....47
Figure 3.10 Top-down graph grammar approach on all three levels of FBS [58].....49
Figure 6.1 IDEF0 model of generation of operand transformation variants within TP.....83
Figure 6.2 An example of TP modelled by $G$with TP entities.....87
Figure 6.3 Multi-digraph with operations, operands and effects and its incidence matrix.....89
Figure 6.4 Example of rule p left hand side L.....92
Figure 6.5 Identification of match of L as given in the Figure 6.4 using morphism $m(L)$ from Definition 6.5 93
Figure 6.6 Example of the BNF derivation process and its map to graph grammars 95
Figure 6.7 Example of resulting TP structures after map from BNF to graph grammar language 96
Figure 6.8 Transformation steps $G \rightarrow G \dashrightarrow GT$ 100

Figure 6.9 Emergence of secondary flows (principle transformation marked with asterisk) in right hand side of rule $p: L \rightarrow R$ 101
Figure 7.1 Semantic links between terms related to TP's according to *WordNet* [104]..... 107
Figure 7.2 Detail semantic links in the context of operand transformation as in *WordNet* [104] 108
Figure 7.3 Upper ontology as proposed by SUMO [107]..... 110
Figure 7.4 Tea-brewing black-box process formulation..... 115
Figure 7.5 Production application sequence of tea-brewing process (only left hand sides of applied production is shown, unreachable branch expressed as dashed)..... 119
Figure 7.6 Stiffened panel assembly black-box process formulation..... 120
Figure 7.7 Production application sequence of tea-brewing process (only left hand sides of applied production is shown, recursive branches expressed as dashed, goal gray-shaded) .. 124
Figure 8.1 Schematics of the kernel of system for generation of operand transformation variants..... 128
Figure 8.2. Three-tier computational tool's architecture with participation in design process 129
Figure 8.3 Component diagrams of DMU and GMU with data/object flow..... 130
Figure 8.4 Architecture of Preparation and Execution modules with principle components, dynamic link libraries and dataflow..... 131
Figure 8.5 Class diagram of *TP Entities* and multi-digraph with operations, operands and effects..... 135
Figure 8.6 Class diagram of grammatical evolution search and optimisation framework 137
Figure 8.7 Screenshot of interactive rule builder GUI in Preparation module..... 139
Figure 8.8 Screenshot of synthesized tea-brewing process (example 7.4.1) as shown in Execution's module GUI 140

LIST OF TABLES

Table 2.1 Overview of design activities 20
Table 3.1 Overview of CDS methods and tools 51
Table 4.1 Chomsky's hierarchy of grammars [19] 60
Table 5.1 Decoded chromosome and rule sequence selection 76
Table 6.1 Matrix representation of operand transformation process 89
Table 7.1 A taxonomy of operands (operand flows) as accepted by NIST [22] 113
Table 7.2 Taxonomy of technical products' functions as accepted by NIST [22] 114
Table 7.3 Context-free grammar $CFGTP$ of tea-brewing process in BNF 117
Table 7.4 Correspondence between $CFGTP$ and TTS 117
Table 7.5 Graph-grammar of tea-brewing 118
Table 7.6 Context-free grammar $CFGTP$ of stiffened panel assembly process in BNF 121
Table 7.7 Graph-grammar of stiffened panel assembly (part I) 122
Table 7.8 Graph-grammar of stiffened panel assembly (part II) 123
Table 8.1 Correspondence between TP data model and method as defined within Chapter 6. 134
Table 8.2 Correspondence between GE search and optimisation classes and GE (as in Chapter 5). 136

LIST OF SYMBOLS AND ABRIVIATIONS

A	String variable
a	Evolutionary population individual/chromosome
a^*	Optimal individual/chromosome
B	Any number of variables
BNF	Backus-Naur Form
CFG	Context-free grammar
CFG_{TP}	Context-free grammar of technical processes
c	Single terminal
D, D_B	Decoding function, Binary decoding function
$D_{\mathcal{L}(\mathcal{G})}$	Map from decoded technical process individual to context-free language of technical processes
$D_{\mathcal{L}_{TP}(\mathcal{GG})}$	Map form f context-free language of technical processes language of technical processes
Eff	Effect
e	Relation in graph
e_{mn}	Dependency
F	Fitness function
f	Any objective function
f_{TP}	Technical process related fitness function
G	Multidigraph with operations, operands and effects
$\mathcal{G}$	String grammar
$\mathcal{GG}$	Graph grammar
I	Individual space of an algorithm
I_{TP}	Technical process related individual
$I^{\mu}, I^{\lambda}, I^{\kappa}$	Set all evolutionary populations of sizes μ, λ, κ
l_E, l_V	Labelling functions
k_i	Insertion place within incidence matrix
L	Left hand side of graph-grammar production
$\mathcal{L}, \mathcal{L}(\mathcal{G})$	Formal language, formal language of string grammar $\mathcal{G}$

$\mathcal{L}_{TP}(CFG_{TP})$	Context-free language of technical processes
$\mathcal{L}_{TP}(\mathcal{GG})$	Graph-grammar language of technical processes
m	A match of left hand side of a graph-grammar production in a host graph
m_{EVOP}	Evolutionary mutation operator
Od	Operand
Op	Operation
P	Evolutionary population of parents
P'	Evolutionary population of offspring
P''	Mutated evolutionary population
P^*	Optimal population
$\mathcal{P}$	Finite nonempty set of production rules in string grammar
$\mathcal{P}_G$	Finite nonempty set of production rules in graph-grammar of technical processes
$\mathcal{P}_s$	Production rules in context-free grammar of technical processes
p	Graph-grammar production
R	Right hand side of graph-grammar production
r_{EVOP}	Evolutionary recombination operator
r_{mn}	Incidence matrix of multidigraph with operations, operands and effects
s	Source mapping to relation e
s_{EVOP}	Evolutionary selection operator
$\mathcal{S}$	Starting symbol or axiom
$\mathcal{S}_s$	Axiom in context-free grammar of technical processes
T_s	Evolutionary fitness scaling function
t	Target mapping to relation e
u	Vertex of a graph
V	Set of graph's nodes
v	Vertex of a graph
$\mathcal{V}$	Finite nonempty set of variables in string grammar
$\mathcal{V}_s$	Variables of context-free language of technical processes
α	Left hand side of production in string grammar
β	Right hand side of production in string grammar
γ_i	Binary decoding function per individual substring/chromosomal gene
Δ	Alphabet of graph-grammar language of technical processes

δ, δ_i	Substring/gene of evolutionary individual/chromosome, Substring at position/locus
λ	Size of offspring population
μ	Size of parent population
ε	Zero length string
$\Theta_s, \Theta_m, \Theta_r, \Theta_\iota$	Evolutionary selection, mutation, recombination and termination condition parameters
ι	Evolutionary termination condition
ρ	Connecting procedure
Σ_{Eff}	Set of effects
Σ_G	Set of technical process entities
Σ_s	Alphabet of context-free language of technical processes
Σ_{Od}	Set of operands
Σ_{Op}	Set of operations
Φ	Evolutionary objective function
Φ_{TP}	Universal virtues objective function
ω	Word in formal language
<token>	BNF representation of starting symbol/variable
_token	BNF representation of terminal

1. INTRODUCTION

Understanding activities and mental processes that are performed when designing creates a rational basis on which these activities could be logically structured with the purpose of improving the object that is being designed. Development of various design methods and tools for supporting a design process give better overview and understanding of the task at hand, which in the end may increase the possibilities of designing more efficient and performance-optimised products. Introduction of a systematic method to the product development [1], [4], [5] [6] and [7] is therefore motivated by causal relationship between a product and the design process through which that product was conceived and created. Stimulating innovativeness and creativity, be able to objectively evaluate in order not to jump to the first available solution to a given problem and implementation of different search strategies are some aspects of the support provided to designers when following systematic methods [1]. Moreover, the transformation of a design process into a prescribed problem solving procedure enables activities such as planning, standardization of task solving process and utilization of past solutions while providing the possibility to learn on account of prior experience.

As in many other disciplines, computers have also found their place within the product development process. The advent of computers has made possible the development of engineering design support tools which have been intended for tasks that required a large number of repetitive operations performed accurately. These tasks are hard and tedious for designers to be solved manually and although they could perform sufficiently well in respect to being accurate, requirements such as shortening of product time-to-market in combination with the increasing complexity of products that has to be managed imposed serious time constraints. So in order to be able to manage increasing number of tasks and assignments, a translation of problem solving procedures to computational algorithms and tools is a reasonable step to be taken. One of the most common examples are various types of computational analysis and simulation tools where a given problem is described by a finite set of equations that need to be solved numerically in order to obtain solution. Likewise, computational support is desired in order to be able to perform fast and efficient engineering optimization tasks. The problem of finding an optimum may well include various search

strategies being applied to a multidimensional space comprised of a large enough number of possible solution variants. Application of heuristics is a probable best fit for exploration of such large design solution spaces which emerged as a result of involving multiple-criteria and n-dimensional vectors as arguments and the results obtained most often outperform designers not only by calculation time but also by its quality.

Depending on the purpose the translation of complex engineering problems and problem solving procedures to develop computational design support tools sometimes requires formalisation that moves beyond just proper problem representation and solver algorithm development. In addition an adequate knowledge encoding must be devised which will have to suit both the problem statement and the mechanism of the search algorithm. The engineering knowledge about the problem has always been and will always be an integral part of these support tools at least as a part of the evaluation step where behaviour is determined by calculating the properties of a technical system considered. However the distinction is in the approach by which the formalisation has been accomplished and for which purpose. Rather than being solely for the evaluation of the solution, formalised engineering knowledge can also be a part of the solution proposition process where it is used to generate the structure of a design which will be later evaluated. That is a distinction in respect to search strategies that rely only on randomness and heuristics when proposing solution alternatives. Therefore, to be able to efficiently solve a given problem the engineering knowledge about the problem tackled must be involved on both ends, on the proposition and on the evolution side of the search process. Such approaches to knowledge formalisation bring computational design support tools a step closer to emulation of process of engineering design synthesis. The explanation of engineering design synthesis as interplay between structure proposal to define design characteristics and then evaluating properties to determine behaviour is given within the Domain Theory [8] and will be used throughout this thesis as a reference model of engineering design synthesis process. Although proven reliable, computational support to product development is left domain limited and oriented to solving of the specific types of engineering problems. Very often the scope of support is realized as a limited mapping of existing method to computational environment thus not offering support to the individual phases of engineering design or to the engineering design process in-whole. Usually it is found that the later stages of product development which are more or less bounded to the specific calculation type or details fine-tuning became more extensively computationally supported. Such computational support is realized by various expert and analysis systems,

feature-based solid-modelling packages. The latter is aimed at components parameterization, preparation of technical documentation and similar type of activities in order to assist designers [1], [9].

Work that is presented within this thesis aims at development of method and computational tool intended for the very beginning of the computational design phase where according to the Theory of Technical Systems (TTS) [1] a technical process needs to be established. Thus, this work is coherent with the efforts made within Computational design synthesis research community (CDS) with its aims set as to provide better search and solution generation possibilities to designers by developing computational support for the early design stages [2]. Introduction of new aids to design process, as in this case a computational tool, should make the process of designing more efficient and in that way increase the possibilities for design of a better product [10]. Development of method will seek out a way to formalise technical process in computationally acceptable manner, as well as trying to define decomposition of technical process using available computational modelling methods and techniques. The result of methods application within computational environment should be generation of variants showing how technical process could be accomplished. These variants will serve as a foundation for further concept development.

1.1 Motivation

Conceptual design is an early product development phase that is most intensive in respect to the implementation of heuristic search strategies and knowledge-based techniques when generating a solution. Designers have a task to generate several unambiguous solution variants or concepts that is based on the initial state of the recognized market and societal needs. Since a concept is defined in terms of working principles on which a designed product will operate, all of the design stages that will follow will be greatly affected after a decision concept was made. As design progresses, some improvements to the designed product could still be made, but changing of the core principle on which the product operates could not be done without creating major set-backs in the product development process.

Introduction of computational support to the conceptual phase of engineering design was predominantly motivated by the fact that it could possibly provide designers with novel or even creative concept alternatives as a result of an efficient solution space search. Generation, evaluation and selection of a concept variant are processes based on designers knowledge,

experience in the field and information retrieved from the external sources [7]. A computational system that is able to process information and generate solution alternatives becomes a vital part of conceptual design therefore affecting both the technical system being designed and the process of designing [1]. Solution alternatives provided as a result of efficient search across the design space should create a foundation on which designers can make well established decisions. Reasonable expectation is that such system that would be initially aimed at the specific areas of application is capable of performing efficient search both in respect to being fast and to usefulness and the number of solutions being created. The possibility to store feasible and useful concept alternatives should be incorporated as well. Generated concept alternatives should serve as valuable starting points to solution development or as a complete conceptual solution to a given design problem statement. A good example of how computers can be used to generate creative solutions is often advocated by the evolutionary computation community [11]. As argued many times before, search done by computers is performed in an objective manner which is not burdened by conventional or prescribed solutions to a given problem. Navigation through the search space in one part relies on randomness, and in the other it is a learning process exhibited by the exchange of solution building-blocks. Exchange of building-blocks is a non-biased since If the information content of building-blocks is of no matter and if only their contributive occurrence in the formation of the solutions is accounted for, than for it could be said that the exchange of building-blocks is a non-biased process. Such objective search is capable of creating novel concept alternatives that would normally require human reasoning that surpasses common engineering practice. Keeping the search space as broad as possible in respect to consideration of unconventional solution principles and the reuse of previous solution building-blocks in order to generate creative concepts is something that systematic approaches to design also try to stimulate.

According to The Theory of Technical Systems [1] which will serve as a theoretical foundation to this research, technical evolution, design and product development are explained as a response to those needs and requirements within human society for which, to be satisfied, an assistance of technical means was necessary. Such teleological view implies as a starting point to a development of a new product concept the definition of technical process as a process of technical system usage in which necessary effects must be delivered by technical product and human beings in order to enable purposeful transformation of operands. Built in the systemic reasoning, TTS models technical processes as transformation systems composed of series of operations interrelated with operand flows and supported by necessary

effects. Changing of state of operands in the desired manner can be accomplished in different ways in respect to various the existent technological principles which prescribe and establish a sequence of necessary operations that must be performed in transformation processes. Technology is defined as a collection of knowledge describing how and with what to perform a transformation in order to achieve a desired state of operands [1]. However, to choose which technological (working) principle to select and how to compose the whole transformation process is an assessment made by designers [7]. Based on designer's in-filed experience, the knowledge of existing technologies (working principles) and on the understanding of the task, a decomposition of technical process is performed in order to gain insights and to reveal details about the transformation process. The result of decomposition performed is conception of information necessary for design of technical system [7].

By reasoning about transformation variants within technical processes designers are in fact considering different duties that human operator and technical system have to fulfil in order for transformation to be possible (Figure 1.1, [1]). The interplay between human operator and technical system, i.e. the product being designed, is necessary to provide effects. Depending on the complexity of the given tasks, the process might be supported by several technical systems and human operators. Design theory states [1], [7] and [12] that technical processes are established as sequences of operations based on different technological (working) principles. Designers than to extent of their knowledge compose technical processes and try to select the most suitable one that satisfies given requirements and constraints. The assumption on which the research presented in this thesis is based on is that if engineering knowledge about technical processes, technological principles and the necessary effects is formalized and embedded within computational tool that when such tool is provided would enable designers to consider optimal product realization possibilities more efficiently.

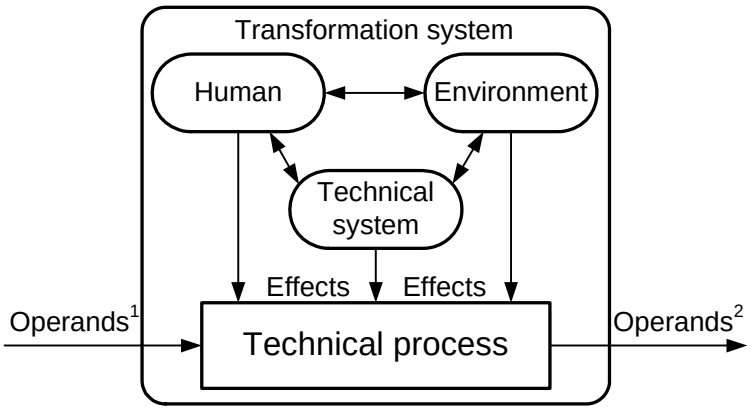


Figure 1.1 General model of transformation system according to TTS

Possibilities of product realisation are therefore understood in respect to the extents of technical system's participation in the transformation. Determination of these duties as an ability to deliver necessary effects thus defines a technical system's function as an entry point by which the organ structure of a technical system will be established. Implications and importance of the consideration of technical processes in respect to whole of the conceptual design was thus one of the prime motivators why to research computational means that could aid designers at that particular stage of design process. Taking into an account how a technical system would participate within technical process is clearly at least equally important to the other design phases and since it is the first stage it may contribute the overall success of design process by the most.

1.2 Aim and objectives of the research

The Design Science considers two main research areas: formulation and verification of models and theories related to designing and process of design; and development of support founded on design models and theories to aid designers in product development process [10], [13]. Research presented within this thesis fits into research group of computational design synthesis, with purpose of development of computational support for establishment of technical processes. For computational implementation it is necessary to consider design automation both syntactically and pragmatically. Syntactical aspects should include adopting one of the existent design process or product models offered within the existent design theories. Prescribed and well-structured design phases with appropriate product models provide foundation and a starting point for development of computational tool. Within this research the focus is set at model of technical processes as defined by TTS [1]. Pragmatic aspects are aimed at devising a method that should be a well suited match both for design theory modelling and for the computational implementation. Most often, development of a method considers application and embedding of the existing computational algorithms and techniques. In case of this research pragmatics related to method definition goes well into how to define formal model of technical processes and process of decomposition of these in a way that is acceptable for computational implementation. The outcome of how well the method will perform, whether results will be purposeful and method could be extended further, depends directly on the applicability of developed formal model.

The aim of this thesis is formulated as to provide a support to the beginning of the conceptual development stage by offering designers the possibility to computationally explore operand

transformation variants in technical processes. To accomplish the postulated aim it is proposed that following objectives have to be reached:

1. To devise a method for generation of operand transformation variants based on different technological (working) principles, and
2. To implement the method for generation of optimal operand transformation variants based on different technological principles as a computational tool built to a completion stage that allows verification of the research results.

According to research methodology in applied sciences to which Design Science is a part of, the mathematical definition of the proposed method can be understood as a theoretical objective of this thesis. Development of computational tool on the basis of the method for generation of optimal operand transformation variants based on different technological principles can be regarded as a practical objective. Most often in the research projects that include a development of computational support to design, the practical objective is a prime motivation behind the existence and realisation of theoretical research objectives [10].

The expected long-term objectives would be aimed at further development and improvement of computational tool for generation of operand transformation variants in development of technical processes and possible integration of the developed method in the existent frameworks for Computational design synthesis and application in industry. It is reasonable to expect problems regarding the integration, since most of the present research are function-behaviour-structure (FBS) oriented [2]. One of the goals is to emphasize the importance of consideration of technical processes which enables to keep the search space as broad as possible by taking into account technological principles and man-machine interaction. By consideration of technical system's function as the top abstraction level with effects as input imposes serious search space constraints if one is to design a complete design synthesis framework.

Research conducted in praxis [13] have clearly shown that engineers seldom use or deviate from as they feel fit methodological approaches as prescribed by design theories. A hypothesis formulated by Jensen [10] probably as a reflection on Herbert Simon's claim [14] that "The Design Science's models and theories had become widely accepted only when there was a need for the introduction of computers as a helping tools for design", states that engineers can be involved in practicing methodological approaches if these are embedded within available computational tools. Attracting an engineer working inside his or her

everyday environment to use a problem solver or design support software package requires more than devising a proper and clever method. The user interface with visual tools to ease the work or cross platform operability is necessary for the acceptance of tool; however, all of these features although being a must, do not add weight to method's scientific worth.

1.3 Hypothesis

An output to the scientific endeavour should be the generation of hypothesis in order to find the explanation of some given phenomena [15]. To achieve that, the research process must evolve both the content of hypothesis and the body of facts for which the hypothesis is supposed to hold. The goal is that the explanation of the matter provided should be more complete and better than the already existing ones.

Hypothesis as a generalization is used to explain the facts established by the observation and measurement or else, the hypothesis is formulated to define non-observable phenomena by which observable phenomena could be explained. Those phenomena are presumed to exist in the world around us, meaning they are of natural origin, and are as such considered as observable entities [16]. However, the Design Science to which this research belongs to, for a research focal point has set phenomena which are not of natural origins [10]. The design process and output of it, technical system designed as is, thus both belong to the body of knowledge that is in literature referred to as The Sciences of the Artificial [14]. As a consequence the task of The Design Science is not only to describe or to explain, but also to prescribe the procedures through which design should be carried out. Put succinctly, design is concerned with how things ought to be, with devising artefacts to attain goals [14]. The same line of reasoning must hold with the understanding of the aim and objectives of this thesis, where the introduction of computational tool in order to improve the product being designed alters the process of designing. One of the focal points of this research is to propose formal models of both technical processes and their decomposition. How to verify these models in a strict sense, like for instance as advocated by the logical-positivism [17], is a daunting task which more or less depends on the acceptance of the computational tool itself.

Thoughts and considerations expressed in the previous sections as well as the research aims and objectives lead to formulation of the following hypothesis:

1. *A set of production rules formalising engineering knowledge about technical processes, technological principles and required effects might be developed for a particular engineering problem domain.*
2. *Using a set of defined production rules a number of variants of operand transformations for a particular design problem may be generated.*
3. *Generated variants may be used as a foundation for further concept development.*

The hypothesis of this work is related to the theoretical aspects of the research. It is directed to the development of the method based on the production rules formalism. A set of condition-action production rules is a mean for achieving knowledge formalisation. In this case the knowledge about technical processes, technological principles and required effects is the knowledge to be formalised. Rules when applied within a production system generate set of valid operand transformations variants achieved through a successive rule application sequence. A set of production rules is therefore understood as a grammar of a language of technical processes consisting of all possible operand transformations variants.

The empirical part of the research is related to computational tool development. Real verification of the purposefulness can only be obtained if the tool is accepted by designers. For that to occur, the long term goals of this research should be accomplished. The method should be integrated into one of the existent computational design synthesis frameworks (like shown in Chapter 3. of this thesis) where its usefulness could be fully verified.

The research questions of this thesis are summarized as follows:

- How to represent technical processes in a computationally acceptable manner? The applicability and usability of the whole method depend on the way how the problem will be represented, thus it is necessary to devise an appropriate technical process formalisation.
- How to generate variants of operand transformation within technical processes? What mathematical concept to use in order to formalise and model decomposition of technical processes, thus rendering it computationally performable? Will this formalised model tend to emulate and resemble by the most the human problem solving activity or it will be drawn to computational problem solving methods like heuristics?

- If possible, how to introduce optimisation methods within decomposition process as optimisation is a must within engineering problem solving?
- Since the proposed method for operand transformation variants will be knowledge-driven than what are the fundaments for engineering knowledge formalisation about technical processes, technological principles and necessary effects?

1.4 Research methodology

This thesis follows descriptive Design Research Methodology or DRM as proposed by Blessing and Chakrabarti [13]. According to DRM, a research work plan is organized within the following four steps:

1. **Analysis** consisting of the literature review to establish the state of the art on development of computational tools for conceptual design support and Computational design synthesis in general. As a result of the review aims, the objectives and purpose of the research will be set and clarified. The literature review will also serve as to determine the scope of the research, i.e. the stage of the design process for which the tool should be implemented and thus possibly enhanced. The aim of the research will be defined as dual since development of computational support tools unequivocally asserts prior development of computational method.
2. **Determination** of theoretical foundations necessary for the definition of the method for generation of operand transformation variants. In order to identify and to select appropriate model, the focus would be set on a consideration of phenomenological models of the design process as offered by the design theories. Research that for an aim have a development of the computational design support most often demand a multi-disciplinary approach. To fully understand the problem and to identify the means of solving it, it is necessary to expand to other disciplines whose scientific knowledge might be helpful to this research effort. Some of the scientific disciplines considered are: formal languages, multi-objective optimisation, systems theory etc.
3. **Synthesis** involving development of the actual method for supporting technical process stage of conceptual design. Based on the established facts as a result of previous research steps, the formal and information models of the method will be defined. This step will conclude with the definition of the architecture of computational tool.

4. **Verification** consists of the results analysis to which extent the proposed method may influence the increase in efficiency and quality of technical systems designed. To accomplish that a computational tool will be developed within the available programming languages and frameworks. An outlook of the applicability of the developed method and computational tool in respect to type of technical system designed will be given. The latter is most important since the determination of technical processes is proposed when developing completely new and/or complex technical systems. Complex technical systems involving multitude of technological (working) principles on which the transformation is based, may prove to be an ideal application framework for this method.

1.5 Expected contribution and results

The state-of the-art-review in research area of the design synthesis as provided within Chapter 2 of this thesis will show that there is no formal model of decomposition of technical processes as provided and described by the Theory of Technical Systems. Formal models are prerequisites for computational implementation. The formal model presented within this thesis will have to describe decomposition of technical processes. The outcome of decomposition should be synthesised operand transformation variants that have to clearly depict the necessary effects that are required to sustain transformation. Moreover, since this method is knowledge driven method, then at least guidelines for the formalisation of knowledge about technical processes, technological principles and necessary effect will have to be provided.

The state-of-the-art Computational design synthesis (CDS) [2]overview that is presented in Chapter 3 of this thesis on provided with the findings that none of the existent methods and tools consider synthesis of technical processes, thus most of these stay limited to supporting of the the early design stages ranging from the establishment of product's functional structure to the determination of components which can realize these functions. While it may be that the approaches to the CDS emerged out of different theoretical backgrounds, the claim that the process level should be included in the reasoning according to the TTS is strongly supported within this thesis. If the function of a technical product is understood as its ability to deliver the effects necessary for supporting of operand transformation, than it is clear that non consideration of technical processes will impose limitations to the design search space, thus inducing the loss of potential solutions. Further methods such as the morphological chart

method can add new effects to the established functional structure. The only other way to accomplish this is to affect the technical process inside, where the main operand transformation is realized. In order to provide solid foundations on which designers can make well established decisions as a result of efficient search across the design space a technical process must be included. Although the search for innovative technologies for operand transformation is usually considered only for the design of completely new products and not for redesign tasks, the intention is to provide the basis on which the functional structure of the product can be determined computationally.

The expected contributions and results of this thesis are given as follows:

- Formal model of technical process. With the introduction of a technical process level a contribution will be achieved to the research area of Computational design synthesis which was until now only focused on function-to-component method and tool development.
- Formal models describing decomposition of technical processes. The outcome of decomposition should be synthesised operand transformation variants describing the necessary effects that are required to sustain transformation.
- By implementation of defined method within computational tool it is possible to efficiently and fast generate variants of operand transformations, thus presenting a foundation which may further excel the concept development within a product development process.

1.6 Thesis outline

The Chapter on **engineering design synthesis** will provide a viewpoint to design as a problem solving process in order to determine principles which can be utilized to develop a computational support to design process. Many of activities performed at everyday problem solving are already being prescribed by the design methodology and it is necessary to identify these and determine whether it is possible to transfer them to computational environments. The the early design stages where the synthesis is most intensive will be considered in general with the generation of operand transformation variants put in focus as a subject of this thesis. Formal models of design synthesis will be examined as a prerequisite for the development of computational tool. Finally, theoretical model of technical processes as well as reasoning

about design synthesis and designing will be adopted according The Theory of Technical Systems (TTS) [1] providing understanding what a synthesis of technical processes is.

The third Chapter of this thesis will present the state-of-the art of the research efforts in **computational design synthesis (CDS)** [2]. Present computational methods and tools will be analyzed both in respect to their theoretical foundations, and mechanisms they apply to perform at different Computational design synthesis stages which are determined according to the generic CDS framework [2]. The findings provided will help to establish the principle mechanisms of the method for operand transformation variants and to compare its scope and theoretical foundations to the other existing methods and approaches. As a result, a contribution to the field of CDS can be determined.

Formal grammars will be presented in the fourth Chapter both as a generic knowledge formalisation approach in the field of AI and as a mean to formalise engineering knowledge about technical processes as required by this thesis. Rather than just accepting those as a popular approach in the CDS today, it is presented how formal grammars came to be and how they were used in studies of cognitive processes to develop robust problem solving systems, thus consequently relating them to engineering problem solving. Formal definitions of generative grammars and formal languages will be given, as well as categorisation of grammars according to Chomsky's hierarchy [19]. An example will be given which emphasizes the difference between sequential depth first and breadth first production rule application sequences of string grammars and implication of these when applied to graph grammars rewriting procedures. Graph grammars will be used throughout this thesis to model decomposition of technical processes.

The fifth Chapter will give an outlook of **grammatical evolution** [3] which is a population based stochastic optimiser that will be used to provide optimal decomposition of technical processes. Engineering design synthesis will be compared to the findings of the evolutionary computation community in respect to the paradigm of evolutionary design to emphasize similarities between an explanatory search and creation of solution within evolutionary algorithm. The evolutionary operators based model of genetic algorithm will be presented, as well as a description by which grammatical evolution performs the search. Additional mapping functions will be defined that take into account the requirements for technical process modelling, thus extending the generic evaluation procedure of GA model to tackle

both the GE and graph grammars. To clarify, an example of breadth-first rewriting will be shown using the same grammar as in the previous Chapter.

The method for generation of operand transformation variants will be presented in the sixth Chapter of this thesis. The needed knowledge about technological principles on which the transformations are based is formalized within set of production rules in Backus-Naur form (BNF) [20]. Since The Theory of Technical Systems models technical processes as operand transformation processes, thus the proposed method for generation of operand transformation variants will be designed as a formal system based on graph grammar transformations that should perform synthesis of technical processes throughout a series of consecutive decomposition steps. It is proposed to base method on established concept of node rewriting. For modelling of operands transformation inside technical processes a directed multi-digraph with operands effects and operations is designed.

Knowledge formalisation and examples will contain examples of technical processes concerning the participation of technical systems which differ in their complexity. Since the aim of this thesis is not to develop ontology of technical processes and technological principles , an adaptation of standard taxonomy intended for product functions [21], [22] will be utilized in order to define production rules with the least possible ambiguity. Moreover, suggestions will be given how to formalise knowledge about technical processes.

The eighth Chapter of this thesis will elaborate in brief the prospects of the **computational implementation**. **The** model of threepart architecture of computational tool, realised by object-oriented programming and interconnected by a relational database, with one designed to visually model rules and the other to process them will be presented. Also, diagrams of principal classes will be elaborated concerning graph transformations, formal languages and of larger multi objective genetic algorithm optimisation framework that supports grammatical evolution as well.

The conclusion and the prospects of the future work will close this thesis.

2. ENGINEERING DESIGN SYNTHESIS

Nothing is more important than to see the sources of invention which are, in my opinion, more interesting than the inventions themselves (Gottfried Wilhelm Leibnitz, Art of Invention, in his unfinished work [23]*).*

Design is a backbone of every product development process, meaning that demand for innovative designs is expressed as the need to excel technical development which "naturally" should occur evolutionary. From the literature it is known that almost 60% of total costs of product development are derived as a result of decisions made at the early stages of design [1]. Moreover, up to 80% of the entire product malfunction cases which occur over in-service time are related to the early design phases [4]. Removing possible errors from the early stages might also be achieved if design space is being thoroughly checked and evaluated against the requirements. Therefore, there is an ever growing demand for understanding of design as a problem solving process and of synthesis as a design's processes key element. Exploration of the process of design, development of new methods and tools, altogether provide designers with means to efficiently search and generate the most feasible or even innovative concepts that may lead to a new product being put on the market. Openness do different solutions and suggestions of how to tackle the problem are being stimulated at brainstorming and 6-3-5 sessions, or by using different methods like Delphi method for instance [1], [4]. What is hopped for is that the solution synthesis will be performed at the state of deep understanding of the task producing a few alternatives to choose from as a result of methodical search across the design space. In respect to computational tools these findings about design process, engineering design synthesis and the principles of how do creative solutions emerge provide fundamentals on which computational support could be developed thus excelling the design process even further.

This Chapter will present an outlook on principle problem solving methods and approaches frequently used in engineering, like for instance generating and testing, decomposition and heuristics. For all of the methods and approaches it will be found that they are carefully integrated into many of the existing design methodologies. Moreover, the models of engineering design synthesis according to the Domain Theory [8] and the Theory of Technical Systems [1] will be presented. Insights that will be provided will prove helpful for design of

computational method and tool for generation of operand transformation variants in technical processes. The related questions that will be addressed is what parts of these problem solving approaches and design methodology regarding the synthesis of technical processes are suitable for computer implementation, can they be translated completely to computer environment, if not than to what extent and what are the means to accomplish that?

2.1 Design as a problem solving process

The fundamental principle on which design rests is generating and proposing alternatives until they fulfil a set of given requirements and constraints [14]. Taking a guess about the possible solution to a given problem and navigating through design space in order to find a sufficient solution is a valid strategy when dealing with design problems. Depending on the complexity of a task, design search spaces will most often end up as being vast, constrained, multimodal and full of discontinuities, frequently requiring a heuristic based strategy to be efficiently explored. Consequently, by making assessments about the feasibility of the individual solutions alternatives at different levels of abstraction, the designer will in a stepwise manner progress from abstract to concrete. Each of these assessments is therefore a necessity and each one will inevitably reduce a portion of the available search area. Building on the latter the design process can be understood and modelled as a tree structured state-space search process (Figure 2.1). Navigating through design search space and testing possible solutions follows the established line of reasoning about how to solve a problem.

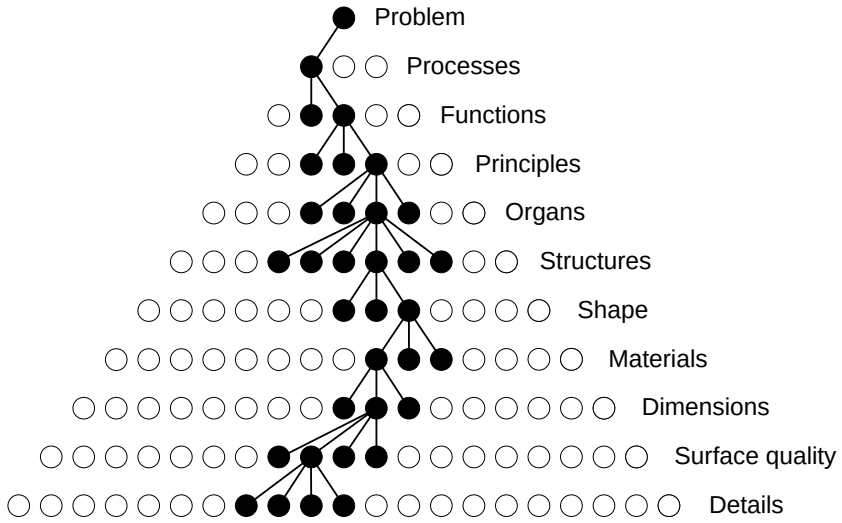


Figure 2.1 Causality in design's degrees of freedom [8], [24]

However, at one arbitrary point in design process the idea that looked promising at the beginning might prove to be exhausted as designer realizes that if going further following that

path the product designed will never be able to successfully meet the given requirements and constraints. Then, designer has to turn back and to consider the other available options. Reasons for being that so can be argued since as design progresses the environment of design process changes as well as the understanding of the task and object being designed thus preventing envisioning of the course of the search. As once said for the artists that in creating the work of art, the artist "evolves" his or her own brain to a new stage [25], in the same manner the designer evolves understanding and viewpoint of the task and the object being designed. Design stalls and stops, or even going a few phases back in design process are the issues that are also tried to be addressed by the introduction of computational design support tools (elaborated within Chapter 3.).

Solving a design problem by using a random walk or a TABU search could prove to be a reasonable strategy; however there is more involved, and it precedes the straightforward appliance of a heuristic search algorithm. When dealing with complex problems like design, a direct approach with immediate mapping from question to answer usually won't work. If ever obtained, the results usually end up being of a poor quality. Research in the field of human problem solving and cognitive psychology have shown [26] and [27] that every human being will more or less in a similar manner react when facing a problem situation. Usually, a complex top-down reasoning process involving the expectation of the outcome will govern the search. However, if there is a possibility to introduce a methodological approach prescribing how to deal with a certain class of problem which is of course an art and science for itself, then there is a chance that the results will turn up better as an outcome of more efficient problem solving process. The role of design theories [1], [4], [5], [6] and [8] can here be reemphasized by explaining their task as to offer descriptions of intermediate steps with corresponding activities and to prescribe methods that might help guide designer to a successful design.

A generic methodology for problem solving including design problems can be summarized in a few generic steps put inside a loop: understanding of the problem that initiates the solution process, devising a search plan or problem analysis, applying solution strategy and reflecting back to see whether the results meet the requirements [28]. The ability to ask and formulate the right question that will be able to describe most of the problem aspects is as equal in its importance as the solution itself. Problem definition is a statement of its meaning on other terms which are supposed to be well known to a person. Understanding of a design task is an internal representation of the problem including person's initial beliefs and analysis of the

given problem, its constituent elements and the representation of the goal. Search strategies which are employed and no matter how clever they might be depend on the initial input and the goal of the search. Problem analysis involves the application of various techniques in order to gain insights on the problem. Framing or relating problem to similar situations encountered helps restating problem to identify all of the aspects that need to be addressed. As once recognized by Herb Simon [14], decomposition of the problem to a smaller chunks of a size that still maintains and tracks the relations to the system but permits independent consideration of each of the elements is the norm for the engineering design. The same reasoning was adopted by most of the methodological design approaches and design theories like TTS [1], Systematic Design [4] and Domain Theory [8]. Building on the foundations of the General Systems Theory [29], thus embracing a holistic reasoning, within most of design theories [1], [4], [8] and it was adopted to model a product as a transformation system the early design stagesat the the early design stages. Eloping to abstract enables designers to focus on the architecture of the system, rather than dealing with the details. However, there are degrees and alternatives to decomposition. As design process can take different paths, so can the decomposition process. Alternatives of decomposition are to be understood in the sense that there are different ways by which the designer can define the scope and assign duties among the system elements. The assignment of duties defines the system in respect to the elements which is in fact a bottom-up solution process. The latter is a synthetic activity under inductive inference thus clearly showing that decomposing is not always performed and understood in strict reductionist sense, but it may be quite the opposite. Since numerous alternatives can appear as a result of different viewpoints on what an element is and how it should perform, the introduction of computational support might prove valuable to designer.

The study of human cognition aimed at finding out how creative ideas emerge shows a pattern in problem solving processes [26], [27]. Through analysis of artwork and engineering achievements, it was shown that creative ideas were conceived as a result of an ordinary and deliberate process. Intensive problem solving techniques based on the knowledge from a decade long collection of expertise in the field as well as other domains were applied. The same applies to the engineering design it is; solving involves both heuristic search with knowledge intensive methods like analogical transfer and in-the-filed expertise. Heuristics in general tries to find a solution candidate that will at least achieve the given goal without considering the utility of the solution. When performing a design task, designers most often recombine chunks of past solutions into a novel solution. In fact, this is the norm [1]. Past

solutions are often mental models and to patch them together requires an abstract search process. Inductive and deductive reasoning and the resulting expectations of the product's behaviour based on a designer's knowledge and experience are involved in such search [30]. Knowledge help constrain the search area; however, when heuristics progress stalls with no answer found, a work around by establishing analogies might prove fruitful. Analogous objects are these that match to a certain extent by the way in which their constitutive elements relate [28]. On foundations of expertise and external sources like text books or reports, designers can transfer solution or at least a work principle that might solve the problem. Recognizing analogy, i.e. inference by analogy, to generate solutions is most often present inside a design process. The awareness of the new solution possibility extends the borders of design space thus enabling further heuristic search. Possibilities that arise by patching-up of new and old building-blocks in a novel way might lead to a problem solution. Only recently the advances in the field of computational cognitive linguistics presented a conceptual blending method which to an extent enables a knowledge-intensive reasoning by establishing analogies applied to general and common problems [31].

Abstract reasoning and specialization or detailing, are two more problems solving techniques that are most often present within a design process. Ability to use symbolic systems like natural languages and to establish hierarchical class structures are the key element of cognition that allows abstract reasoning [32]. Abstraction enables to focus on the important but general, to disregard all of the features that are at the moment irrelevant or too distant to be accounted for. A systemic reasoning about the world surrounding us is an archetype of abstract representation. The same approach was adopted by design theories to manage with the complexity of product development the early design stagesat the the early design stages. In design moving from abstract to concrete is understood as process of the assignment of new attributes to the product under the consideration [8], [33]. If framing the same process to optimization problems then it can be understood as to put the design objectives in order by the degree of importance and acting accordingly when searching for a solution. Specification or detailing is understood as assignment of values to attributes [33]. From the perspective of design methodology, design stages and phases are ordered in a way that they maintain the introduction of design characteristics gradually at the points in evolution when design solutions are evolved enough to allow proper reasoning. Therefore, detailing requires concretization of an attribute first.

Based on the design theory and methodology literature review Sim and Duffy [33] tried to identify a generic set of design activities. The outcome of their research efforts resulted with an overview of design activities per design process model as shown here in Table 2.1. As defined by [33] generic activities performed in design process are subdivided in three categories: design definition activities aimed predominantly at problem restructuring and managing problem solving complexity, design evaluation activities concerned with the testing of solutions and reduction of search space by identification of unfeasible candidates and design management activities with the purpose of coordinating the previous two set of activities. In fact, the whole of design process can be modelled as a process of search done by a specific method and involving different reasoning principles and solution representations. To conclude; in respect to synthesis a similar viewpoint is adopted as presented by the GDT equating it with the explanatory search process of design.

Table 2.1 Overview of design activities

Design activity	Hubka (1982)	Pahl & Beitz (1996)	Pugh (1991)	Suh (1990)	Ullman (1992)	Ulrich & Eppinger (1995)
Design definition activities (function to form/structure)						
Abstracting	✓	✓		✓	✓	
Associating	✓	✓	✓			
Composing		✓			✓	
Decomposing	✓	✓		✓	✓	✓
Defining	✓	✓	✓			✓
Detailing	✓					✓
Generating	✓	✓	✓	✓	✓	✓
Standardising	✓	✓	✓			✓
Structuring/integrating	✓					✓
Synthesising	✓	✓	✓	✓	✓	✓
Design evaluation activities (form/structure to behaviour/effects)						
Analysing	✓	✓	✓	✓	✓	✓
Decision making	✓	✓	✓	✓		✓
Evaluating	✓	✓	✓	✓	✓	✓
Modelling	✓	✓	✓		✓	
Selecting	✓	✓			✓	✓
Simulating	✓				✓	
Testing/experimenting	✓	✓	✓			✓
Design management activities						
Constraining		✓				
Exploring						✓
Idnetifying	✓	✓	✓	✓		
Information gathering	✓	✓	✓		✓	✓
Planning					✓	
Prioritising						
Resolving						
Searching	✓	✓			✓	✓
Selecting	✓	✓	✓		✓	✓
Scheduling		✓	✓			

Based on this section and on the presented analysis of general problem solving methods, the principle mechanisms that drive an engineering problem solving process can be summarized in:

- Problem analysis and variation,
- Generating and testing of solutions,
- Abstracting and specifying,
- Decomposing to realize and define elements of product as a transformation system,
- Deciding upon a solution variant, and
- Heuristics and knowledge intensive search methods.

2.2 Design as an explanatory search

The study of a creative process and how creative ideas emerge requires an analysis directed both to the problem being solved and to the methods and techniques being applied for solving of that particular problem. Many research efforts in the past and present were directed towards the understanding what a creative process is and how to stimulate and simulate it. Our world is shaped by the activities of creative people [26] where design and designing is the mean through which that shaping was achieved. Therefore, with no doubt, a creative process is most often taken as the subject of analysis of how solutions to a problem emerge. Design can be understood as performed in a cycle that interlopes analysis, synthesis and evaluation. Each of these phases is equal in its importance for solution generation; however in respect to reasoning processes performed they differ a lot. By all means, the synthesis as a key creative step in which by combining and mixing of ideas new solutions emerge is a focal point of interest. The previous sections addressed the design as a problem solving process from the methodological point of view leaving the classification of problems and reasoning processes that are being applied to be addressed in the forthcoming text.

Design practice shows that almost often design problems go beyond being ill-defined. The initial problem statement is incomplete, the goal is unknown thus recursively affecting the input. Because of the uncertainty on both ends, the literature categorizes design problems being as even more elusive than wicked problems [33]. Complex problem solving processes as design is, exhibit activities that are of iterative and evolutionary nature guided by different logical principles. Finding out the logical reasoning process was a prerequisite in attempts that were made in order to create computational simulations of human reasoning processes as test beds for determination of creative solution emergence [26]. Development of the computational design tools aimed for design synthesis is no exception. The necessity to

investigate logical principles of reasoning processes is driven by the assertion that all formalised processes can be rendered computational. If the formal models could be devised then there is a chance to simulate the engineering design synthesis by computational systems. In respect to reasoning performed the research has shown [23] that solely deductive models do not suffice. Deductive reasoning process is by definition as good as the initial premises are. If performed correctly, the outcome must hold, since the truth of inference is determined by the truth of the premises, assuming a valid inference process [15]. The goal of design is understood as the satisfaction of the existent market and societal means that can be achieved using the technical system designed, then using deduction a direct mapping from the list of requirements and needs to design solution should be possible. However, due to incompletes of premises and the evolutionary nature of the design process, the appropriateness for design synthesis to be modelled as deduction was refuted [30]. Deduction is a one-way process that cannot provide necessary means required for solving of design synthesis problems.

Assigning deduction to analysis and evaluation seems as a reasonable step. Deductive inference applied to conclude some facts about design situation assumes a valid outcome as a result of true premises. However, the uncertainty that comes with the absence of absolute knowledge is present both in the formulation of the design requirements and the object designed. Therefore an extension to the reasoning model is necessary calling for addition of an inductive reasoning part. Induction can handle situations of incomplete knowledge including beliefs and expectations of the person thus mimicking top-down reasoning processes. Examples of such are classifier computational systems or concept-learners like PI [15]. The valid results of induction include occurrences where although the inference process itself was consistent, the outcome must be considered with a degree of uncertainty and not as an absolute truth.

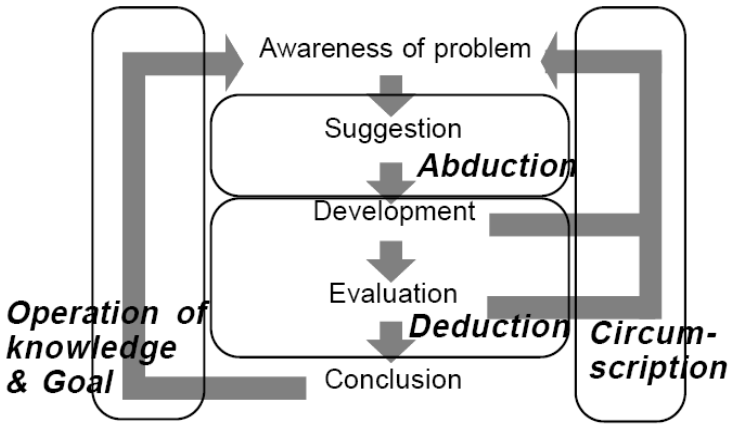


Figure 2.2 Design as an explanatory search according to GDT [30]

The expansion of the deductive model of design synthesis to include both uncertainties of inference results and iterative nature of design process was accomplished with the formulation of The General Design Theory (GDT) [30]. GDT modelled designing as a process of knowledge manipulation based on the axiomatic set theory. Axiomatic foundation offered a possibility to derive theorems and to develop a formal model of synthesis process within intention to run it computationally. Although, the initial assumption of the GDT that the design knowledge can be organized as a topological structure proved too idealistic [18], the contribution of mathematical modelling of design synthesis was valuable. GDT presented deductive-abductive model of a design process (Figure 2.2).

In respect to the problem solving strategy that is applied for generating solutions to engineering design problems, it can be said that designing, and in particular the early design stages where the need for generation of novel solutions is most emphasized, are performed as an explanatory search. An abductive reasoning is a logical principle behind the explanatory search undertaken by designer which results in the creation of a technical description of the designed product as an explanation to the recognized market and societal needs. The solution to a design problem is achieved through an iterative process which assumes co-evolution of the object being designed and designer's knowledge and understanding of the problem he or she is facing, i.e. the problem and solution are being redefined iteratively in relation to each other. Explanatory search and co-evolution result in a solution which can be systematically evaluated, but because of causal relationship that exists between a problem understanding and its solution it cannot be told whether the generated solution is a final and best answer to the existing requirements and needs. Moreover, the implication that with a designed product the existent market need will be fulfilled is a premise in abductive reasoning. Put differently, there may be many solutions to the given design tasks. Deductive part of model is of course linked to analysis and evaluation parts. More recent advances in understanding of design synthesis and its modelling offered an extension to abductive model replacing it with creative abduction or innoduction [23]. The Assertion is that abduction alone cannot produce creative solutions since a solution proposition is already included in the premise as a part of explanatory search. In respect to that, abduction can yield only a design the principles of which are already known. Therefore, it is proposed to include innoduction to synthesis modelling, and that inference search should result both in the explanation and the design specification. However, at the moment, design support tools do not go beyond deductive and inductive reasoning models.

2.3 Engineering design synthesis according to the Domain Theory

According to Hansen and Andreasen [8] the explanation of engineering design synthesis should at least address the following three points: understanding of synthesis activities, synthesis as cognitive activity performed by humans and object of synthesis, designed artefact as is. Theoretical model of synthesis should try to unify these three points, what is in contrast to most of the design theories which offer only partial descriptions necessary to elaborate their viewpoints to engineering design. The Domain Theory (DT) [8] which served as a foundation for synthesis modelling [8] was proposed by Andreasen. Built on The Theory of Technical Systems (TTS) [1], The Domain Theory adopts viewpoints to the product being designed consisting of transformation, organ and part views (Figure 2.3).

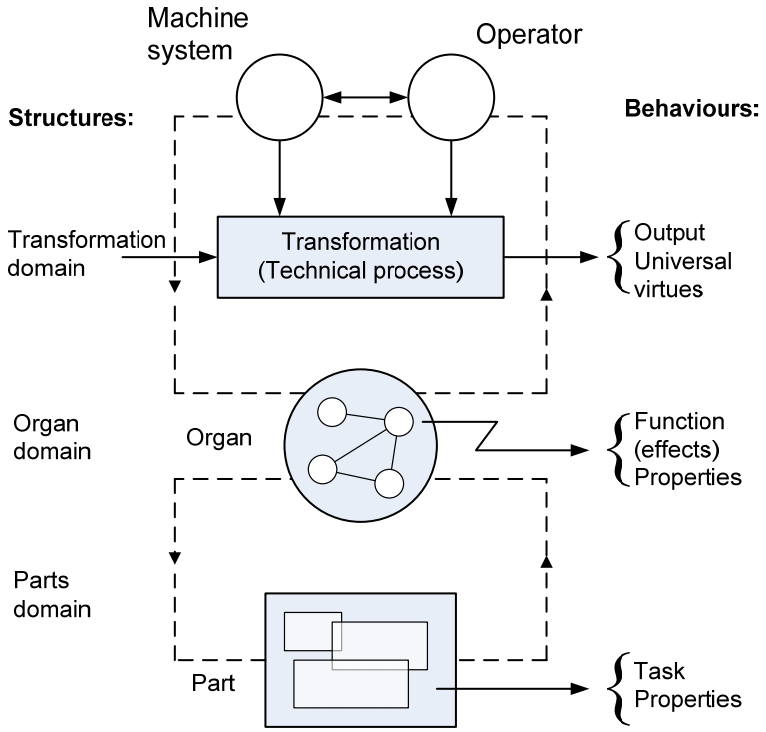


Figure 2.3 Domain Theory applied to design object [8]

The core principle of designing as advocated by the DT is compliant to the fundamental principle on which design rests as to generate and propose solution alternatives until they fulfil a set of given requirements and constraints. Generation and test [14] of alternatives is a process where the DT recognizes two important aspects; what has been accomplished as a proposition of structural characteristics of the product being designed, and what is tried to be accomplished as behavioural properties exhibited as a result of the proposed characteristics. Such generation and test algorithm is performed at all three domains (see Figure 2.3). Applying the DT to the object being designed results in designer examining behaviour in

respect to object's structural realisation in each of domains. As a consequence, decisions made in one domain affect other domains.

The transformation domain considers the purpose of technical product as to support transformation of operands by delivering necessary effects, the organ domain considers structures and modes of actions which create and deliver necessary effects, and the part domain resolves how to distribute organs into machine parts. At this point, it is important to stress out that among all of the design theories only TTS [1], TRIZ [5], and DT [8] realize the importance of transformation performed within technical processes as being crucial for design of technical products. It is of an interest to note that product function as a domain per se, has been omitted from the consideration. Explanation for the latter can be drawn from the assertions made by The Systems Theory [29], where a function of an observed system is defined as a property of that particular system. Building further, if the function of the product is to deliver necessary effects, than a product function is a class of behaviour realised as an output of the organ domain. Put succinctly, the function of organs is to deliver effects.

According to the Theory of Domains two approaches to modelling of engineering design synthesis are understood in a sense of function-means law [14] relating to two design situations:

- **Design-process-oriented synthesis** referring to design of a completely new product. Generation and test algorithm performed over all three domains thus synthesising a new product design.
- **Artefact-oriented synthesis** which is based on the reuse of the past design processes where the relationship between product characteristics and required properties is at least partly known.

DT realises that a frequent activity inside process oriented synthesis is a decomposition carried out for two reasons; to be able to reduce the overall complexity of the design task by dividing it into more manageable chunks, and secondly to assign these chunks to designers allowing them to work in parallel. The authors argue that there is no definition what decomposition really is [8]. This thesis however, as explained before, understood decomposition from a systems' theory viewpoint as a synthetic activity. Proposing of structural characteristics, testing and navigating between the three domains is the explanation of the synthesis under DT. After the identification of the purpose of the technical product, following generic synthesis steps are proposed according to the function-means law:

- **Detailing**: select means to realise behaviour.
- **Concretisation**: assign values to means.
- **Decision-making**: select best means.
- **Composition**: compose selected means.
- **Decomposition**: identify support for selected means.

The identification of support is performed according to Hubka's Law which states that the means synthesised for solving of the required function are seldom self-sufficient thus most often accompanied by auxiliary functions realised by additional means.

Artefact oriented synthesis allows the implementation of heuristic computational methods. It relies on re-usage of past design solutions in order to create new ones. Past solutions may be stored inside design catalogues or a specific design task methodology aimed at solving the given class of problem may be applied. As DT claims, utilization of the past design is a powerful method offering high optimization potentials and generating promising design synthesis solutions. However, the way in which reuse of past solution has been framed is somewhat in contrast to the explanation of how creative solutions emerge. Combination of heuristics, top-down reasoning as a result of past experiences combined with the knowledge intensive methods relies on the past solutions. Reuse of building-blocks is an invariant approach applicable to any-kind of problem solving, thus being difficult to accept its presence, being emphasized only inside artefact-oriented synthesis.

2.4 Synthesis of technical processes

Following sections will provide an approach to engineering design synthesis as defined according the Theory of Technical Systems. The teleological viewpoint which is undertaken within TTS that sets synthesis of technical processes as an initial stage of conceptual design is theoretically fundamental to this thesis, thus it is necessary to elaborate it. Two types of synthesis which are recognized by the TTS will be presented as well as the models of technical processes and technical process decomposition. All of these are required to emphasize the importance of technical process synthesis, the relation between technical processes and the function of a technical system that needs to be designed and to provide the insights needed for design of the method for creation of operand transformation variants.

2.4.1 The Theory of Technical Systems

The Theory of Technical Systems (TTS) is concerned with studying technical systems as artefacts that are of technical or engineering content [1]. Based on the legacy from the systems theory and cybernetics, the TTS argues that it is necessary to understand both the behaviour of technical system and the methods and processes which conceived and created that technical system. The TTS builds its reasoning by bringing together how technical systems came to be in the first place what is necessary to understand when designer needs to establish duties of the technical system under development. Such reasoning is deeper and in contrast to the most of the other design theories which tend to accept technical products as a fact per se, it offers the possibility to recognize more of the important aspects necessary to design successfully. Technological progress achieved throughout centuries of innovation, creation and continuous improvement was pushed by societal needs that demanded assistance of technical systems in order to make possible and to alleviate attaining of the existent purposeful goals. Understanding the motivation for development of technical systems as tools being required by the human society is observed throughout the history and in a way it equates the evolution of mankind with technical evolution. The history of engineering, presents the evolution of artefacts driven by teleological principles, and is therefore an inherent component of the civilization and human society. To study and understand the history of mankind, one must study and understand the history of design [14].

Like any other problem solving, the fulfilment of the existent societal needs and requirements can be framed as a transformation process inside which the initial unsatisfactory state is transformed through a series of operations into a presumably desired state. In different domains involving designing in a broad sense, a transformation through series of operations is managed differently but understood as the same. For instance, a straightforward computational sciences example of a programming function that ought to be designed in order to transform the input variables into the desired output. In the literature one's own thoughts and impressions are transformed into a written text using grammar of a language to make it understandable and interpretable for the reader. The examples are numerous and are all look alike. Therefore, in mechanical engineering and design a transformation is always perceived as being performed inside an artificial process in which an object is transformed intentionally and with the purpose. The object that is undergoing the transformation is regarded as an operand; a passive member of the whole process that exists in the world around us and which exhibits both structural and behavioural changes. How to achieve a desired sate of operands

that may fit the existent societal and market needs is put as a central question and it is most often answered in terms of which technology to apply. Technology is a collection of knowledge describing how and with what to perform a transformation in order to achieve a desired state of operands. Technologies differ in principles by which the change of operand's state has been performed, i.e. technological principles based on physical laws by which the transformation sequences with their requirements are being derived from. In terms of the Theory of Technical Systems, applying a technology is always considered together with the assistance of supporting technical systems. Human operator and technical system compose an execution part of the transformation system which by interaction with the environment provides all the means necessary for transformation to be possible. According to TTS, these means are denoted as effects which include actions exerted onto operands by technical system, human operator and environment, auxiliary operand's flow with regulation and control [12]. A general model of the transformation process [1] is presented in Figure 2.4 as an extension of Figure 1.1 in Chapter 1.

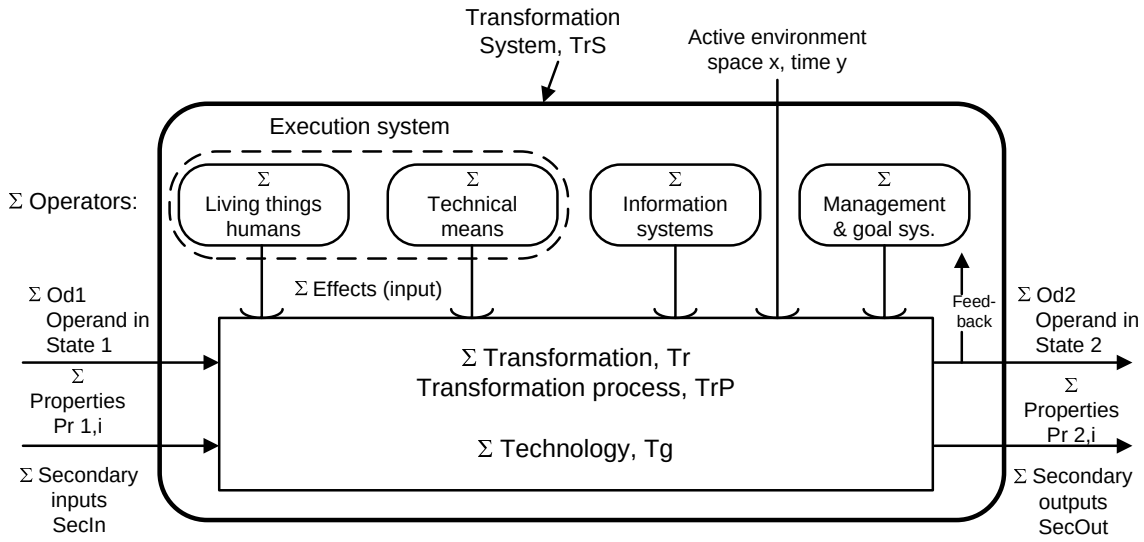


Figure 2.4 General model of transformation process [1]

Following Figure 2.4 a technical process is defined as an artificial process in which the state of an operand is transformed intentionally under the influence of effects delivered from a technical system, a human operator and the environment which are collectively referred to as transformation system's operators. In the context of the existent societal needs and requirements, it can be said that the fulfilment of these may be achieved within a technical process as a transformation processes in which technical systems are in extensive use [1].

According to the TTS, determination of the operand transformation process as a sequence of operations and operand flows in respect to different technological principles and corresponding necessary effects to drive the transformation is referred to as the establishment of technical processes and is in a fact the first step which is proposed to occur when moving into development of a new product. Technical systems are objects of design and since technical processes may be based on different technological principles, then the duties and roles of technical systems in the transformation will vary accordingly. These duties and roles must be assigned optimally alongside technical process distributing them among all of the parties inside transformation system [12]. This is an important aspect since designer must establish the expected behaviour, i.e. a function framed as the collection of effects, of a technical system that is about to be designed. Interplay between human operator, technical system and the environment must be envisaged in a way that all of the parties inside a transformation system are capable of producing effects necessary for supporting the transformation of operands. Such systemic approach imposes early design stage considerations of how the technical system is going to be used once in service. The most suitable technologies for operations which are intended to be driven by effects from the technical system must be in compliance with the existent market and societal needs. As a consequence, various product realization possibilities can emerge as a result of considering technical processes variants. Search for optimal or at least suitable near-optimal technical process is precisely that what is advocated by the TTS and TRIZ for instance [5], as being necessary step, if one aims at designing of truly new and innovative products.

2.4.2 Decomposition of technical processes

Following the systems theory approach, systems can be modelled as the collection of elements put into a connection inside system's boundary. Connections or connecting elements, most often arrows rather than solely edges are used to introduce the order into the structure thus making it interpretable in different ways by which system elements can be related one to another. In the same manner, the TTS approaches the modelling of technical processes (Figure 2.5).

Elements of technical processes are operations put into a mutual relationship by operand flows which connect between the outputs of one operation to the inputs of subsequent operations. Such structure clearly depicts the operand state transition exhibited by operands

during the transformation process. Building on the latter implies that operations can be in a sequence or parallel.

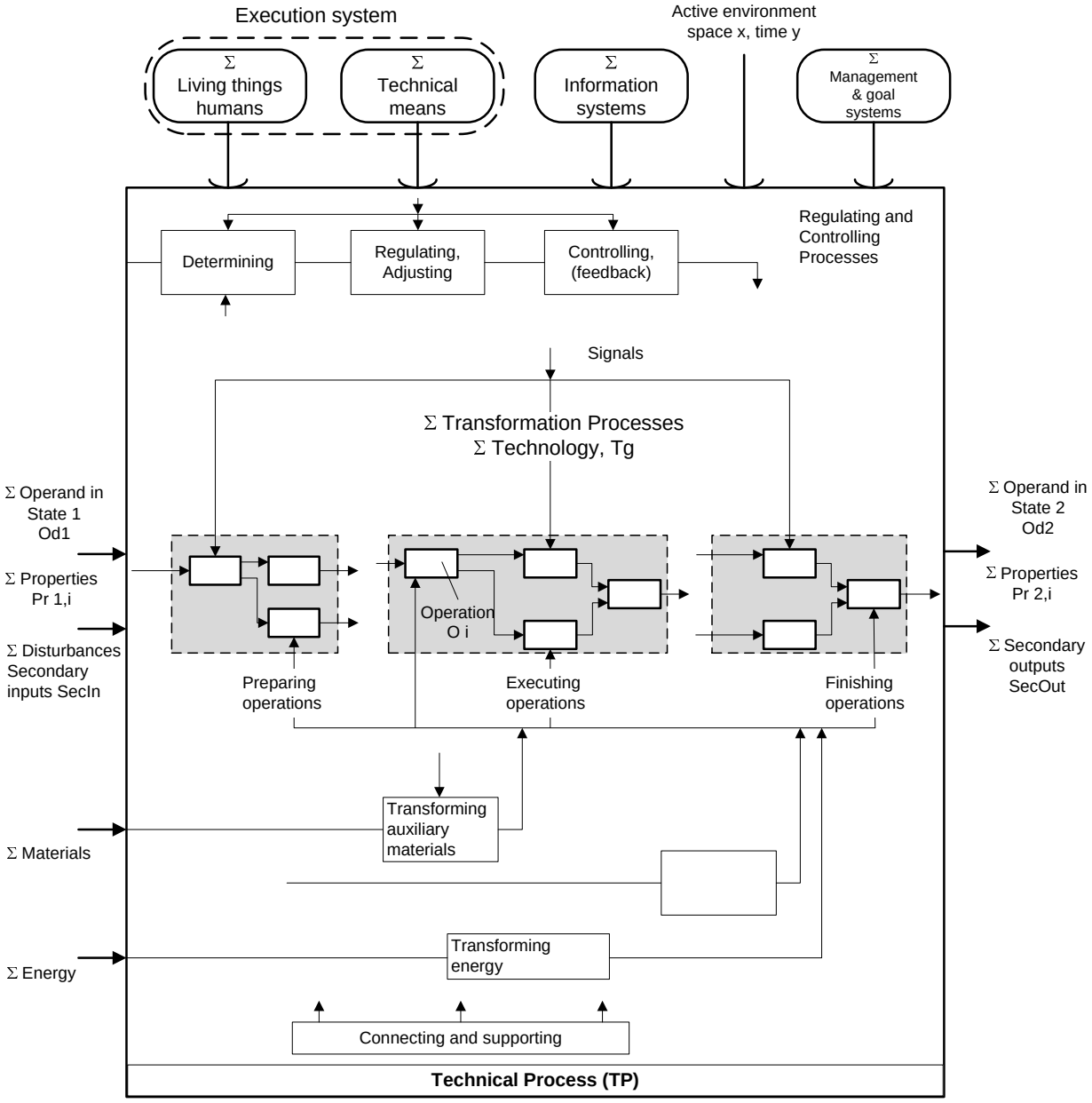


Figure 2.5 Structure of technical process according to TTS [1]

The effects delivered from the execution part of the transformation system and the environment, are modelled as arrows directing from operators to operations. Flows emerging from an output can be multiple as well as the inputs. The task of designer is to structure on the bases of the selected technological principles or a set of different technological principles to decompose technical process in such a way that it would yield in transformation of operands that hopefully corresponds to the existing needs and requirements. If more than one alternative is created the most suitable one is to be selected.

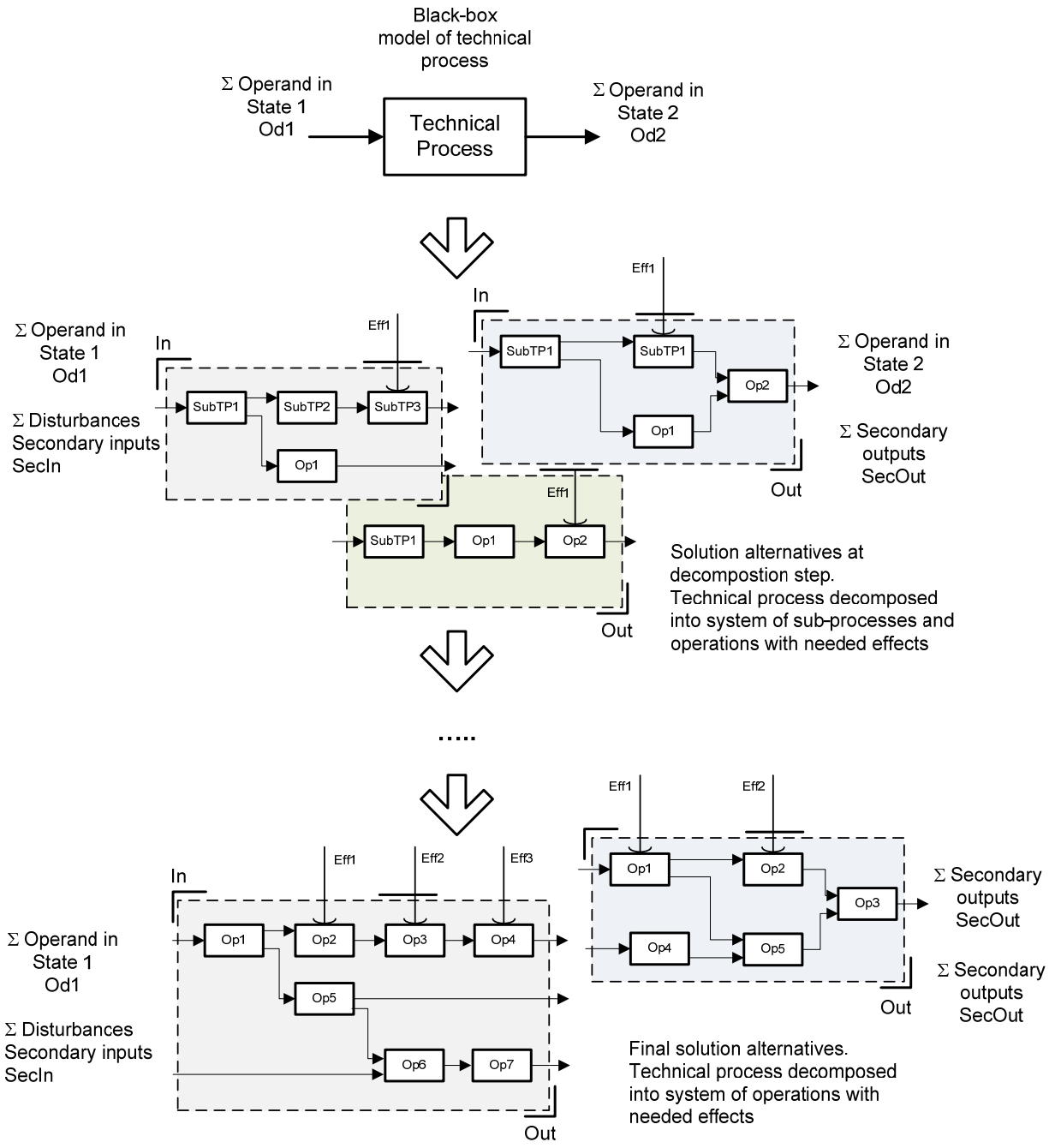


Figure 2.6 Decomposition of technical process

The TTS proposes a methodological approach, since the establishment of technical process is a problem for itself. Designer starts from a single operation creating a black-box type of problem representing the existing needs as transformation and then gradually decomposing it (Figure 2.6). As decomposition progresses towards the problem tearing down to its constitutive parts, it becomes easier for designer to grasp all of the aspects and to better understand the nature of the problem at hand. High-level decomposable operation is denoted as the process, which can be decomposed in a transformation sub-system composed of sub-processes and operations interconnected with operand flows. Decomposition process is

repeated again and again resulting in the growing number of simpler elements interconnected with flows inside fixed systems boundaries. The whole process lasts until designer gains sufficient insights about the duties of technical product that is ought to be designed and the secondary operands flows that emerge as a consequence of technology being applied. Since the definition of technical processes structure involves selection of operations, relating them with operand flows and adding the corresponding effects and since it occurs at every level of decomposition process thus creating iteration in which designer fine tunes at all of the decomposition levels, it is more reasonable to address it as synthesis rather than just establishment of technical processes. The analytic part of decomposition is concerned with the insights that are gained, but to attain these, a transformation process synthesis is required at each and every step of decomposition. As argued in the previous sections of this Chapter a decomposition involving the definition of system boundaries, elements and prescribing interrelationships between them is not an analytic activity, but on the contrary it is a synthetic activity. Therefore, hence forth the term used in respect to design activity will be the synthesis of technical processes.

2.4.3 Synthesis of technical process as a prerequisite for novel product design

In respect to engineering design synthesis two design situations are envisaged by the TTS; design of a completely novel product and a redesign task [12]. Designing is a mixture of systematic and intuitive processes and the TTS defines its task to stimulate efficient search of solution alternatives on all the levels of abstraction [1]. The focus is set to tackle the problems arising with the novel design, but the theory might be successfully applied also for the redesign tasks. According to the TTS, during design process a technical system can be considered on different levels of abstraction including technical system's internal transformation process, organs and components used to realize functions and organs at any level of completeness. Depending on the intended aims and resources assigned, most often a product redesign starts with the identification of the existing organs structure or functions these organs fulfil [12]. Starting at higher abstraction level will enable broader search, thus generating more solution variants, where if the aim is to preserve product development resources than design process will be limited adopting the most of the existing product's structure and behaviour considered at different abstraction levels. After the initial analysis step the rest of design process can evolve on the basis of the prescribed methodology given by the TTS exhibited in the very well known analysis-synthesis-evaluation cycle.

As any problem solving, design starts with a task clarification process which for an outcome has to point out at least vaguely what is to be achieved, i.e. what is the required behaviour of a future product in order to help it satisfy the existent market and societal needs. Apart from design specification as an initial and fixed starting point which is a result of awareness of current needs resulting from a market analysis and surveys, problem definition also takes part later on inside the design process itself. The latter can be understood if bearing in mind the nature of design process which is performed as an explanatory search with constantly evolving mutually dependant problem statement and solution. Of course, the explanatory search, or creative innoduction, occurs at the point of design process where synthesis is the most intensive, therefore mostly constraining the problem redefinition to conceptual design phase. Building on the same foundations, the TTS states the principle differences between design of a novel product and a redesign on the grounds of the type of initial design activities performed in order to determine and clarify what is and how is to be accomplished. With redesign task a problem definition emerges as a result of an analysis of the existent product's structure at desirable level of abstraction whereas a design of a completely new product considers technical processes synthesis as a part of design problem definition. For instance, the function of technical system is a capability to deliver necessary effects is derived on the basis of technical process synthesis, what is in contrast with redesign which would start with an analysis of an the existent product in order to obtain the function structure.

Natural question which has to be stated at this point is what are the central points that need to be addressed when designing a novel product? According to the TTS, the main areas of concern are synthesis of the optimal or at least of the appropriate technical process which is referred to as synthesis of horizontal causality chain and establishment a structure of the internal technical system's transformation denoted as vertical action chain [1], [8] (see Figure 2.7).

Each technical system exhibits different structures and relationships of its elements depending on the abstraction level considered, e.g. function, organs and components. After the technical process and the necessary effects have been clarified, design of technical system can be equated to establishing the vertical action chain and making it realizable by physical components. Depending on the task, the exploration of systems structures at various abstraction levels may or may not be used by designer to help design a technical system, however, the outcome must be in physical components structure. The vertical action chain is

performed through a set of actions which occur within technical system's internal transformation.

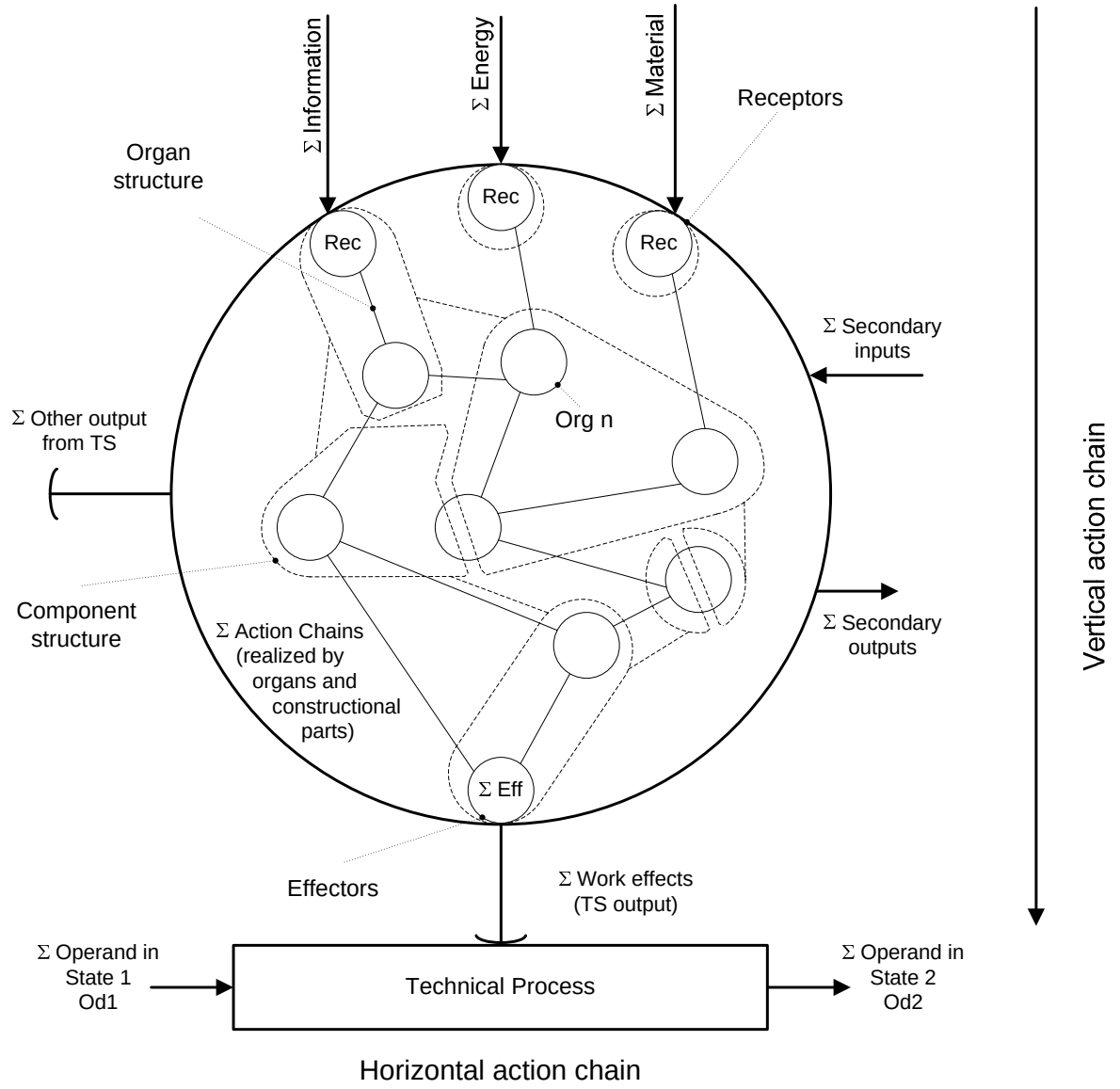


Figure 2.7 Horizontal and vertical action chain [1], [8]

The latter is modelled as an analogy to technical processes and it may be considered as the upmost abstraction level considering technical systems as proposed by the TTS. Since this thesis is concerned with the development of computational support for generation of operand transformation alternatives it feels necessary to address the distinction between the two transformation processes and clarify the relation between internal transformation and functions of technical system. Internal transformation process represents a technical system in its working state. The input on the system boundary inputs consist of material, energy and information which are within internal process being transformed into the needed effects. Effects may be movement, energy flux as heat or cooling for instance, protection, force and so

forth. The principles of transformation are prescribed inside the mode of actions which are derived from physical laws. Output locations on which the effects are delivered to the corresponding operations to drive technical process are denoted as action locations. Principle difference between technical process and internal transformation process, lays in the fact that technical system's transformation is performed internally by technical system itself [1]. What is of interest is that the TTS when considering function structure models technical systems as being in the state capable of working, and the effects which are in fact products of the internal transformation as a technical system's capability of performing necessary tasks what is most commonly referred to as product's function. Even further, the TTS proposes to equate internal transformation and functions in one to one correspondence [12] thus rendering one level of consideration as unnecessary.

2.5 Implications to this work

This thesis will not undertake such ambitious task that aims to simulate the whole process of engineering design synthesis; rather it will just try to automate one stage, i.e. technical process synthesis as given by the TTS that is, of methodological design approach inside its own-build logical framework where some of the problem solving methods and techniques can be computationally utilized. Computationally supporting the existent methodological design approaches additionally benefits with the ease of in-practice acceptance, since designers are supposed to be familiarized with various approaches to design during their studies at universities or as a part of their professional everyday routine. Moreover, since the methodological approaches serve as templates which are to help designers to more efficiently solve the given task within a meaningful sequence of steps and although these steps involve different cognitive processes which cannot be so easily put into an algorithm, yet it is still a firm foundation for the creation of computational design support tools. The extent of mapping from systematic approaches to computational environment usually cannot be achieved in one-to-one manner. Either appropriate encoding of preferred design methods must be devised in a manner acceptable in order to suit the existing computational techniques or new computational methods must be developed altogether.

The aim of this thesis is set at generation of method and operand tool for operand transformation variants in order to support design process. As shown, the transformation is of special interest to the TTS as it recognizes both, the horizontal or technical process and the vertical as the internal transformation process. Designer in relation to different technological

principles composes the structure of technical processes in order to achieve desired operand states corresponding to the existent market and societal needs. Technical processes synthesis is a part of design task clarification. It contains both the operand in its end state which should be in compliance with the existing market and societal needs and the behavioural properties of the transformation process itself. According to the Theory of Domains (TD) the latter is referred to as universal virtues of technical process encompassing cost, quality, risks, environmental aspects etc. If metric could be established over universal virtues than they could be used as objectives which alongside operand's end state as a goal could serve to optimize technical processes. Both the TTS and the TD realize the importance of technical process synthesis as the requirement for successful design of a novel technical system.

3. COMPUTATIONAL DESIGN SYNTHESIS

It takes two to invent anything. The one makes up combinations; the other chooses, recognizes what he wishes and what is important to him in the mass of the things which former has imparted to him (P. Valéry; taken from D. E. Goldberg's, The Design of Innovation [11]*).*

A fairly young research field of Computational design synthesis (CDS) to whose body of knowledge would the findings of this thesis be hopefully incorporated, has in the recent two decades emerged as a constitutive part of The Design Science. The advent of computers has excelled solving of engineering design problems which initially were more or less computable as calculation based tasks. However, with the ever present Moore's law the increase in the available computational power, the ongoing development of programming frameworks, and advances in the field of discrete search and optimisation techniques, the extension in the applicability to tackle problems involving complete designs generated by computer applications have become possible. About a decade after the widespread of expert systems in the eighties of the last century, the first significant successes were produced inside the evolutionary design frameworks aimed predominantly at the engineering optimisation tasks [11], [34] and [35]. Most often the results of topological optimisation surpassed the initial intention of just optimising yielding in complex designs which have emerged as a consequence of evolutionary learning processes encompassing trial and test search method which altogether very much resembled the process of engineering design synthesis [36]. Currently the CDS supposes an algorithmic creation of designs implemented on computers involving an organized approach and methodological modelling [2]. It is a complex multidisciplinary research area that brings together advanced computational techniques and search algorithms with the knowledge about the object of design and design processes. Rather than just aiming at the optimisation, by building on the principles on which human designers arrive at a design solution the goal of the CDS can be formulated as to provide an assistance in situations where solving of a problem would require generation of a too large to cope number of variants. Chapter 2 presented findings about the problem solving in general. The models of engineering design synthesis and the synthesis of technical processes as the focal point of this thesis where adopted according to the TTS. These postulated the theoretical foundations of this research taken from a Design Science's viewpoint. This Chapter, on the other hand, will give the state-of-the-art overview on Computational design synthesis methods

and tools in order to add the computational perspective of this thesis. Theoretical origins, product modelling approaches and mechanisms by which the existent CDS methods and tools generate designs will be explored in order to determine the general principles that will be used for development of computational method and tool for evolution of operand transformation variants. Since grammar based approach will be used in this thesis a more thorough overview of similar methods will be presented. The necessity to considerate technical processes inside computational synthesis tools will be re-emphasised.

3.1 Generic model of CDS

To consolidate various approaches, methods and tools that emerged over the years, efforts were made to establish a generic model of a Computational design synthesis process. Two correlated models appeared in the literature: a generic framework [2] that proposed representation, generation, evaluation and guidance as four basic steps (Figure 3.1) which must be addressed inside a computationally driven design synthesis process; and a performance-based framework emerged for topological synthesis proposing investigation, generation, evaluation and mediation as steps of a parametric based computational synthesis [38] (Figure 3.2). Although the two approaches differ slightly by the nomenclature, the content of the proposed steps is almost the same. To reflect on the CDS steps, this research will adopt a model nomenclature according to a generic model of CDS [2] (Figure 3.1):

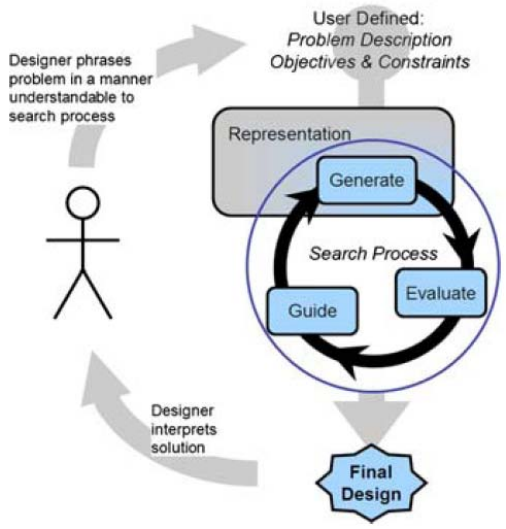


Figure 3.1 A generic model of CDS [2]

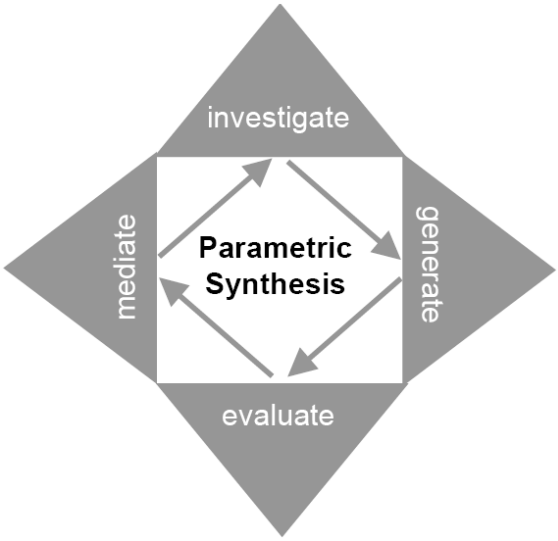


Figure 3.2 Parametric synthesis model [38]

Both of the models consider the Computational design synthesis in the context of a very well known analysis-synthesis-evaluation cycle which is so often present in the everyday problem solving. However, since CDS assumes a design synthesis performed on the computer and by

the computer, than the content of the proposed steps is tailored so to meet the requirements of such execution. Of course, the immediate and obvious is the necessity to represent the problem of interest in a way which is understandable to computational environment. Formalizing a problem using a language is a kind of problem representation for itself which rests on a well defined formal system, but what is more elusive is how to cope with different mental models that designer constructs when dealing with design problems. In the opposite to static bounds of programming code, the human mind when confronted with a task acts as a pre-processor using various representations as felt fit in order to patch up a complete solution inside a generate-and-test processing loop. Creation of analogies, understanding of semantics and utilization of abstract reasoning which occur so easily inside one's mind during problem solving are difficult to be performed computationally, and in order to even get near the simulation of these cognitive processes, an establishment of in-domain knowledge understandable to computers is necessary. As shown many times before by the evolutionary computation community [34], [11] and [35], the outcome of a search process is directly dependant on the feasibility of the encoding that was applied. An effort was made to emulate designers "out-of-the-box" kind of reasoning [26] framing it by using dynamic fitness functions which were evolvable themselves thus adapting to the course of evolution [37]. However although promising, such considerations always retain the same founding model thus being not sufficient to achieve the desired effect. At the moment, the CDS understands problem representation as a static part of synthesis framework which ought to be designed in a fashion that captures the most of the form and attributes of the design search space [2]. Moreover, the problem representation should be founded on the established design theory models as these are familiar to designers and are constructs of years of research and expertise in design processes study. Depending on the purpose of design synthesis, the complexity of the task and considered viewpoints on technical system's structure applied at different stages of design process, the solution representation may be realised as a geometrical expression of object's form as distribution of material in space, a real valued vector, matrices, graph structures according to different design process models etc.

In contrast to representation, the generation is of course a dynamic part of the model, as well as the evaluation and guidance. As shown in Figure 3.1, the mechanism of solution generation is closely linked to the type of representation which can be selected or devised anew. It should be the best possible fit to the encoding requirements in order to get as much as possible from the search across the design space. Moreover, since solution generation very often includes

systemic integration of building-blocks, a generation method must also obey the approach by which the synthesis method utilises engineering knowledge necessary to perform that integration. Therefore, it can be said that the principles by which solutions are synthesised are founded according to human problem solving methods and techniques ranging from the straightforward random generation to more complex knowledge intensive methods. Typical generation principles may be heuristics, matrix transformation, and state space search over a tree based model. Solution can be generated as a result of collaboration of various types of semi or fully autonomous programs, i.e. agents or soft-bots, which act as driven by the perception of their environment [2]. Very often genetic algorithms are utilised for design synthesis, and these require additional decoding function which accepts genotype at which evolutionary operators perform and transforms it into a phenotype to obtain the structure and behaviour of the system in order to be able to evaluate, rank and select the most feasible solutions [11]. For the creation of computational support aimed at conceptual design, graph structures are of special interest since most of the design theories use these to model technical system in the early design stages. Graph grammars are then used as a means to construct and explore different solution variants.

Evaluation considers ranking as a result of solution analysis showing how well the solution candidates perform against the preset requirements. Assignment of a rank to a solution over multiple objectives in the present engineering optimisation tools is most commonly performed using the well known the Pareto principle. The same is encountered in the CDS. Rather than guessing weights and summing them up, the weak the Pareto principle for example provides at least an objective ranking approach which for a consequence results with a collection of feasible solutions bearing the same rank. Ranking demands the establishment of some kind of metrics according to which evaluation can be performed. It was shown that performance based synthesis [2], [38] usually opts for integration of a commercial simulation and analysis software packages. Most of these packages rely on numerical solving of systems of equations composed in respect to mathematical model applied and boundary conditions input. Integration of such systems which are often originating from different developers results in tedious but unavoidable programming and scripting involving several programming languages to be able to patch up a semi automated and to an extent controllable framework. The other option, which is, of course, more elegant and totally controllable, is to develop analysis systems from the scratch which most often surpasses the intended resources and stems out of research focus leading to a trade of situation between the two options.

Guidance completes the whole CDS loop. Based on the results of the analysis and assignment of rank to individual solutions the feedback is provided to the synthesis system enabling it to guide search further in the direction of improvement of the results [2]. To select the most feasible solution different strategies can be applied ranging from random walk, or in contrast, it can be guided as a result of an emergent property of the algorithm, like for instance in the case of genetic algorithms. Sometimes the assistance and supervision of designer is called for in order to interpret the results of search accordingly. Some of the very well known selection principles include TABU based selection, greedy search,, evolutionary steady-state, elitist or roulette wheel selection principles, burst algorithm selection etc.

In the following sections of this Chapter a related work will be analysed in respect to four steps according to Computational design synthesis model [2], [38]. The findings will be used to in order to specify scope and impact of this thesis.

3.2 State-of-the-art on CDS

The first few approaches examined consider various non-grammar based techniques which stem at synthesizing of product concept as a configuration of components. The search for solution alternatives is always conducted only at the component level by associating them on the grounds defined by users' inputs. Most often configuration of components emerged as a result of stochastically controlled mapping from predefined product functions to components. All of the three following methods have adopted theoretical foundations according to Systematic Design [4].

A good example on how to apply the existent search method for the conceptual design stage is a genetic algorithm (GA) based search inside a morphological chart [39]. The problem of generating optimal concepts was reduced to a combinatorial search with GA searching for an optimal set of technical solutions that can realize product sub-functions. The components were divided into the domain dependable families and had to be pre-selected by the user in order to obtain meaningful concept solutions. The approach only dealt with problems having a number of functions for which it was meaningful to represent their relationship inside a simple chain. If considering more complex structures, the user had to compose function structure in such manner that the overall structure is reducible into series of function chains. For the creation of functional models it was proposed to use standard taxonomy as defined by NIST [21], [22]. The validation of the solution principles is performed by the energy flow

compatibility check. To tackle the problem of multiple objectives, a canonical GA was used with fitness function defined as algebraic sum of objectives multiplied by a corresponding weighting factor.

Similar approach was undertaken with the Concept generator which is a computational tool developed by Bryant et. al. [40] intended to create design solutions by establishing mapping from a predefined function structure to the lists of components using matrix algebra. Solutions are generated on basis of a web sited repository of function-to-component matrices (FCM). The FCM's show which technical solutions can realize a given function and a design structure matrices (DSM) in which component to component compatibility in respect to energy flows is defined. Ranking is achieved by comparing the frequency of occurrence of components inside the generated solutions to the data gathered from over 70 consumer products and put inside the repository. Since the Concept generator can only accept chains of functions as an input to the search, then the initial functional decomposition of a product considered has to be partitioned in the same manner. Based on the sub-function chain input and corresponding technical solutions derived from FCM using matrix algebra, the outcome is a full set of all possible component configurations. DSM is then applied to filter incompatible components. Apart from chaining that occurs both in the input and the generated solutions, the main drawback was not considering the possibility that multiple components can realize one product function.

A different approach was presented with A-Design [41] which included a collection of software agents or softbots with embedded knowledge, enabling them to perform specific duties in order to create meaningful solution concepts using a catalogue retrieved components. A-Design is founded on an assumption which relates design to optimization processes with a solution generated and improved through iteration until meeting the set of predefined objectives. Different agent types where developed; configuration agents which performed an interface based connection of components managed by an input-output type compatibility check, instantiation agents the duty of which is to retrieve new component from the catalogue and fragmentation agents that segmented solutions and preserved them to be improved in iteration steps to come. The selection of agents based on their merit of past performance was controlled by a manager agent, however to avoid local optima the cooperation had to be randomized to an extent. Learning was achieved in a process similar to TABU search algorithm; designs were identified as the Pareto optimal, and as good or bad, and were stored as such. Than at the end of each iteration step the manager agent was employed to start a

dialogue with the user prompting for his or hers action to additionally interpret the stored designs. The user had a chance to affect evolution the course adapting it to its own preference so that the created designs successfully meet the preset criteria in a desired way.

3.2.1 Shape and spatial grammars

Using formal grammars for engineering applications was first achieved by Stiny and Gips [42] who developed shape grammars as a production system that specifies a set of design solutions called a language, by the transformations required to generate that set [43]. Thus, to create a formal grammar it is necessary to define elements as well as a set of production or inference rules on account of which these elements are transformed inside a formal system. To specify transformations, two sets of elements have to be defined; a set of symbols denoted as variables onto which transformations can be applied, and a set of terminal symbols or alphabet from which onwards no more transformations are possible. In respect to developing computational support for product development and design, applying a sequence of rules from such formal system as formal grammar is, implies caring through a series of transformations necessary for the creation of design solution. Thus, a successive derivation of all possible combinations of rules creates a design search space regarded as a formal language of specific engineering domain for which the rules were defined for. The resulting design solution can be understood in a linguistical sense as a syntactically correct expression or a sentence composed of alphabet of a formal language. In order to produce optimal solutions an extension to the method was made with an addition of various stochastic search algorithms. For instance, a shape annealing become by matting shape grammars with a simulated annealing algorithm [43], [44] and these were successfully applied for solving topological optimisation problems of truss structures involving both in-plain and in-space problems. Design of product's form or shape assumes aesthetics and appearance that can attract potential customers. A grammatical approach to structural design offered that by including aesthetic principles or specific style in a set of rules by which a form of an object will be created (Figure 3.3 - Figure 3.5), [43].

A series of industrial design papers attempted to identify brand style features and then to generate solutions using these specific styles embedded within grammar rules. Research was predominantly funded by the automotive industry [45], [46]. Although shape grammars are invented to create and investigate solution alternatives for architectural or industrial design where function is a direct consequence of object's form or shape, the basic principle of

grammatical formalism seemed extensible and applicable for supporting of other engineering domains concerned with different types of artefacts.

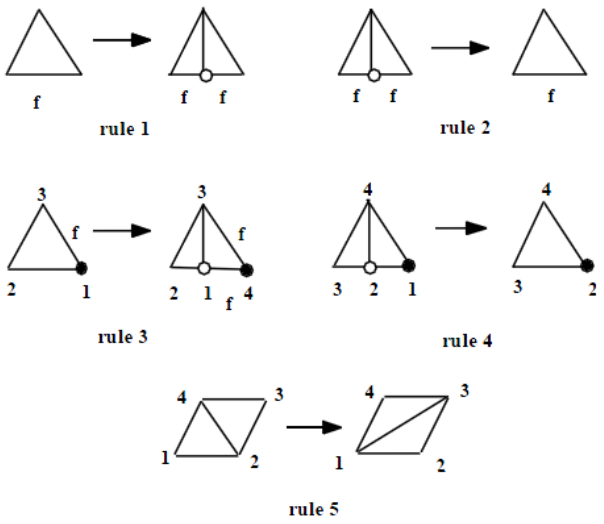


Figure 3.3 Planar truss grammar (f - free line, black dot - fixed point, white dot - free point) [43]

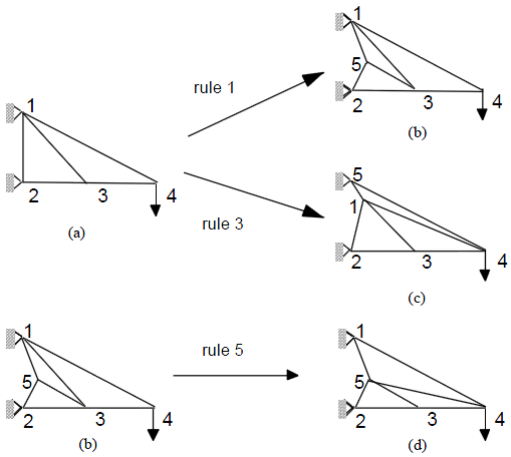


Figure 3.4 Example of rule applications [43]

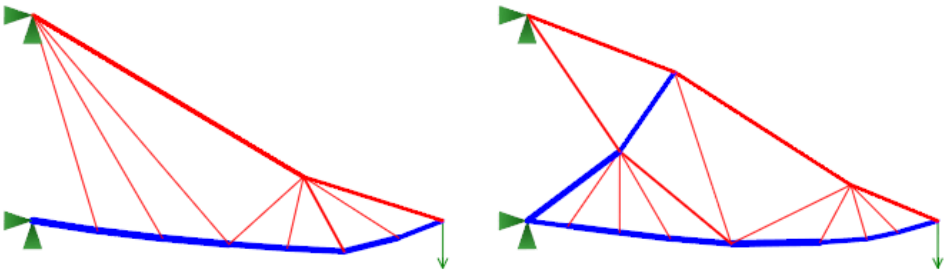


Figure 3.5 Cantilever truss designs for minimum mass using shape annealing [43]

In view of mechanical design and product development, shape grammars based methods and tools are most commonly aimed at supporting design phase which includes late conceptual and embodiment design stages. Often, the structure of a product or the optimal configuration of product components to achieve for instance close packing but to retain desired functionality is the aim of support. As a consequence, the grammatical systems which are used are referred to as spatial rather than shape grammars. Most of the existent methods and tools developed and used today are domain and product specific and are therefore lacking a common model [47].

A parallel grammar for mechanical design synthesis was developed by Starling and Shea [48]. The aim of the research was also to investigate the feasibility of creating simulation-driven search in order to produce better quality designs. To achieve that, a cross-domain modelling

language *Modelica* inside *Dymola* was used to obtain simulation results. However, full automation was never reached, so instead, precompiled simulation executables were used that required only parameter value update inside input files to deal with new designs. Parallel grammar was founded on FBS product representation [49], [50]. It consisted of two types of rules: function grammars which generate function structure using predefined building-blocks and structure grammars which then create parametric component structure as a simulation starting point. The Pareto optimal solution was found using hybrid pattern search algorithm. As an extension of the parallel grammar method and the research of Starling and Shea, a simulation-driven method for gearboxes synthesis was developed by Lin et. al. [9] (Figure 3.6 - Figure 3.8):

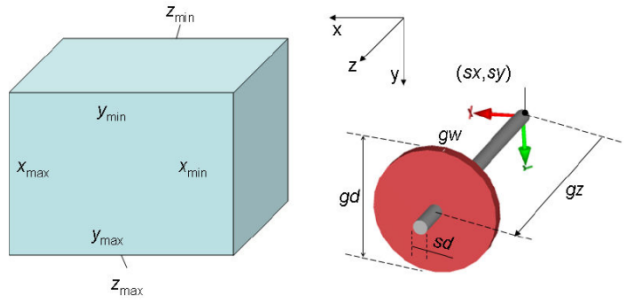


Figure 3.6. Packing bounding box and shaft and gear [9]

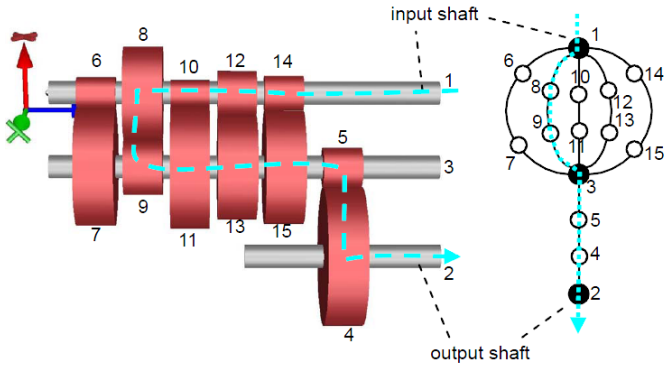


Figure 3.7 Component vs. graph representations [9]

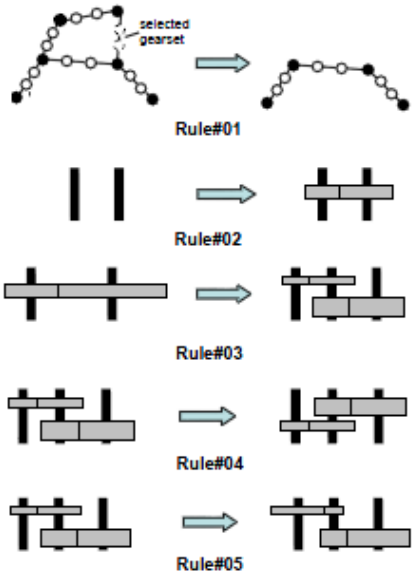


Figure 3.8 Rule examples [9]

The method scope was limited to embodiment design. The component structure was represented using a virtual graph consisting of gear pairs and shafts which thus were depicting a power flow inside a gear-box. The system topology and geometry modification were derived by following a set of spatial grammar rules inside a simulated annealing search process. Grammar rules where ranked according to the performance of designed they created.

An interesting approach for computational support of simulation driven microelectromechanical system (MEMS) synthesis was presented by Bolognini et. al. [51]. CNS-Burst method was developed as a combination of Connected-Node System which is in fact a hypergraph based representation of MEMS systems, and a multi-objective generate-and test search algorithm denoted as BURST. Search principle is based on the procedure where the CNS modification operators are applied in short bursts to current system layout. Frequencies of modifications are user-defined. Special evaluation module was used to obtain performance metrics of the created system by which a non-dominated solution population was created.

3.2.2 Graph grammars

Capturing the engineering knowledge as a set of production rules and then obtaining solution alternatives by automating derivation process computationally provides a generic model of design support on the basis of which basis designers can establish their decisions when considering product realization possibilities. For the creation of computational support for the early stages of product development most common is the application of graph grammars [2], [48] and [53].

Following the systemic reasoning, technical processes and technical products are most often modelled as transformation systems, both formally and visually represented as graphs. Depending on the abstraction level and context of respective early design stage, transformation system's elements can differ but the basic graph representation will be retained. Graph grammars are like shape grammars defined as production systems consisting of vocabulary and alphabet, and a set of rules for implementing graph transformations. In most cases computational support is provided for product function structuring and component configuration. Optimisation is most often aimed at component level.

A good example of how to tackle the problem of computational concept generation using grammars was presented by Jin and Li [54]. The idea of their hierarchical coevolutionary design approach (HiCED, Figure 3.9) is to iteratively co-evolve products on different abstraction levels in parallel. First, based on the knowledge stored inside a rule library, an initial population of functional decompositions is created. Then, a genetic programming and genetic algorithm are triggered to co-evolve products' functions and components as functional means. Functional and component structures are represented with simple flow graphs. The

fitness function is formulated using multiple weighting factors. The idea of co-evolution on different abstraction levels was explained earlier in the General Design Theory (GDT) [30] and function-behaviour-state functional modelling (FBS) [49], [50].

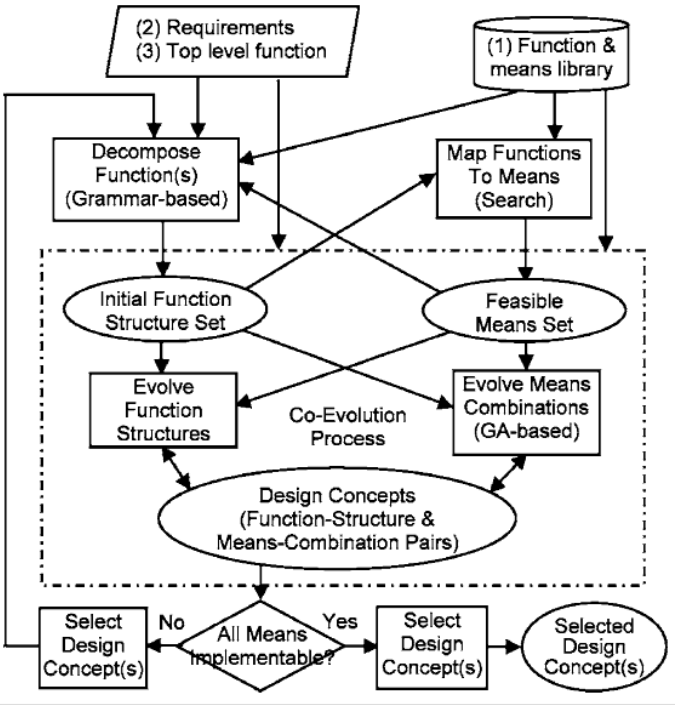


Figure 3.9 Design process of HiCED [54]

The existence of product functions is confirmed through the behaviour of components as a result of deductive and abductive reasoning. Therefore, there is a point for using co-evolution on different abstraction levels of the product abstraction. The only problem is how to formalize a co-evolution process for computational purposes in order to make it generic.

Schmidt and Cagan developed GGREADA [55] which is an approach to graph grammar for support of mechanism synthesis. Built on the foundations of its predecessor, the FFREADA algorithm which is a function-to-form recursive annealing algorithm that used a string of symbols to generate hand drill designs, GGREADA uses graph grammars to generate concepts using components based on a *Meccano*® parts set. It is a mixture of configuration and catalogue selection design tasks. Function-to-form transformation is realized by top-down reasoning by which a component is selected to realize a product function. Also, a function-sharing in respect to a component realizing several functions is supported. Instead of state-space search algorithm, GGREADA tried to use a simulated annealing to recursively evolve a product on two abstraction levels: function level and form (component) level. In that way it

outperformed state-space search. The objective function was formulated as multi-objective with multiple weighting factors.

Determination whether a solution is unique and the recognition of similarities in structure are most often present in the evaluation phases of engineering design. Design synthesis methods and frameworks which rely on graph representation of a product, resolve that issue by detection of isomorphism among solution variants. Most often this demands a special detection algorithm to be developed. Siddiqque and Rosen use graph grammars to develop a Product Family Reasoning System (PFRS) which would help designers in developing product platforms [56]. Two questions were addressed: how to establish common platforms for a set of different products, and the opposite, how to specify the product portfolio supported by the platform. Using sub-graph isomorphism, common functions can be identified. First, the production rules were applied to generate a variety of product function structures which were then mapped to components containing relationships among functions and components. Afterwards the identification modules were represented as hypergraphs. Answering how to specify the product portfolio supported by the platform, results in viewing the grammar not as a generative but as an acceptance grammar thus parsing the product architectures to see whether they fit in the language of the specific product family. Slightly different approach aimed only at structure synthesis was developed for the automated synthesis of mechanisms, for epicyclical gear trains in specific [57]. Graph grammars were used to add vertices and loops to the initial start graph, and with the interpretation of resulting structure by processing vertices and edge labels the desired gear transmission ratio was obtained. Additional graph grammar rules were added for identifying isomorphic graphs what enables designers focus just onto unique solution variants.

Wu et. al. developed a systematic approach for automated support for design of mechatronic dynamic systems based on bond graph formalisms [52]. It is a simulation driven approach which requires as an input a conceptual definition of dynamic system to define a state space. For that purpose, a conceptual dynamics a CD graph is introduced, which represents information about the connections between components of a system. Generic models of components having various types of connection possibilities are stored within a repository. Dynamic model of a system represented with state space equations is automatically generated on account of a defined concept using bond graphs transformation and user defined goals. Optimization is performed using real-valued genetic algorithm with individual solution

genotype being derived based on hierarchical representation of component design rules, constraints and physical laws.

BOOGGIE, which is a recent method developed according to FBS product model, tends to make use of the available graph grammar transformation tools and other available open-source software packages and to integrate them into a framework for synthesis of mechatronic products [58]. GrGen [59] which is utilised for conducting the graph grammar transformations, and open-source TULIP provides a graph visualisation. Integration of SysML modelling language is also being considered. The framework enables user to visually define rules which are then interpreted to GrGen's internal script language. Framework considers top-down approach of decomposing product's structure on all three levels of FBS (Figure 3.10). Currently the framework only aims at variants generation without the optimisation support.

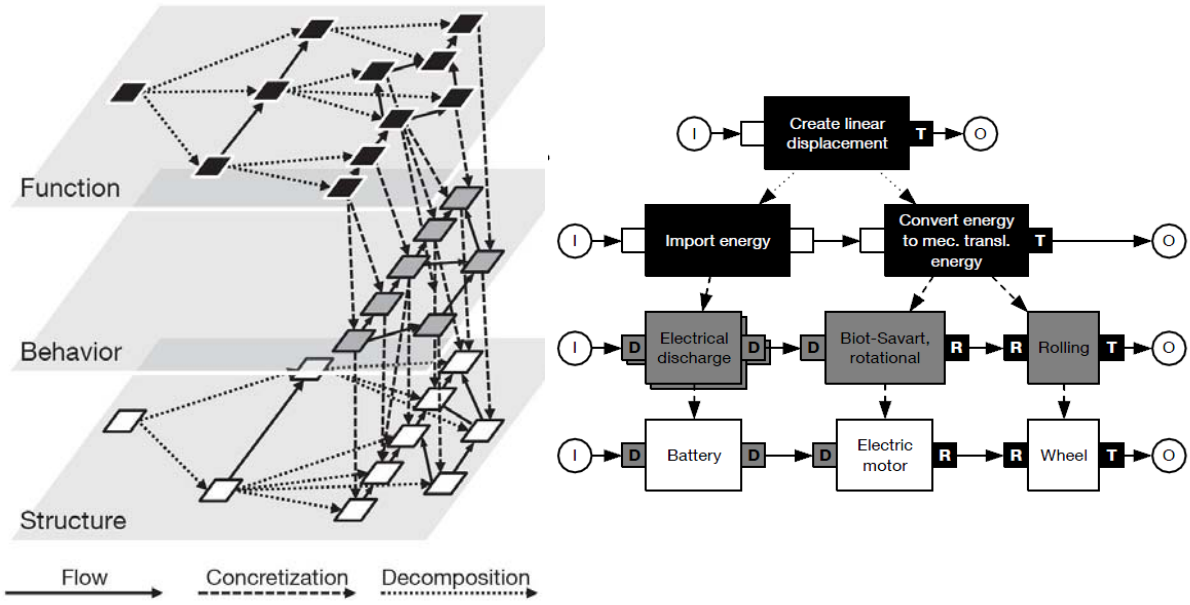


Figure 3.10 Top-down graph grammar approach on all three levels of FBS [58]

3.2.3 Other approaches to early design support

Cambridge Advanced Modeller (CAM) developed by Wyatt et. al. [60] is a computational tool built on top of P3-Signposting which aims to support product architecture design. In order to help designer systematically consider conceptual variants of product configurations, the method generates a set of all possible alternative architectures for a given product. The basic principle of the approach claims that for any given initial architecture, any other architecture of that product is reachable through a state space search process by carrying out

sequence of transformations. Therefore, it is required that designer using a graphical modelling language for an input defines a schema which is a graph composed of a finite set of different relations, components and logical constraints. The schema logically frames an architecture state space. Using a depth first search, elementary transformations on initial product architecture are executed and then tested against the proposed schema. Two evaluation metrics are proposed: changeability representing immunity to change propagation, and designability which represents the relative design effort required for architecture.

Another recent approach that is worthwhile mentioning, is a SOPHY developed by Rihtaršić et. al. [61]. It aims at supporting designers by generating a sketch of a concept clearly depicting the working principle on which that concept is based. Process of sketch generation is performed with the assistance of designer but using predefined building-blocks or schemes which are coupled into complete concept. These basic schemes are in fact a representation of wirk elements. Which schema will be used is resolved with automated part of the method and the tool which manages to create linear chains based on physical laws, which is based on the principle of causality. The tool substitutes one variable from the chain of equations, i.e. physical laws, until a derived output equates the one as specified by designer. More physical laws can help to resolve one product's function, or more alternatives for the same function can be generated. At the moment, neither open systems nor expressions containing n-dimensional vectors can be modelled. Similar approach based on the chaining of equations was utilised some years ago for development of mechatronic systems.

3.2.4 Implications on this thesis

The overview of methods and tools shown in Table 3.1 (see the following page) gives a summary of the state-of-the-art analysis on CDS presented within this Chapter. Recent methods and tools are put in comparison in order to justify the scope and principle of synthesis method as proposed in the introductory Chapter of this thesis. The overview is structured to show what means are used to perform synthesis steps according to the adopted generic CDS process model (Figure 3.1), and to show for which design phase was the computational support intended. If relevant, the underlining design theory and methodology according to which a method was founded is also identified. What is clearly visible from the presented data is that the bulk of methods and tools for the support of conceptual design and concept generation (see Table 3.1 rows 6-16) choose graph grammar or spatial grammar representation.

Table 3.1 Overview of CDS methods and tools

No.	Authors	Method	Design Theory	Scope					Method			Year
				Technical process	Function structure	Organ structure/ Behaviour	Components/ Structure	Architecture/ Platform	Representation/ Investigate	Generation	Evaluation	Guidiance/ Mediate	
1	Campbell et. al.	A-Design					x		Catalogue based design	Agent based/ Heuristics	Multi-objective, Pareto	TABU based learning	2003
2	Hutcheson et. al.	Concept Variant Selection	Pahl & Beitz				x		Morphological chart	Heuristics	Multi-objective, multiple weighting factors	Evolutionary Building-block hypothesis GA	2006
3	Bryant et. al.	Concept Generator	Pahl & Beitz		(x)		x		FCM Tree	Matrix algebra	Component occ. frequency based ranking/ Component Compatibility from DSM	Enumerative	2006
4	Wyatt	CAM						x	Graph (Schema/Configuration)	Elementary operations applied to schema	Multi-objective, Changeability, Designability	Constraint based Depth First Search	2009
5	Rihtaršić et. al.	SOPHY	TTS			x	x		Schema (Physicall law-wirk element-ports)	Variating over chains of linear expresions/ Semi-automated sketch generation	Causality	Enumerative	2010
6	Schmidt, Cagan	GGREADA	Pahl & Beitz		x		x		Function structure Component structure Graph	Graph-grammar	Perfomace metrics embeded inside rules	Metropolis criteria SA	1997
7	Siddique, Rosen	PFRS PC/PSPV	Pahl & Beitz		(x)		x	x	Function structure Graph	Graph-grammar	Sub-graph isomorphism/ Acceptance grammar	Enumerative	1999
8	Starling, Shea	Parallel grammar for Simulation - driven Mech. Design Synthesis	Pahl & Beitz		x	x	x		F-B-S Graph	Graph-grammar/ Structure grammar	Simulation driven Multi-objective, Pareto Performance metrics - semi-autom. simulation	Hybrid pattern search	2005
9	Jin, Li	HiCED	Pahl & Beitz		x		x		Function structure/ GP Tree/ Binary string	Graph-grammar	Multi-objective, multiple weighting factors	Evolutionary Building-block hypothesis GP, GA	2007
11	Helms, et. al.	BOOGGIE Mechatronic design synthesis	Pahl & Beitz			x	x	x	F-B-S Graph	BNF Graph-grammar	Componenet Compatibility Simulation Driven Performance metrics embeded inside rules	Enumerative	2009
12	Schmidt, et. al.	Structure Synthesis of Mechanisms					x		Labeled graph	Graph-grammar/ Grammar-based rules for isomorphism detection	Powertrain ratios on basis of labeled nodes and edges	Enumerative	2000
13	Bolognini, et. al.	Computational synthesis metoda for MEMS Design					x		CNS (Hypergraph)	Rule-based spatial graph transformations	Simulation driven Multi-objective, Pareto Performance metrics - automated simulation	CNS-BURST with non-domination principle	2007
15	Lin et. al.	Automated Gearbox Synthesis					x		Graph, (Power flow paths)	Spatial-grammar	Multi-objective, multiple weighting factors	Past performance rule selection SA	2009
16	Wu at. al.	Bond Graph, CAD of Dynamic Systems				x	x		Bond graph	Automated mapping from system concept to Bond graph	Simulation-driven Performance metrics - automated simulation	Evolutionary Building-block hypothesis GA	2008

Argumentation for that can be drawn on account that almost all of design theories model product as a transformation system of some sort which is then in a formal and visual manner represented as graph. From mathematics and computational sciences it is known that graph transformations can be achieved using formal grammars [94] and with an addition of advanced stochastic search methods the possibility is created to evolve graph-like solutions in an elegant manner. What adds more points in favour to the selection of grammars is that engineering design knowledge formalization for specific area of application can be achieved using formal grammars. Moreover, the decomposition process as an activity that is so often performed by designers the early design stages is to an extent analogous to grammar derivation process. The latter will be addressed in the next Chapter in more detail. Formal grammars are means that were extensively used thought the history of AI to create formal systems that allow machine learning and machine induction [15], [62] . At the moment no grammar based system of such capabilities has been implemented in the research area of product development, but in time, they will appear since grammars allow such possibilities. Moreover, rather than creating a number of methods based on different principles (see Table 3.1 rows 1-5), formal grammars offer the possibility of creation of a unified formal language of product development.

3.3 Implications on this thesis

The creation of a unified formal graph grammar based language of product development is a distant and visionary idea, but as people as individuals may understand and speak several different languages, then why shouldn't the language describing engineering knowledge in different domains be shared and understood by different computational systems. The research presented within this thesis will embark on route of developing computational support for operand transformation variants in technical processes. Given an overview showing current research efforts aimed at graph grammars utilization for the creation of early design support and in a view of justification presented it seems reasonable to selected graph grammars as a mean to synthesise technical process variants.

As shown in Table 3.1, the highest point of abstraction from which current approaches start is the functional level not recognizing technical processes at all. Reasons for being that so is argued in some of the recent publications in CDS [63], [64] where it is pointed put that TTS and its philosophical views are not widespread often concealed by the well known Systematic design. Focusing only on technical system excludes other parties participating inside a

transformation system (Figure 2.4) and thereby neglects processes of interaction with human operator and the outside environment. If the ability to deliver desired effects is considered a function of the product (behaviour), then the need for these effects must be recognized before the functional decomposition. Converting effects into initial conditions of search starting at technical system's function level has for a consequence an unavoidable restriction in design search space. Further consideration of technical system at lower levels of abstraction cannot add new effects since they would redefine what a technical system should do wthin a transformation system, and the only other way to accomplish that change is to affect the technical process inside where the main operand transformation is realized. Therefore, a selection of the Theory of Technical Systems to provide a theoretical foundation to this research in field of the CDS can be summarized with the following claim: variations on the process level yield different function decompositions; as a result a design search space broadens.

Fellow researchers in the field of design theory and methodology may point out that only TTS, TD and TRIZ in their special way recognize technical processes as such, and that different methodology such as Systematic Design is or FBS product modelling excludes technical processes from consideration (Table 3.1), which is in fact true [50]. However this does not mean that technical processes do not have to be considered and that an addition of another layer, the technical process as a top layer, in a form of computational method and tool could be beneficial to the current research efforts made in CDS. From design theory point of view current approaches that aim at differencing functions as lower and upper, where the latter denotes fulfilment of societal needs, would hopefully embrace reasoning as presented by TTS or TD, since functions do not equate processes.

4. FORMAL GRAMMARS AND LANGUAGES

The question, 'Can machines think?' I believe to be too meaningless to deserve discussion. Nevertheless I believe that at the end of the century the use of words and general, educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted. (Alan Turing, 1950 [65]*)*

The introduction of information theory and the advent of computers made possible that any kind of mathematical objects like graphs, matrices, images or even sound waves can be interpreted transformed and conveyed as sequences of strings composed of symbols [66]. Development of programming languages to write computer programs that when compiled and executed pass sequences of instructions to processor, all of this, each in its own special way operates on the basis of some sort of formal system which involves transformation of string like, or, to use a more precise term tree like structures. Clarification of what transformations on strings of symbols have to do with a development of computational design support in general case and in specific with the generation of operand transformation variants is closely linked with the studies of linguistically based knowledge representations done in AI. The aim of the previous Chapter was to provide an overview of computational design synthesis methods and tools showing that recent research more or less, tend to develop computational support based on formal grammars and languages. Although it could be sufficient to accept them as a foundation of this research, it seemed necessary to somehow relate current CDS efforts and the method for generation of operand transformation variants produced within this research with its theoretical foundations by explaining how formal grammars and formal languages came to be. This Chapter will try to lay out a formal point of view on grammars and languages which is necessary for understanding of method's definition in the Chapters to come. Moreover, the distinction between sequential depth-first and breadth-first grammars will be stressed out since this thesis assumes breadth-first derivation sequence. The Chapter will conclude with a small example of string formation in context-free language the grammars of which will be expressed in meta-language of Backus-Naur form [20].

4.1 Grammars, knowledge representation and engineering design

How to tackle the knowledge formalisation problem in a such manner that it can be effectively used and interacted or integrated is the key issue in the field of AI [67]. Rather

than embarking on the approach which aims at achieving increase in the performance by developing new types of hardware or more optimized search algorithms, the epistemological approach considers development of new and efficient ways of knowledge formalisation techniques to excel computationally performed problem solving. Efforts in the AI community done from 1950s to the end of the 1980s resulted in development of computational models of human cognition where the increase of performance of the respective model is to be achieved by an increase in the amount of the knowledge acquired [67]. Modelling of human cognition as an information-processing system [26], [14] was created as an analogy to computational systems; it could be emulated by computer. In fact the aim is to understand cognition thus creating platform independent models [67], and since computers are designed to operate in symbolic languages like humans do, they proved to be ideal test beds for the theories about human cognition.

The pioneering work, of course not related to knowledge formalization when it appeared in 1940s, were production systems which is a formalism conceived by Emil Post [68]. The Post production system in particular performed transformations on strings composed as sequences of symbols using a finite set of condition-action rules, or simply productions. Formal languages as a scientific discipline came to be as results of studies that Noam Chomsky performed in 1950s. In attempts to determine the basis and goals of linguistic theory he devised a formal model for the description of natural languages. Establishing of a formal basis was necessary to create a systematic approach to formation of scientific theory. Guided by the previous work of Post and others, Chomsky seeks out transformational model for language syntax as a mean for producing the sentences of the language under analysis [19]. In contrast to the semantics of a language that gives meaning to the sentences, grammar only determines the correctness of the form of sentences. Productions systems being nothing else but string based transformation systems provided principle for the creation of formal grammars. As a case study Chomsky defines a context-free grammar as at least adequate to capture formalisms of the English language grammar. In the early days of artificial intelligence and machine learning it was recognized that it was possible to tackle machine induction problems by means of formal grammar [70], [71] and [19]. The basic principle was that after learning an initial set of rules, a machine using a formal language would be able to create grammatical statements with the possibility to explain given phenomena beyond the set of given rules. According to Solomonoff, a machine could accept categories that have been useful in the past and than by means of a small set of transformations, derive new categories that have

reasonable likelihood of being useful in the future [70]. In 1970s, Minsky, Newell and Simon and others developed computational models of human cognition and problem solving where formalisation of knowledge was achieved on the basis of production systems and formal grammars. Problem solving process is modelled as a formal system comprised of intermediate problem-to-solution states which are transformed under effects of knowledge formalised within a set of condition-action rules. If the knowledge about domain can be encapsulated inside these set of rules, then a robust problem solving system could be created.

Translation of engineering design methods into a computational environment has a prerequisite in creating formal models of the method under consideration; a GDT as a logical description of design process is an example of such attempts. Design is a problem solving activity which exhibits formation of space of possible solutions and exploration of it using different search strategies [14], [68]. Decomposing a problem in order to reduce it to its constitutive elements or generating and testing solution candidates against the requirements by heuristic recombination of solution building-blocks are some examples of solution strategies, but how a person will perform and navigate through search space depends on the knowledge and experience of the person in a particular domain. Rather than rely on random walk, the knowledge about a task may narrow the search space, thus minimizing the number of candidate solutions. Creating analogies require out-of-domain knowledge so that designer can frame problem in respect to different contexts in order to identify similarities in structure and to ultimately produce an analogy. If the aim is to computationally emulate some of the more complex problem solving processes so that their prospects could be utilized for engineering design purposes, then a knowledge formalisation presents a necessity. For example, complex processes like learning and then usage of the acquired knowledge to expect and foresee a solution of the problem situation to which a person is confronted demand knowledge formalization of some kind which has to be embedded into a more flexible programming architecture, rather than hard coding of all the possible instances that might occur [26], [69]. Formalisation using a set of rules enables easy extension of the body of knowledge that has already been implemented by adding the new rules from within or from the outside of application domain. Up to recently, optimisation methods were usually those that were transferred to computer environment; such methods are most often numerical in-core, not aiming in simulating human performed problem solving. Most often these methods are limited in their scope in respect to design process and operate within limits of very concrete attributes of the product under consideration.

4.2 Production systems

Production system is a formal system designed to perform transformation of certain input to particular output using a set of condition-action rules that can be applied whenever conditions to do so have been met [72]. The set of condition-action rules is denoted as a set of production rules or simply productions; whereas a sequence of rule application is referred to as a derivation sequence. In case of strings, the production rule application is often denoted as rewriting. Control of which rule will be triggered is performed dynamically at run-time in respect to the current state of transformed object. Since the inference procedure is embedded into a set of rules performing transformation governed by condition-action principle, it can be said literally that a system's output has been produced rather than derived or inferred, what consequently resulted in naming the whole system as a production system. It is necessary to present this brief classification of production systems in general, since formal grammars are nothing more than a special type of production system.

According to Stiny and Gips, every production system can be categorised in terms of objects for which they are intended to transform, the way by which productions are defined, mechanism by which rules are applied, and finally in respect to objects that they generate [68]:

- Object types: initially Post's system transformed strings as sequences composed of symbols belonging to a specified fixed vocabulary. However, since production systems became of interest in other domains, like theory of computation, linguistics, automated text processing, image processing, biology and even within mechanical engineering and design, altogether resulted in a development of production systems that accept more complex types of objects including graphs, lists, trees and so forth.
- Definition of productions: the generic form by which production rules are defined is expressed as $\alpha \rightarrow \beta$, where α stands for the left-hand side of the rule denoting objects that will be replaced by the objects on the right side of the rule for whom β stands for. Moreover, production systems must also contain an object ω onto which the transformation is being applied. Elements that are belonging to fixed vocabularies, set of objects' building-blocks that is, are used to construct α, β and ω. In addition to fixed objects, both α and β may contain variables. Like specified in Post's production system, a production could be applied to string ω whenever strings could be assigned

to variables so that α is identical to ω [68] (for extensive explanation of Post production system see [73]).

- Rule application mechanism. Procedure starts with an initial object w which is then being transformed through consecutive steps using production rules of $\alpha \rightarrow \beta$. To apply a rule and perform the transformation, first the identification of any of α inside ω is performed. After all of the requirements have been met to identify α inside ω, the identified structure is subtracted from ω, and then on its place object(s) inside β are added and integrated. The production process stops after there is no more rules to apply to ω since none of its parts and ω as a whole do not match any of α. Post's production system which contains variables has an additional operation first of assigning values to variables and then if the match of object α to the whole ω is positive, the whole ω is being replaced with β. Not all of production systems perform in such manner, most often only the sub structures are being replaced in order to transform ω.

4.3 Grammars as production systems

As defined by Minsky [73] a formal language is a set of expressions formed from some given set of primitive symbols or expressions, by the repeated application of some given set of rules; formal language is than defined as primitive expressions plus the rules. Primitive expressions are the sentences of the language and can be infinite in numbers. Formal grammars are a type of production systems, aimed at describing linguistic structures of the language under consideration. Initially developed for modelling purposes within linguistic theory, they have found extensive use for defining syntax of programming languages which are artificial languages by which we communicate with computers. Syntax defines the formal relations between the constituents of a language, thereby providing a structural description of the various expressions that make up legal strings in the language without the consideration of their meaning [75]. This section will attempt to present some formal definitions of what formal grammars and languages are. The aim is to provide only what is necessary for this research, by skipping some of the intermediate steps. Moreover, various authors [66], [72], [74] and [75] tend to present same concepts in formal language and grammars theory as felt convenient to fit the intended purpose; this thesis adopts consolidated approach predominantly founded as given in [66].

The same set of letters of the Greek alphabet in lowercase which are already being used to explain principles behind general string based production systems, will be used again to define string based formal grammars; left-hand side of production rule will be denoted as α and the right-hand side will be denoted as β. String at arbitrary derivation step including initial step will be denoted as ω. The definition of formal grammar $\mathcal{G}$ is given as follows [66]:

DEFINITION 4.1 A grammar $\mathcal{G}$ is expressed as a quadruple $(\Sigma, \mathcal{V}, \mathcal{S}, \mathcal{P})$ where: Σ is a finite nonempty set of terminal symbols or alphabet, $\mathcal{V}$ is a finite nonempty set of non-terminal symbols or variables satisfying $\Sigma \cap \mathcal{V} = \emptyset$, $\mathcal{S}$ is a starting symbol or axiom with $\mathcal{S} \in \mathcal{V}$, and $\mathcal{P}$ is a finite nonempty set of production rules of the type $\alpha \to \beta$ where: $\alpha \in (\Sigma \cup \mathcal{V})^* \mathcal{V} (\Sigma \cup \mathcal{V})^*$ and $\beta \in (\Sigma \cup \mathcal{V})^*$.

Clarification of the former definition [74], [72]:

- Asterisk denotes all possible arrangements of elements contained within the set to which it is being applied including the element of zero length denoted with ε, $|\varepsilon| = 0$.
- Terminal symbols or simply alphabet are these which constitute all of the expressions within language.
- Variables or sometimes referred to as syntactic categories are symbols which are to be substituted during the derivation as specified by production rules.
- The set of all possible rules to which P belongs can be expressed using Cartesian product if considering each production as an ordered pair of α and β, thus:
$$\mathcal{P} \subset (\Sigma \cup \mathcal{V})^* V (\Sigma \cup \mathcal{V})^* \times (\Sigma \cup \mathcal{V})^*.$$
- Left-hand side of production α, or simply *head*, is a string that always contains at least one variable. Right-hand side of production β, or simply *body*, may contain any of the symbols.

DEFINITION 4.2 A formal language $\mathcal{L} = \mathcal{L}(\mathcal{G})$ generated by grammar $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{S}, \mathcal{P})$ is defined as:
$$\mathcal{L}(\mathcal{G}) = \{\omega | \omega \in \Sigma^*, \mathcal{S} \Rightarrow_{\mathcal{G}}^* \omega\}$$

Production to form a string in language is applied successively to all symbols of the string ω by substituting or simply rewriting variables and leaving terminal symbols as they were.

Derivation process $\mathcal{S} \Longrightarrow_{\mathcal{G}}^{*} \omega$ is conducted as series of productions under grammar $\mathcal{G}$ until initial symbol $\mathcal{S}$ is transformed into string ω consisting only of terminal symbols from Σ.

4.4 Classification of grammars and Chomsky's hierarchy

Grammars as formal systems can be categorised in respect to principles by which they operate, what kind of inputs they accept and what kind of outputs they produce. If input to a formal system is an initial symbol, a single start symbol, and the produced output is a sentence in the language defined by its respective grammar, then such grammar is referred to as generative grammar. Generative grammar will be thus used throughout this thesis for formalisation of operand transformation variants. Almost all of CDS methods intended for early product development support are generative grammar based (see overview in Table 3.1). On the other hand, accepting systems belong to the automata theory and are in opposite to generative grammar; they accept a sentence of formal language as an input and, at the output, it usually end with a stop symbol [72]. The Turing machine is a very well known example. Grammatical inference accepts a set of sentences written in the language under consideration and tries to determine the grammar of that language. Generative grammar can be differentiated in various ways; as deterministic or stochastic in respect to how to select among the productions, rewriting can be conducted sequential or in parallel, grammars can be parametric or non-parametric, pass attributes and so forth. However, the hierarchy devised by Chomsky which categorises grammars by stepwise introduction of restrictions to productions, shows in a practical way the capabilities of individual grammar for representing formal languages. Assumption is that the grammars are of course generative and that rewriting is performed sequentially. The types of grammars according to Chomsky are given in the Table 4.1:

Table 4.1 Chomsky's hierarchy of grammars [19]

Type	**Grammar**
type_0	unrestricted grammars
type_1	context-sensitive grammars
type_2	context-free grammars
type_3	regular grammars

Definition of grammars according to Chomsky's hierarchy is given as follows:

DEFINITION 4.3 Let $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{S}, \mathcal{P})$be a formal grammar of language $\mathcal{L} = \mathcal{L}(\mathcal{G})$ [66], then:

1. $\mathcal{G}$ is called unrestricted grammar or type_0 grammar if no restrictions are applied to productions.
2. $\mathcal{G}$ is called context-sensitive grammar (CSG) or type_1 grammar if each production in $\mathcal{P}$ satisfies that $|\alpha| \leq |\beta|$ resulting that no production can decrease the length of string. The productions of type $\mathcal{S} \to \varepsilon$ are allowed if no occurrences of head symbol exist on the right-hand side of any other production.
3. $\mathcal{G}$ is called context-free grammar (CFG) or type_2 grammar if each production in $\mathcal{P}$ in addition satisfies that $|\alpha| = 1$ resulting that productions are of form $A \to \beta$, where A is single variable.
4. $\mathcal{G}$ is called right-linear or regular grammar or type_3 grammar if each of the productions in P comply to any of three following forms: $A \to cB$, $A \to c$, $A \to \varepsilon$, where A and B are single variables and c is single terminal ($A = B$ is allowed).

Regular grammars might as well be left-linear, which intuitively results with quite the opposite formation of body of productions given as $A \to Bc$. It was necessary to present distinctions between grammar types since context-free grammars will be used for formalisation of operand transformation variants (see Chapter 6).

Relationship between different types of grammars is given with the following expression:

$$type_3 \subset type_2 \subset type_1 \subset type_0 \tag{4.1}$$

For example, any language defined with $type_2$ grammar is a subset of language defined with $type_1$ grammar and so forth according to expression (4.1). Of course, $type_0$ grammar can be used to describe any language according to proposed classification. However, that doesn't mean that language described with $type_2$ cannot be described by $type_3$ grammar if productions and restrictions allow that transition. In field of theoretical computation there are means to perform such transitions by eliminating occurrences of recursions in rules. Most common example is translation of CFG to regular expressions.

4.5 Backus-Naur Form

Backus-Naur form or simply BNF was initially developed for describing the syntax of *Algol 60* programming language [20]. BNF is another way to express the grammar of a language; in

fact the BNF is a language for itself thus considered as a meta-language for representation of other languages. BNF formalizes and simplifies syntactic expressions and since it is mostly unambiguous it allows construction of language parsers, or compiler-compilers, for a given BNF language grammar. A brief definition of BNF is given hereby, since the underlying search mechanism of method for generation of operand transformation variants will be based on grammatical evolution algorithm which is a genetic algorithm that operates within the bounds of BNF. An example of small context-free grammar in the BNF is given as follows (4.2). In the BNF notation the non-terminals are represented as tokens or phrases with <*non-terminal*> and the terminals are bracketless. The left-hand-side of the production rule is separated from the right-hand-side by the production sign : : =. If the right hand side of a rule has multiple alternatives then they are separated using the *or* sign.

$$
\begin{aligned}
&\Sigma = \{t_1, t_2, t_3\} \\
&\mathcal{V} = \{start, n_1, n_2\} \\
&\mathcal{S} = < start > \\
&\text{Production rule set } \mathcal{P}: \\
&< start > ::= < n_1 > \\
&\qquad\qquad | < n_2 > \\
&\qquad\qquad | < start >< n_2 >< n_1 > \\
&< n_1 > ::= t_1 \\
&\qquad\quad | \; t_2 < n_2 > \\
&< n_2 > ::= t_3 \\
&\qquad\quad | \; t_3 < n_2 >
\end{aligned}
\tag{4.2}
$$

The rewriting process starts from a predefined single symbol or axiom, defined by $< start >$ in (4.2). The initial symbol is rewritten in the first decomposition step, and then the process is repeated until the rewriting string is comprised only of terminal symbols. The examples of one possible derivation sequence assuming sequential depth-first left side rewriting (4.3) and then breadth-first rewriting (4.4) are given bellow:

$$
\begin{aligned}
< start > &\Rightarrow < start > \; < n_2 > \; < n_1 > \quad (1) \\
&\Rightarrow < n_1 > \; < n_2 > \; < n_1 > \quad (2) \\
&\Rightarrow t_1 \; < n_2 > \; < n_1 > \quad (3) \\
&\Rightarrow t_1 \; t_3 \; < n_1 > \quad (4) \\
&\Rightarrow t_1 \; t_3 \; t_1 \quad (5)
\end{aligned}
\tag{4.3}
$$

$$
\begin{aligned}
< start > &\Rightarrow < start > \; < n_2 > \; < n_1 > && (1)\\
&\Rightarrow < n_1 > \; < n_2 > \; < n_1 > && (2)\\
&\Rightarrow < n_1 > \; t_3 \; < n_1 > && (3)\\
&\Rightarrow < n_1 > \; t_3 \; t_1 && (4)\\
&\Rightarrow t_1 \; t_3 \; t_1 && (5)
\end{aligned}
\qquad (4.4)
$$

What can be easily noticed by comparing (4.3) and (4.4) to each other is that although the final sentence in the derivation step (5) is identical in both examples, the derivation process is not. The difference is noticeable in steps (3) and (4).

4.6 Implication to this work

Development of computational tools for engineering design support should include methods modelled according to human reasoning and problem solving and formalisation up to a limit that is convenient for computer application. Formal grammars can accomplish both of the requirements. Manipulating symbolic expressions composed as strings to convey meaning and to communicate is an inherited feature of human cognition. The computers were conceived in the same manner. Production systems and finite automata all initially operated with the sequences of strings, creating foundations for computation which held in-core of any computer's architecture. Advances in the theory of formal languages and grammars and the theory of categories have shown that all kind of objects can be transformed and manipulated as sequences of symbols, which qualified grammars as a powerful mean for knowledge formalisation being close and understandable to both humans and computers.

When speaking of grammars in the context of design and product development, it can be said that solving of a design problem can be achieved through means of grammars by prescribing finite sets of primitive knowledge building blocks used to form rules by knowledge has been formalised. The generation of good or feasible solutions is therefore analogous to the generation of grammatically valid statements using a formal language. This presents a pragmatic approach in contrast to computational simulation of cognitive processes and techniques that occur when person is designing by hard-coding such models inside the programming code. If the grammatical rules are logically sound, they can be continuously improved and expanded further either wholly computationally or with the assistance of the designer thus creating a robust problem solving mechanism. As Herb Simon [14] wrote down and expressed the most practical advantage of rule-based formal systems that rather than

adding new lines of fixed programming code, it is much more convenient to just add a new rule.

Each decomposition step of technical process into a set of interrelated operations and sub-processes must transform operands from the initial to the desired state obeying given constraints set for the transformation. Thus, the definition of the decomposition step in respect to engineering and technical process would be a single step that may consist of multiple derivation steps in which the whole transformation of operands from the input to the desired states should be performed. To use generative grammar to technical processes decomposition following the aspects must be met:

- A mapping from strings to graphs must be defined accompanied with additional connecting rules according to which nodes and sub-graphs will be integrated in the existing graph structures.
- At each decomposition step, a breadth-first rewriting starting from the leftmost operation must be performed until all of operations are rewritten or copied and transformation process is established.
    - For example, these which can be considered decomposition steps are derivations (1), (2), (5) in (4.4). In fact steps (3) and (4) in (4.4) are passing steps thus not increasing depth of decomposition. It is to assume that insights will be provided to engineer not only by the final all-terminals step as a sentence in the language of technical processes, but that also decomposition steps composed of variables and terminals mapped as graphs will provide information about transformation process. Of course, terminals that cannot be rewritten are simply added to new decomposition step.
- A rewriting of a graph node should always refer exactly to a preceding decomposition step and not to an arbitrary number of steps back to determine exact surroundings of a node or a sub-graph, thus requiring a breadth-first rewriting.
    - If considering a graph rewritings thus assuming existence of mappings from strings to graphs, then in the first column of (4.3) derivation (3) the determination of surroundings for node t_1 is done in respect to n_2 and n_1, while in (4.3) derivation step (5) the surroundings of the "same" node is performed in respect to t_3 and t_1. When transforming graph by parallel

rewriting, at each derivation step graph structure is produced dynamically both in respect to initial graph structure as a whole and to new emerging structure. Presented examples are simple; however in a larger custom-made grammar defined by a user, which is not necessarily more complex grammar, unexpected results may appear. Formal grammar usually is or should be defined non-ambiguously so that no matter what rewriting principles are applied the result should be in the same language produced by the same derivation process.

Questions left open at the moment are what type of grammar to use to formalise engineering knowledge about technical processes, and can such grammar be recursive. For the latter, if the knowledge is represented with the condition action rules, can the body of a rule contain the head of a rule? Grammar in (4.2) is a recursive grammar, and it is to assume that recursion will not be necessary to model engineering knowledge regarding decomposition of technical processes. The type of grammar used for string rewriting doesn't have to be in CFG but instead regular grammars can be applied relating more to engineering purposes. However, graph rewriting on the other hand is a context-sensitive process thus opting for a CSG.

5. GRAMMATICAL EVOLUTION

…and point out in addition the isomorphism between the genetic logic system, the logic systems of communication systems, the logic systems of computers, and the logic system in mathematics by which theorems are proved from a list of axioms. These systems may be regarded as being the same abstract system. (D. S. Ornstein; taken from H. Yockey's, Information Theory, Evolution and the Origin of Life [89]*)*

Evolutionary algorithms (EA) are population based stochastic optimisers that are built on mimicking the notions from the natural evolution. Charles Darwin wrote that evolution begins with the inheritance of good gene variations and that basically defines what the evolutionary algorithms are all about [90]. Enforcing the survival of the fittest principle is managed by allowing higher ranked solutions to influence the course of a search process by the most. By stochastically mixing together building-blocks which constituted two parent solutions an offspring is produced. If building blocks originate from higher fit individuals than there is a chance that newly generated individual might get a bit closer to a feasible solution of a problem. With the whole process repeated a population of offspring is produced from parent population. In fact the emergence of solution occurs as a consequence of a learning process that is exhibited by the algorithm since it tries to construct the optimal solution by arrangement of most fit building blocks.

The famous class of evolutionary algorithms, genetic algorithms (GA) resembles core principles of natural evolution by the most. Information exchange between solutions is performed by exchanging binary strings, or chromosomes, by stochastically invoking a recombination operator. To avoid pitfall of local optima low probability mutation operators for bit flip operators are introduced. Inside a genetic algorithm two levels of representation exist. At the genotype level, the solution representation is a binary string, at which the mechanisms of evolution operate with recombination and mutation. At the phenotype level the results of evolution manifest. Phenotype is realisation of genotype into problem specific form which is obtained by decoding the information stored inside a genotype. It is necessary to evaluate solution performance inside the given problem environment in order to enable fitness comparison between the individuals on the grounds of which the selection operates. As genotype, a solution is only a bit string of structured data that needs to be interpreted As

phenotype the solution is "full-grown" possessing different attributes thus representing behavioural level of an individual. Chromosome is structured as the collection of genes situated at their respective places or locus. If chromosome is a binary string, then gene is a binary substring which is recognized by decoding function according to its place on the chromosome. The applicability of algorithm to tackle discrete problems is devised from its recombination operators, which when generating new solutions, are reusing and mixing together pieces of the past solutions making it very useful when dealing with non continuous problems. Being robust and natively not calculus based, genetic algorithms are used as a good all-purpose optimization algorithm. The range of applications included scheduling, TSP class problems, tiling, close-packing, single or multi-objective optimization and so forth.

This thesis considers also a grammatical evolution (GE) based method for the search and optimization of the needed operand transformation for a selected technical process. GE is a population based heuristic search algorithm built up on GA which obtains a solution to a given problem by evolutionary means through recombination of the rule-based rewriting sequence [3]. The formalized knowledge regarding technical processes, technological principles and needed effects to support the main transformation of operands inside technical processes are stored in a set of production rules in Backus-Naur form (BNF) [20]. GE searches for the rule sequence that can perform the decomposition of the technical process black-box level to a structured system consisting of sub-processes, operations and operand flows according to TTS [1]. The following sections will try to explain similarities between design and evolutionary computation, which emerged as a paradigm of evolutionary design [11], [35], [81] and [88], where, design is to be understood in a broad sense including both the artefacts and the objects of natural origin. Generic EA model will be presented as a foundation to explain principles on which algorithm of grammatical evolution operates. An example of string derivation process in GE on CFG represented in BNF will be presented.

5.1 Recombination and Evolutionary design

The exploration of alternatives by combining chunks of past solutions can yield creative solutions; although the principle can be simple and not creative, the results could be quite the opposite. Goldberg's attempt of proving convergence of genetic algorithms published in his famous *Building Block Hypothesis* [35] was all about how iterative recombination of building blocks composed as strings of binary digits stems towards an optimum in a search process. If taken from a formal standpoint, to prove a heuristic process might be regarded as somewhat

dubious, however the scientific evidence confirm that transformation systems the principles of which are based on simple information chunk exchange involving a degree of randomness can generate creative and complex results. The study of cognition has shown that human achievements ranging from art forms like painting and music to scientific work involve some kind of building block recombination [26]. Maybe one of the earliest and according to some the most surprising examples is Shenker's theory of tonality made for Western music in 1935, where he using a set of rules expands initial motive into a complete musical composition [68]. In other domains like in engineering, one could create a data base of chunks of successful past solutions and by mixing those in a synergetic and holistic way, produce new innovative solutions can be produced; the example is Roth's catalogues. Even the processes that deal with artificial, but are of natural origins confirm the latter; information exchange during forming of amino acids is linear string like and digital involving finite number of building blocks which resemble the principles on which computers are being founded on [89]. Moreover, the recent advances in molecular biology confirmed that it is possible for broken sections of chromosomes to recombine and to change genomes to spawn new spices [76].

The second Chapter of this thesis has shown that engineering design is difficult to be described inside an algorithm, since it is an explanatory search process. However, let us assume that acceptable correlation can be established between design process and an iterative problem solving procedure involving a finite number of steps. One could define such a procedure as a search algorithm where the search space itself is built on lists of requirements or design variables and constraints. The feasible solution is being created by proposing solutions iteratively using suitable encodings acceptable both to the computational environment and to the problem of interest. From computational point of view an ideal algorithm candidate that would be able to carry such search process the be one belonging to the class of evolutionary algorithms. Evolutionary computation community expressed such claims frequently [11] and [35], since there really are some resemblances to design process with obvious one of being both evolutionary in respect to solution emergence. The design process modelled using EA's can be viewed as a shortcut to a satisfying technical product using knowledge and experience of designing in order to accelerate the technical development which naturally should occur evolutionary [77]. In fact because of the similar nature, the evolutionary methods may provide enhancement of design process or findings about process itself. Inside an EA, at each evolutionary turn solutions are being generated and tested by using selection, recombination and mutation operators and evolved as a result of exchange of

good building-blocks. Similar reasoning activities occur not only at designing but at any kind of problem solving situation. In respect to the properties of search spaces that depend on complexity of the design task being constrained, multimodal and full of discontinuities, they still can be handled since EA's rely on randomness inside mutation operators for their search.

5.2 Generic model of EA

Based on the same process of that they are mimicking, all classes of evolutionary algorithms are similar both in the structure and the behaviour exhibited during the search process. *Back et. al.* [34] have established following similarities between EA's:

- Evolutionary algorithms exhibit collective learning process.
- Each of the potential solutions to a problem is an encoded point in a search space which may hold additional information that can be used to enhance further search.
- Offspring population is created from parent population by random recombination which represents information interchange.
- Simulating error in information transcription may occur randomly thus introducing mutation.
- To converge towards a solution it is necessary to evaluate individuals therefore introducing a fitness function which assigns a real value to each population individual corresponding to how well it solves a given problem.

As a consequence of their findings *Back et. al.* [34] defined a finite set of classes of input parameters and evolutionary operators which perform transformations over populations, they formalised a fitness function which altogether resulted in the creation of a generic model of EA which is presented within a pseudo-code (5.13). Existence and influence of individual evolutionary operator, and relation between sizes of parent and offspring populations are used to unambiguously define a search process as being driven by genetic algorithm, genetic programming or evolutionary strategy. However, model as presented by *Back et. al.* didn't include classes of EA's where evolutionary operators, sizes of populations or evolution stopping conditions are functions of iteration step, generation, or some other specific parameter. Stochastic change of parameters which is measured over populations during the search process can be utilized to influence both the evolutionary operators and the fitness function tailoring them to best fit thus successfully guide the evolution process. Put

succinctly, in their model *Back et. al.* did not acquire with dynamical behaviour by generalizing it completely providing only basic outlook on EA's. Similarly, the generalisation at the level of selection strategies could be more complete since it only accounts for complete replacement of parents with their offspring. If that would be the case then the elitist selection strategy that favours a single best solution throughout generations until outranked would not be possible. *Merkle* and *Lamont* [82], [83] proposed a complete generalisation to overcome deficiencies of *Back's* model by introducing a random function based framework for evolutionary algorithms. Such framework enables to formalise occurrences where the operators are choosen stochastically, like in cases of genetic programming assuming both evolvable code and solution. To be strict, *Merkle* and *Lamont* have introduced evaluation as a part of selection operator, which differs with the common EA understanding. What is to be adopted from their model is a fitness function representation as a composition of functions containing objective function, decoding function and scaling function enabling to explain each of its parts in detail. A generic model of EA will be accepted as proposed by *Back* et. al. [34], but it will be supplemented by findings of *Merkle* and *Lamont* [82], [83] as required by the scope of this thesis.

5.2.1 Population

In general a population $\mathcal{P}$ is understood as a finite or infinite set of objects which can be enumerated. Building on the latter, let I, $I \neq \{\emptyset\}$ be a non-empty set of all possible potential solutions such that for every population member a it holds that $a_j \in I$, with $j \in \mathbb{Z}^+$. Population P of size $\mu \in \mathbb{Z}^+$ at generation $t \in \mathbb{N}$ can be defined with the following expression [34]:

$$P(t) = \{a_1(t), a_2(t), a_3(t), \dots, a_\mu(t)\} \in I^\mu \tag{5.1}$$

I^μ is a set of all populations $P(t)$ of size μ. Most often in EA community I is referred to as the individual space of an algorithm, and a is simply denoted as an individual. Depending on the type of individual and the purpose of the algorithm a population accepts any kind of objects. According to [82], [83], to allow dynamical alteration of population sizes it is possible to express them as functions of iteration step t as $\mu = \mu(t)$ and $\lambda = \lambda(t)$.

5.2.2 Fitness function

Fitness function expresses how an individual $a_j \in I$ satsifies the given problem. By structure fitness function equates objective function of standard optimisation, however due to specificities of EA's and different strategies that may be applied when evaluating the individual there exist additional mapping between the two. However, what remains in spite of differences is that ordering of population done over fitness of individuals is retained by objective function. Providing guidance to the evolution course, individuals with higher fitness tend to exchange their building blocks more often thus creating a collective learning process as a result of generating and testing. In general the objective function may be defined as:

$$f : \mathbb{R}^n \rightarrow \mathbb{R} \tag{5.2}$$

Expression (5.2) is given assuming n-dimensional search space. The set of all possible solutions I doesn't prescribe the type of individual, so for instance, genetic algorithms operate with bit strings while other may use real numbers or parse tree structures. In order to be able to map to real numbers it is necessary to apply additional mapping by using a decoding function. Introduction of another level of individual representation might add some weight to the complexity of the overall algorithm, but what is accomplished is the creation of a robust framework to tackle different kinds of tasks. What becomes problem specific is determination of individual encodings and decoding functions, where the former need to be designed as a best fit to describe the nature of a problem and to be adequate for direct application of evolutionary operators leaving them problem invariant. Assuming an n-dimensional search space, the decoding function D is defined as:

$$D : I \rightarrow \mathbb{R}^n \tag{5.3}$$

Let there be objective function Φ which maps a positive real number, or evaluation mark, to every individual $a_i \in I$ [34]:

$$\Phi : I \rightarrow \mathbb{R}^+ \tag{5.4}$$

Different strategies can be applied to enhance the performance of EA's. To prevent influence of a high fitness solutions which can sway the course of evolution especially, different mathematical functions T_s have been introduced to scale fitness in order to reduce the differences between solutions. In that way individual solutions that are less fit, get a chance to exchange their building-blocks. At the early stages, or early generations, it is difficult to

determine which building-blocks will compose the optimum since collective learning process has barely started, therefore scaling is necessary. Common example is ranking of solution for which functions of type $k \ln(n)$ can be used, where $k \in \mathbb{R}$ and $n \in \mathbb{Z}^+$ is ordinal number of individual inside population sorted over f. Scaling fitness function T_s is defined as follows:

$$T_s : \mathbb{R} \to \mathbb{R} \tag{5.5}$$

Finally, the fitness function $\Phi : I \to \mathbb{R}^+$ may be expressed as composition of functions following (5.2)-(5.5), [82]:

$$\Phi = T_s \circ f \circ D \tag{5.6}$$

The result of calculating fitness of each individual inside the population $P(t) \in I^\mu$ at iteration step $t \in \mathbb{N}$ will be denoted as evaluation or simply as $F(t)$:

$$F(t) = \Phi(P(t)) \tag{5.7}$$

5.3 Evolutionary operators

According to [82] and [83] evolutionary operators (EVOP) are defined as population transformations or mappings from populations to populations. A population transformation T is defined as given here by the following expression:

$$T : I^\mu \to I^{\mu'} \tag{5.8}$$

If $T(P) = P'$ than the expression (5.8) shows creation of offspring population $P'(t)$ of size $\mu' = \mu'(t) \in \mathbb{Z}^+$ from the parent population $P(t)$ of size $\mu = \mu(t) \in \mathbb{Z}^+$ at generation $t \in \mathbb{N}$. If the following holds $I^\mu = I^{\mu'}$, than no transformation occurs resulting with $\mu = \mu'$ as the size of population.

Following (5.8) evolutionary operators; selection (5.9), mutation (5.10) and recombination (5.11) are defined over individual space as the following transformations:

$$s : I^\lambda \to I^\mu \tag{5.9}$$

$$m : I^\kappa \to I^\lambda \tag{5.10}$$

$$r : I^\mu \to I^\kappa \tag{5.11}$$

Different types of EA's can be applied by comparing domains and ranges of each transformation as given in (5.9)-(5.11), three possibilities are such as (5.12):

$I^{\lambda} \neq I^{\kappa} \neq I^{\mu}$	Genetic algorithms, evolutionary strategies with $\mu, \lambda > 1$.	
$I^{\kappa} = I^{\lambda}$	No mutation.	(5.12)
$I^{\kappa} = I^{\mu}$	No recombination, evolutionary programming, initial evolutionary strategies with $\mu = \lambda = 1$.	

For each of evolutionary operators there exists a set of parameters on which EVOPs depend on. Let Sets of parameters are $\Theta_s, \Theta_m, \Theta_r$ with subscript denoting their respective operator. Set Θ_ι contains additional real valued parameters necessary for execution of the algorithm, and ι is a logical test or stopping condition which terminates the search loop. Finally, pseudo-code of evolutionary algorithm according to [34] and assuming $I^{\lambda} \neq I^{\kappa} \neq I^{\mu}$ is given as follows:

Input: $\mu, \lambda, \Theta_s, \Theta_m, \Theta_r$
Output: a^*, P^*

1 $t \leftarrow 0$;
2 $P(t) \leftarrow initialisation(\mu)$;
3 $F(t) \leftarrow evaluation(P(t), \mu)$;
4 **while** $(\iota(P(t), \Theta_\iota \neq true)$ **do** (5.13)
5 $P'(t) \leftarrow recombination(P(t), \Theta_r)$;
6 $P''(t) \leftarrow mutation(P'(t), \Theta_m)$;
7 $F(t) \leftarrow evaluation(P''(t), \lambda)$;
8 $P(t+1) \leftarrow selection(P''(t), F(t), \mu, \Theta_s)$;
9 $t \leftarrow t + 1$;
od

Beginning with the initialization which is in fact a random sampling of μ individuals from I, an initial population $P(0)$ is being formed, (2). Following the first evaluation (3) algorithm enters a do-while loop executing it until satisfying the termination condition defined as $\iota(P(t), \Theta_\iota \neq true)$ (4). Offspring population $P'(t)$ and the mutated offspring population $P''(t)$ are determined under the recombination (5) and mutation (6) operators. After the evaluation of the $P''(t)$, (7), μ individuals are selected to create population that will continue the search (8). Asterisk marked individual a^* or population P^* refer to best solutions found during the search. Within this thesis a full GA will be necessary extended as GE to operate with BNF as a mean to formalise graph grammar transformations.

5.4 Grammatical Evolution

Grammatical evolution was initially developed to write computer programs in any language [3], [78] and [79]. Depending on the explicit purpose, both the programs and the rewriting rules can be evolved accordingly. The grammar of a particular formal language is expressed as a collection of rewriting rules in Backus-Naur form (BNF). For the search of an optimal rule-based rewriting sequence, GE relies on an embedded genetic algorithm (GA). In that way, GE inherits the robustness of GAs. The mechanism for creating grammatically valid statements is achieved through mapping from a binary encoded chromosome to the sequence of rewriting rules, thus introducing another layer into a decoding function. Phenotype emerges after the sentence in a language $\mathcal{L} = \mathcal{L}(\mathcal{G})$ has been created and evaluated. However, as in the case of this thesis, additional mappings both from variables and terminals into graphs is necessary, as BNF tokens serve only as a graph rewriting symbols. The search uses the concept of survival of the fittest, where a solution or a set of solutions to a given problem evolve in time using the fitness function as an evolutionary guide. The same grammar $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{S}, \mathcal{P})$ given in BNF as in (4.2) will be used to explain how GE operates. Since it is necessary to introduce enumeration of rule alternatives per rule required by GE search mechanism, an extension of (4.2) is given below:

$$
\begin{aligned}
&\Sigma = \{t_1, t_2, t_3\} && \text{Rule} \\
&\mathcal{V} = \{start, n_1, n_2\} && \text{alternativ} \\
&\mathcal{S} = < start > && \text{e per rule} \\
&\text{Production rule set } \mathcal{P}: \\
&< start > ::= < n_1 > && (0) \\
&\qquad\qquad | < n_2 > && (1) \qquad (5.14) \\
&\qquad\qquad | < start >< n_2 >< n_1 > && (2) \\
&< n_1 > ::= t_1 && (0) \\
&\qquad\qquad | \; t_2 < n_2 > && (1) \\
&< n_2 > ::= t_3 && (0) \\
&\qquad\qquad | \; t_3 < n_2 > && (1)
\end{aligned}
$$

Let an individual $a_j \in I$ be represented as l-bit string composed of concatenation of $n \in \mathbb{Z}^+$ substrings δ_i of unique lengths $l_i \in \mathbb{Z}^+$. [88]:

$$a_j = \sum_{i=1}^{n} \delta_i \qquad (5.15)$$

Each gene carries information necessary for decoding function to create solution phenotype. Depending on their length each of the substrings occupies a space $\delta_i \in \mathbb{B}^{l_i} = \{0,1\}^{l_i}$. The mapping to a real value $x_i \in [u_i, v_i]$, with $x_i, u_i, v_i \in \mathbb{R}$ for each substring δ_i at position $i \in \mathbb{Z}^+$ is performed by standard binary decoding function D_{B_i} [88]:

$$D_{B_i}: \mathbb{B}^{l_i} \rightarrow [u_i, v_i] \tag{5.16}$$

The decoding function γ_i is a simple mapping of a binary coded string into $x_i \in \mathbb{R}$ that is then normalized by desired mapping range $[u_i, v_i] \in \mathbb{R}$. Decoding function γ_i is given as follows [88]:

$$x_i = u_i + \frac{v_i - u_i}{2^{l_i} - 1}\left(\sum_{j=0}^{l_i-1} \delta_{i(l_i-j)} 2^j\right) \tag{5.17}$$

Concerning GE, the usual desired range is a closed interval of nonnegative integers set within $[0,\ 255]$ thus yielding $x_i \in \mathbb{N}$. Complete decoding that produces a vector $\boldsymbol{x}$ of integer values by repeating the (5.16) and (5.17) to whole chromosome composed of $i \in \mathbb{Z}^+$ genes [88] is given here by:

$$D_B = D_{B_1} \times D_{B_2} \times \ldots \times D_{B_i} \tag{5.18}$$

To achieve exact one to one mapping in the desired interval $[0,\ 255]$ each binary string is 8 bits in length. If multiple alternatives for rewriting a non-terminal symbol in α exists, then the selection of body of the $\alpha \rightarrow \beta$ productions has to be determined. Such grammar is a stochastic generative grammar since application of rules depends on randomness embedded into genetic algorithms. The application of rule r_i for rewriting is calculated as a function of the decoded integer $x_i \in [0,\ 255]$ of the gene i and the number of the available rule alternatives $n_i \in \mathbb{Z}^+$ [3]:

$$r_i = x_i \bmod n_i \tag{5.19}$$

To clarify by means of an example: regarding the initial rewriting of the $< start >$ symbol from CFG in (5.14) and decoded chromosome values as shown in Table 5.1, the total number of rewriting alternatives for initial rewriting equals $n_1 = 3$ and the first decoded value equals $r_1 = 8$. The equation (5.19) yields the solution 2 for solution, thereby triggering the last rule alternative $< start >< n_2 >< n_1 >$.

Table 5.1 Decoded chromosome and rule sequence selection

i	1	2	3	4	5	6	7	8
x_i	8	120	33	42	32	39	124	48
r_i	2	0	0	0	0	/	/	/

For the sake of simplicity, the binary genotype has been left out of Table 5.1 and only the decoded integer values x_i are shown. It is assumed that the chromosome is composed of 8 genes. The selection of the rule alternative r_i is calculated with the expression (5.19) based on the grammar shown in (5.14).

Example of breadth-first derivation sequence done by GE that would equate the result of example (4.4) is given bellow:

$$
\begin{aligned}
< start > &\xrightarrow{8 \bmod 3=2} < start > \; < n_2 > \; < n_1 > \\
&\xrightarrow{120 \bmod 2=0} < n_1 > \; < n_2 > \; < n_1 > \\
&\xrightarrow{33 \bmod 3=0} < n_1 > \; t_3 \; < n_1 > \\
&\xrightarrow{42 \bmod 2=0} < n_1 > \; t_3 \; t_1 \\
&\xrightarrow{32 \bmod 2=0} t_1 \; t_3 \; t_1
\end{aligned}
\tag{5.20}
$$

Note that the genes ranging from 1 to 5 are the only ones used and the rest of the chromosome is left unused because an all-terminal state has been reached and no more rewritings are possible. This is in contrast to usual evolutionary approaches, where the entire chromosome is almost always used. With GE there is a possibility of creating redundant information. In the opposite case, if the entire genetic material is used for triggering the BNF rules, the gene reading process starts over again from the first gene. This process of reusing information to achieve a state consisting only of terminal symbols with complete mapping is known within the GE community as *wrapping* [74]. The number of allowable reading runs for mapping to BNF rules is a set using a *wrapping operator*. Both the existence of the redundant genetic information and the re-use of the genetic information occur in living organisms [89].

In general, with, GE on the phenotype level, each solution should be represented with a string consisting of a set of terminals. However, like with the cellular automata [92], a set of rules can produce a rewriting process that is indefinite or very large but with a finite number of decompositions that must be performed before achieving the all-terminals state. Example of

CFG in (5.14) can produce indefinite rewriting sequence. For practical purposes, indefinite or too large sentences yielding in a very well known spaghetti effect, render the solution in an unacceptable manner. Such occurrences are the direct consequence of the size and quality of the utilized grammar. To prevent these occurrences, a special kind of stopping rule needs to be introduced in the form of a maximal derivation step constraint. After reaching the specified number of steps the rewriting process will stop, declaring the finding of a solution unfeasible.

5.5 Extension of fitness function

The specifications of GE in conjunction with the method for operand transformation variants require an extension of fitness function as given in (5.7). The first one is restriction of the individual space of algorithm as in respect to $i \in \mathbb{Z}^+$ substrings of unique lengths $l_i \in \mathbb{Z}^+$ composed of binary alphabet $\{0, 1\}$, thus accepting expression for D_B given in (5.18):

$$I \equiv I_{TP} = \mathbb{B}^{l_1} \times \mathbb{B}^{l_2} \times \ldots \times \mathbb{B}^{l_i} \tag{5.21}$$

Let there exists $D_{\mathcal{L}(\mathcal{G})}$ that maps from $x_i \in [0,\ 255]$, $x_i \in \mathbb{N}$ to sentences in formal language $\mathcal{L} = \mathcal{L}(\mathcal{G})$ defined with context-free formal grammar $\mathcal{G}$ as in (4.3). Mapping within an ordinary GE is thus described as:

$$D_{\mathcal{L}(\mathcal{G})} : [0, 255] \ \rightarrow \ \mathcal{L}_{TP}(CFG) \tag{5.22}$$

Let us assume that there is a mapping $D_{\mathcal{L}_{TP}(\mathcal{GG})}$ that maps from Chomsky's strings to a language of technical processes $\mathcal{L}_{TP} = \mathcal{L}_{TP}(\mathcal{GG})$ defined with graph grammar $\mathcal{GG}$:

$$D_{\mathcal{L}_{TP}(\mathcal{GG})} : \mathcal{L}_{TP}(CFG) \rightarrow \mathcal{L}_{TP}(GG) \tag{5.23}$$

Due to (5.18), (5.22), (5.23) the objective function exhibited domain change resulting with following f_{TP} with $n \in \mathbb{Z}^+$ objectives:

$$f_{TP} : \mathcal{L}_{TP}(\mathcal{GG}) \rightarrow \mathbb{R}^n \tag{5.24}$$

Finally, the fitness function for individual $a_j \in I_{TP}$ representing a technical processes is an extension of (5.6), resulting with the following expression for $\Phi_{TP}: I_{TP} \rightarrow \mathbb{R}^+$:

$$\Phi_{TP} = \ T_s \ \circ f_{TP} \ \circ D_{\mathcal{L}_{TP}(\mathcal{GG})} \circ D_{\mathcal{L}(\mathcal{G})} \circ D_B \tag{5.25}$$

Extending the fitness function to describe a method through mappings is much more convenient than definition of completely new evolutionary operators. To tailor it both to GE

and search of operand transformations, the effective change in pseudo-code (5.13) is just applying population evaluation using $F(t) = \Phi_{TP}(P(t))$ according to (5.25). In that way, a GE layer and graph grammar transformations in the language of technical processes are simply added to GA. The condition $I^{\lambda} \neq I^{\kappa} \neq I^{\mu}$ from (5.12) is applied to account for GA.

5.6 Implications to this thesis

One of the key issues [2] is how to approach problem definition side of an algorithm which always remains as a fixed part of search processes. Creating dynamical fitness functions may help deal the problem of design task re-formulation. The evolutionary computation community often pointed out that in order to go beyond current successes in optimisation, that it is necessary not only to evolve the problem statement parametrically as assignment of values to means, but that it should be reframed if possible. Design process considers evolution of both the design task and the problem. Therefore, the fixed encoding which is a direct consequence of the algorithm should be expanded at least in a direction for which it can be predicted that will proove significant for formation of solutions. In that way navigating to more detailed design solutions by expanding the set of means to realize behaviour can be achieved. However up-to-date efforts stop at optimization directed parameterization to enhance fitness functions dynamically and few attempts of encoding alterations [37].

The reasons why grammatical evolution in particular can be used as a foundation for development of computational support for early product development phases are argued as follows:

1. As it is built on an embedded genetic algorithm (GA), grammatical evolution inherited GA behaviour and robustness rendering it applicable for a wide range of problems. Moreover, since GE is GA on the genotype level having a chromosome representation as binary strings, straightforward application of the entire standard GA selection, crossover and mutation operators and various multi-objective genetic algorithms are possible.
2. Since grammatical evolution is a stochastic optimizer, then it can search for the sequence of BNF rules through which the optimal decomposition can be generated rather than just generating all of the possible solution variants. Initially GE was developed to write computer programs in any language. Depending on the explicit purpose, both the programs and the rewriting rules can be evolved accordingly. The

grammar of a particular formal language is expressed as a collection of rewriting rules in Backus-Naur form (BNF) defined over a finite set of symbols or tokens. The same BNF system will be applied to formalise engineering knowledge about technical processes first using string CFG and than extending it o graph grammars.

3. Derivation process through which grammatically valid sentences are generated inside GE is analogous to the activity of decomposition performed by designers at the early stages of the conceptual design phase. Both processes, the decomposition and grammar derivation using BNF type rules, can be represented as a tree structured on a parent-child relationship. Applying of BNF production rules creates a parent-child relationship between rewritten symbols in successive derivation steps. According to TTS market demands and societal needs inside technical process are modelled using a black-box concept with operands in their states and desired states before and after the transformation. Then the decomposition of the black-box is performed step-wise into the systems of interrelated sub-processes and operations in respect to knowledge about processes and technological principles on which these processes are based on. Breaking of a complex problem into a system of smaller interrelated problems, thus synthesising a transformation system is necessary for designers to establish and consider different product realization possibilities in respect of various effects that need to be delivered in order to sustain a technical process. In the same manner GE performs step-wise derivation starting from an initial symbol and rewriting it to sequence composed of symbols from vocabulary following set of production rules in BNF. Decomposition performed by designers stops after black-box has been decomposed into a system of operations after which further decomposition would be meaningless, what corresponds to GE stopping rules where rewriting of a symbol cannot be accomplished if that symbol is a terminal one.

4. A *closure* problem is avoided since the BNF production rules are applied always as in response to the symbol that is to be rewritten with evolutionary operators being aimed at selecting a variant of a rule that is able to perform a rewriting. That is in contrast to genetic programming where recombination and mutation can result in incompatible building-blocks which when mixed together yield in unfeasible solutions.

5. By following a holistic paradigm TTS relies on a systemic modelling approach used for representing technical processes as transformation systems and products as technical systems. It has been shown [94] that equivalence exists between the

Chomsky's grammars which are in fact used by GE and graph grammars enabling to utilize BNF type rules and context-free grammar to define graph transformations rules. As a result, a robust token based rewriting system is created which is easier for computer implementation. Technical process as an operand transformation system is then modelled as a token composed sentence which has to be interpreted as graph accordingly. The whole decomposition process of technical processes into a system of sub-processes and operations interrelated with operand flows is in fact a successive graph transformation process which is for the purpose of the presented computational method performed by a symbol rewriting system.

6. An important property of GE is that if elevated to a meta-level GE, it can infer new rules. Meta-level GE is referred to as grammatical evolution by grammatical evolution or GE2 [28]. By recombination of the existing rules new rules are generated. The extension of the presented method to include the possibilities of GE^2 will be explored in future work and may provide the true advantage of using GE. If considering Computational design synthesis (CDS) it can be concluded that computational support of design activities such as decomposition and search can be achieved using GE and GA. However, since synthesis also involves design activities such as associating, composing and combining most likely is that the machine inductive reasoning would be necessary in an abductive iterative process in which solution is affecting problem statement. For that purposes GE^2 could be used.

Therefore, GE is a population based algorithm, the search is build around a concept of survival of the fittest, where a solution or a set of solutions to a given problem evolve in time using the fitness function as an evolutionary guide. At the genotype level, the chromosome representation is a binary string, thus enabling easy application of the entire standard GA selection, crossover and mutation operators.

It is to assume that powerful stochastic search like GE will not be fully utilized unless the complexity of technical process offers combinatorial explosion, which may occur only if considered that technical system or process involving industrial plant, ship or similar. However, the intention is to lay foundations for development of complete graph grammar based framework for support of early design phases. GE as a robust problem solver based on GA which can be applicable as a search method for any of the phases in the early design assuming that engineering knowledge is formalized using graph grammars. Almost all of design theories are following systemic reasoning thus resulting with early design modelling

tends towards transformation systems. When developing computational support it would be natural to merge graph grammars with grammar based stochastic search algorithm. Moreover, the method that will be presented in the next Chapter is invariant to the level of abstraction considered the early design stages. Although the search for innovative technologies for operand transformation is considered only for the design of completely new products, the intention is to provide the basis on which the functional structure of the product can be determined.

6. GRAMMAR OF TECHNICAL PROCESSES

In the beginning is the relation (Martin Buber, philosopher, 1878 – 1965, from I And Thou)

Both formally and visually, graphs are widespread as means to express and model various types of systems in order to represent their structure and behaviour. The application range includes entity relations diagrams, UML diagrams, Petri nets, flow diagrams, identification of software product structures, modelling graphical interfaces [95] and last but not the least, graph representations are utilized at least implicitly by different design theories [1], [4] to model technical products at early stages of product development process. In the computing graph based formal systems, like for instance parse trees and term rewriting systems are unavoidable as means for accepting and inferring syntax and semantics of sentences written in programming languages. Further uses are found among more dynamical graph transformation systems developed with the purpose of expressing behaviour of an evolvable system that is under consideration, and as such the most applications include search for optimal resource allocation possibilities both in economics and computation [95], [99]. Thus, decomposition of technical processes performed to synthesise optimal variant of operand transformation modelled according to TTS could be considered as a graph transformation system.

Graph grammars are means to perform a rule-based transformation of graphs. The application of rule first identifies a target structure, a sub-graph that is, inside a host graph, which has to be replaced by a new sub-graph. As the result of deletion of the old and integration of the new sub-structure with the remainder of original graph a transformed graph structure emerges. Thus, bearing in mind advances from AI showing that knowledge from the domain of interest can be formalised within grammars, than decision to design the method for generation of operand transformation variants based on graph grammar transformations becomes well justified. Development of graph transformation systems emerged from three different application areas: from Chomsky's strings grammars which initiated the theory of natural languages by providing it with formal foundations, from term rewriting systems and the theory of computing and programming languages, and to account for enhancement of modelling processes by providing visual interfaces rather than textual ones [95]. This thesis is drawn by the former, thus using Chomsky's context-free string grammar expressed within BNF. A mapping will be established by linking BNF tokens to graph structures thus creating a

graph grammar based formal system over which a heuristic stochastic search of grammatical evolution will be imposed. Such system, providing an adequate and consistent knowledge formalisation, should be able to generate operand transformation variants within technical processes.

This Chapter will deal with the formal definition of graph grammar of technical processes. It will not present new method for conducting general graph transformation, instead it will offer an adaptation of graph grammar node based transformation to suit the purposes of technical process modelling. Moreover, it will add heuristic search to graph grammars in order create a framework suitable for multi-objective optimisation. The single node graph grammar transformation algorithm will be presented, and the implications of the connecting rules to the knowledge formalisation possibilities will be elaborated.

6.1 Method overview

Decomposition and synthesis of technical processes until operand transformation variants are produced will be defined as a formal system by means of graph grammar containing rule represented engineering knowledge of technical processes, technological principles and necessary effects. Production rules within graph grammar will prescribe the mechanism and conditions that must be satisfied in order to conduct decomposition and synthesis process. For the method to be operational, and to be able to conduct graph grammar transformations, an adequate mathematical modelling of technical process must be provided. Thus, a technical process will be defined as a labelled directed multigraph, or multi-digraph with operands effects and operations. The method for generation of operand transformation variants within technical processes, as presented using IDEF0 process model is shown in Figure 6.1:

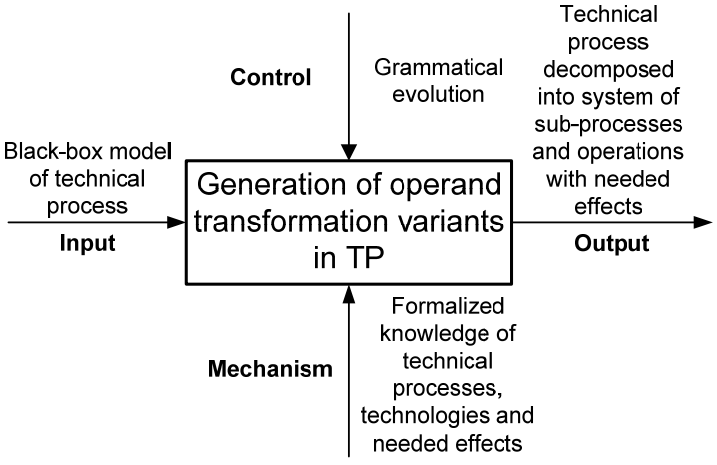


Figure 6.1 IDEF0 model of generation of operand transformation variants within TP

The proposed method relies on the grammatical evolution for controlling and directing the search which requires establishment of a link between Chomsky's string grammars and graph grammar. The result of merger is applying a breadth-first node based rewriting system. The term rewriting is applied hereby, since if the map between symbols within strings and graphs nodes is achieved then it can be said that rewriting takes place, of course, providing necessary embedding mechanisms. Thus, to perform decomposition a breadth-first node based rewriting will be performed with embedding mechanisms as determined by applying a set of connecting rules. That set of connecting rules will help to account of to Hubka's law by integrating secondary flows into graph's structure. Because GE uses genetic algorithm three encodings will exist, as defined in (5.21)-(5.25); genotype level as binary strings, GE intermediate level as BNF token strings, and at upmost level to allow grammatical evolution to conduct the search for optimal transformation alternatives graph structures will emerge.

The same principles used for generation of operand transformation variants can be extended to include other stages and phases of product development providing of course the existence of design language. Product functions are also both visually and formally represented with graphs however function unlike processes, they cannot be considered as sequences of operations since functions represent technical system in a state in which it is ready to operate or be operated. There is no time flow like with processes. The entry point to functional decomposition is the output of TP level, and the result is a technical system being in the state capable to produce the necessary effects. Organism as a technical system is a composition of its interrelated subsystems or organs, where each of them exhibits its own unique graph like structure in which the necessary *wirk* elements appear to fulfil one or more products functions. The establishment of technical system's organs concludes conceptual design stage by providing a sketch, a concept of product that is. In general, a generative grammar driven approach, like the one presented within this thesis, is a process that is conducted in a one way top-down manner. Solution, alternatives can be produced, but after the rewritings have been exhausted the search process inevitably ends. However, since synthesis of technical process and technical system at different abstraction levels should establish relations that extend between these and in fact interconnect elements belonging to different levels thus providing additional semantic meaning to explain and provide behaviour of product as a complex system, then an iteration should be created resulting in top-down and bottom-up refinement scheme. Complex behaviour cannot be represented simply within set a of rules governing the search at only one level of system's representation, but must extend to relate the system

elements between the levels. Creation of these inter-level relationships, which originally occur within designers mind, might be consider as the bottom-up part of the loop. And finally considering other stages of product development, embodiment and detailing namely, assemblies and subassemblies of technical systems are also both structurally and behaviourally describable using graphs. Assemblies are represented as trees of components and parts as trees of features and from them derived drawings, what is indubitably seen within a today's feature based design CAD packages. Although this thesis is limited by its scope to technical processes, the findings provided can be utilized further to create a complete framework driven by unified design language.

6.2 Modelling of operand transformation system

Multigraphs are considered as non-simple graphs in which multiple edges between vertices, i.e. nodes, are allowed but no loops are permitted [91]. In general, a multigraph G can be defined as an ordered pair (V, E), where V is set a of nodes and E a bag of edges. If a direction is required to represent binary relation between the vertices, than edges are replaced by directed edges or put succinctly by arcs. To model technical process formally it is necessary to introduce related technical process entities; operands, effects and operations namely, into a graph's structure. Hence, operations will be mapped to graph's nodes, where operands and effects are mapped to arcs.

The definition of set of *TP entities* Σ_G is given as follows:

DEFINITION 6.1 Let there exists a set of *TP entities* defined as $\Sigma_G = \Sigma_{Od} \cup \Sigma_{Eff} \cup \Sigma_{Op}$, where Σ_{Od} denotes a finite non-empty set of operands $Od \in \Sigma_{Od}, \Sigma_{Eff}, \Sigma_{Eff}$ is a finite non-empty set of effects $Eff \in \Sigma_{Eff}$ and Σ_{Op} as a finite non-empty set of operations $Op \in \Sigma_{Op}$.

In general, depending of the purpose graphs can be defined over different sets of objects, strings, types and instances of these thus relating starting graph structures to other graphs. The set of *TP entities* $\Sigma_G = \Sigma_{Od} \cup \Sigma_{Eff} \cup \Sigma_{Op}$ denotes entities that participate within technical process. Thus in order to achieve robust mathematical modelling, multidigraph as presented within this Chapter is not restricted to accepting *TP entities* as objects. The same holds for the graph-grammar transformation algorithm that is also presented within this Chapter. Moreover, as it will be later shown in Chapter 8 where the architecture of computational tool is presented, data model of *TP entities* will be performed object-oriented, thus defining entities

as class instances. However, although both mathematical and data modelling are robust, Chapter 7 lay out only foundations for formalisation of the knowledge about technical processes. The goal of this thesis wasn't by any means to qualitatively define technical process taxonomies or ontologies which could have served well for generalisation of production rules and transformation system. The latter is the reason why the multidigraph is referred as labelled and not as typed multidigraph denoting that only labels of *TP entities*, alphanumeric strings that is, are considered although *TP entities* are types.

TP entities will be defined by the user when formalizing knowledge about technical processes, technological principles and necessary effects. Let operand Od be an element from a finite set of all possible operands Σ_{Od}, $Od \in \Sigma_{Od}$. Operand as a type has *ID*, *name*, *state, states* and *label* as attributes. In the same manner effect Eff is an element of a finite set of all possible effects Σ_{Eff}, $Eff \in \Sigma_{Eff}$ having *ID, name* and *label* for attributes. Hence, in Chapter 8 data model of both operands and effects is *Flow* class, and edge of multidigraph is an object container which can accept both operands and effects. The same reasoning is applied with the graph vertices where each vertex is a container accepting operation Op, where $Op \in \Sigma_{Op}$ and Σ_{Op} is a finite set off all possible operations. Op is an object with *ID, name, label* and three collections of *input operands*, *output operands* and *effects* as attributes. Most common, symbol Σ denotes set of terminal of symbols, a language alphabet that is. Decomposition which synthesises technical processes lasts until all appropriate rewritings have been utilised which is only dependent on the amount of knowledge that has been formalised. Thus, there is no sense to give the usual meaning to Σ_{Od}, Σ_{Eff}, Σ_{Op} as being variables or terminals, since they are used as building blocks to define graph grammar production rules. In respect to Σ_G a multi-digraph $G$with operands effects and operations may be defined in Definition 6.2.

Figure 6.2 shows an arbitrary structure of technical process modelled as labelled multi-digraph. Operands $Od_1, \dots, Od_7$ can be understood as operands of different types (classes), as operands of the same type but in varying states or both all of them represented as labels. Some of these operands (Figure 6.2) may be the operands the transformation of which directly satisfies the existent users' needs, and some may emerge secondary as required or generated by transformation system. Operations are represented with graph nodes labelled as $Op_1, \dots, Op_3$. $TrS\ Input$, $TrS\ Output$ and $TrS\ Effects$ are represented by labelling of nodes with *in, out* and *eff* respectively.

DEFINITION 6.2 A labelled multi-digraph G with operands, effects and operations with no loops allowed is defined over alphabet Σ_G as ordered tuple $G = (V, E, s, t, l_E, l_V)$:

- V finite non-empty set of nodes,
- $E \subseteq \{(u, v) | u, v \in V \ \wedge \ u \neq v\}$ finite non-empty bag of arcs e, with restrictions to loops
- mapping $s: E \rightarrow V$ assigning for each arc e a source node u,
- mapping $t: E \rightarrow V$ for each arc e assigns a target node v,
- mapping $l_E: E \rightarrow \Sigma_{Od} \cup \Sigma_{Eff}$ which for each arc e assigns operand Od or effect Eff,
- mapping $l_V: V \rightarrow \Sigma_{Op}$ which for each arc e assigns operation Op.

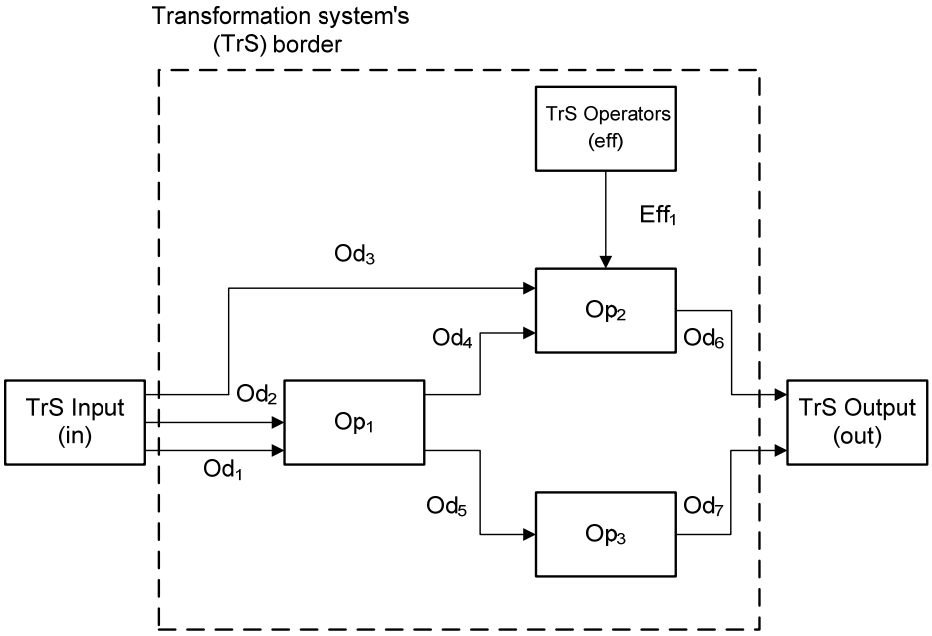


Figure 6.2 An example of TP modelled by $G$with TP entities

These are added to graph's structure to represented flows crossing the systems borders and the source of the effects within the transformation system. Sources of effects are operators: human, technical system an environment, however to simplify the modelling all of them are represented with only one node. Operations Op_2 and Op_3 are performed in parallel, thus creating a sequence when coupled together with Op_1.An effect delivered by transformation system's operators is required to enable Op_2 is shown as Eff_1. Returning operand flows, i.e. arc e from Od_2 to Od_1, are supported by the model but are not permitted by TTS since they would violate the time flow inside transformation process. Returning flows are thus implicitly

omitted through productions rule definition process. Relations of type $s(e) = t(e)$ are not permitted. Thus, additional rules that have to account for modelling of technical processes are given here by:

DEFINITION 6.3 A labelled multi-digraph $G = (V, E, s, t, l_E, l_V)$ as defined in 6.2 is a model of technical process *iff* at least the following is satisfied:

5) $|V| \geq 4$,
   - $\exists_1\, v \in V |\, l_V(V) = in \wedge$
   - $\exists_1\, v \in V |\, l_V(V) = out \wedge$
   - $\exists_1\, v \in V |\, l_V(V) = eff.$
6) restrictions to relations *E*:
   - $\nexists\, e \in E |\, l_V\big(t(e)\big) = in \wedge$
   - $\nexists\, e \in E |\, l_V\big(s(e)\big) = out \wedge$
   - $\nexists\, e \in E |\, l_V\big(t(e)\big) = eff$
7) graph G has to be well connected in respect to the transformation of Op.

To clarify the definition 6.3; 1) requires existence of minimally one operation node, alongside *in*, *out* and *eff* labelled nodes, 2) imposes restrictions to relations stating that *in* doesn't accept any inputs as well as *eff*, and that *out* doesn't emits any output flows, 3) states that operations must be well connected by operand flows, accepting isolation of the *eff* labelled node. Thus $(\nexists\, e \in E |\, l_V\big(s(e)\big) = eff) \Longrightarrow T$ may happen, and is allowed, which assumes that designer doesn't know in advance all of the necessary effects required to sustain transformation within technical process. In fact this is one of the overall search goals. Interpretation is as follows: if an effect already exists than it must be obeyed, which is checked at the production rule definition, otherwise *eff* can be left isolated.

Incidence matrix r_{mn} (Table 6.1.) of labelled multi-digraph G is constructed object-based using a collection type as a prime building block to allow easy insertion of new rows and columns and any other matrix transformation required by phenotype construction within GE. Collection is a type of list accepting and enumerating any object. Thus, to account for dynamic requirements r_{mn} is composed as a collection of matrix rows, where each of rows is a collection of dependencies, with every dependency accepting a collection of arcs and every relation accepting a collection of relations. Such layered structure can tackle any imaginable transformation inside TP providing construction of appropriate methods to achieve them.

Table 6.1 Matrix representation of operand transformation process

Nodes	in	Op_1	…	Op_j	out	eff
in	r_{11}	…	…	…	…	r_{1n}
Op_1	…					…
…	…					…
Op_j	…					…
out	…					…
eff	r_{m1}	…	…	…	…	r_{mn}

Taking into account that *dependency is a collection of arcs* that accept effects or operands, then e_{mn} may be written as $e_{mn} \subseteq \{(u_m, v_n) | u_m, v_n \in V \wedge u_m \neq v_n\}$, with $[m] = \{1, \dots, j+3\}$ and $[n] = \{1, \dots, j+3\}$ where j denotes the number of operations. The complete structure of the transformation process is obtained by assigning a set of TP operations as source or target to relations inside incidence matrix r_{mn} as shown in Table 6.1. Put succinctly, by mapping operations to nodes and then relating them by arcs which are stacked inside the e_{mn}, dependencies are denoted.

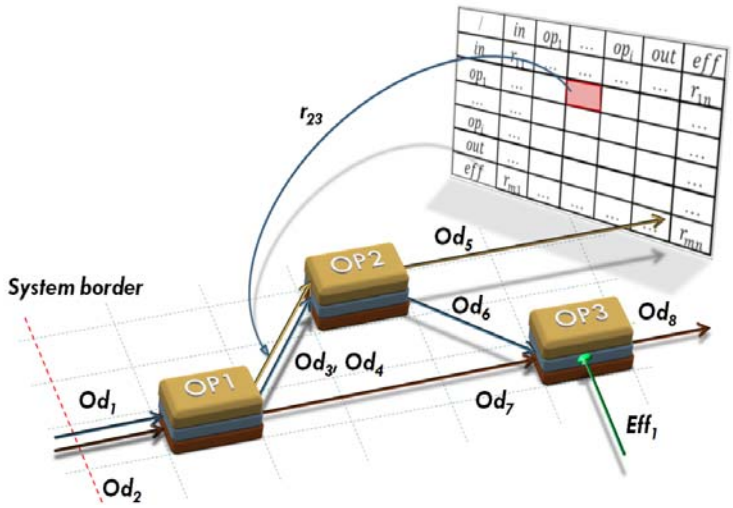


Figure 6.3 Multi-digraph with operations, operands and effects and its incidence matrix

Of course as a consequence of Definition 6.2 it follows that $l_V(s(e_{1n})) = in$, $l_V(t(e_{(p-1)n})) = out$ and $l_V(s(e_{pn})) = eff$ with fixed index $p = m$. Loops are not permitted in any of e_{mn} of incidence matrix fields. Fields of $e_{k(q-2)}$, $e_{k(q-1)}$, e_{kq} with $[k] = \{1, \dots, m\}$ and $q = n$ have no meaning in respect to modelling of technical processes and as such are not being used. Incidence matrix e_{mn} with technical process partitioned in layers depicting different operand flows that may occur between the same two operations is shown in Figure 6.3.

6.3 Graph grammar of technical processes

Rule based transformation of graphs can be understood as performing a local change to graph's structure under the instructions given by the production rule p. These instructions should address the following points [94], [95], [98] and [99]:

- exactly which part of graph's structure will be replaced – definition of matching procedure since L from $p: L \rightarrow R$ must be somehow identified in G,
- the sub-graph that will be inserted at desired place inside the graph G – definition of right side of the rule R,
- and finally, what is the mechanism for inserting R – specification of how to embed R into the structure of G.

Unlike string grammars (Definition 4.3) where rewriting is a straightforward procedure of sub-string or word replacement with the new sequence of symbols inside a sentence as defined by given grammar, graph grammars involve more complex procedures. Graphs are not just plain linear sequences of symbols; instead they have a structure defined through a set of nodes mutually related by a set of edges. In case of G, structure is even more complex, G is non-simple graph, with *TP entities* belonging to Σ_G being mapped to nodes and directed edges. Thus, when replacing a node of graph or sub-graph of graph, it is necessary to consider the surroundings of the structure that is to be replaced. Most often, these surroundings are referred to as a context of the replaced structure. Embedding mechanism prescribes a procedure by which the new inserted structure will be interconnected with the rest of host graph. Embedding assumes respecting the identity of graph's elements including both $l_E: E \rightarrow \Sigma_{Od} \cup \Sigma_{Eff}$ and $l_V: V \rightarrow \Sigma_{Op}$ through which operands, effects and nodes have been assigned to arcs and nodes. How to redirect edges to avoid occurrences of them having source or target pointing to *nil*, or how to reconnect edges properly to serve the purpose of graph transformation system, all of these implications must be accounted for by embedding mechanism or connecting procedure.

Thus, in order to be able to define a graph grammar that consists of production rules $p: L \rightarrow R$, first a mechanism for identification of L inside the host graph must be defined. Then the embedding procedure has to be specified. A definition of sub-graph is a prerequisite and it is given hereby [94], [95]:

DEFINITION 6.4 Let $\mathcal{G}_\Sigma$ be a finite set of all possible graphs that can be constructed over the alphabet of technical processes Σ_G, then a graph $C \in \mathcal{G}_\Sigma$ is called a sub-graph of $H \in \mathcal{G}_\Sigma$, if and only if the following conditions are to be satisfied:

$V_C \subseteq V_H$, $E_C \subseteq E_H$, $s_C(e) = s_H(e)$, $t_C(e) = t_H(e)$, $l_{E_C}(e) = l_{E_H}(e), l_{V_C}(u) = l_{V_H}(u)\ \forall\ e \in E_C \wedge u \in V_C$.

Definition 6.4 simply states that if a graph is also a sub-graph of another graph that the former must match both by its structure, nodes and arcs, and by its labelling to the graph of which it is a sub-graph. When considering the Definition 6.4 in a view of G, then a sub-process of technical process is a composition of the finite number of operations interrelated with operands flows and supported by necessary effects. To be able to apply graph production $p: L \to R$ rule a match L in host graph must be identified [94], [95]:

DEFINITION 6.5 For graphs $C, H \in \mathcal{G}_\Sigma$ a TP graph morphism $m: C \to H$ is a pair of structure preserving mappings $m_V: V_C \to V_H$ and $m_E: E_C \to E_H$ such that the following holds:

1) $\forall\ e \in E_C - \left(e | l_{E_C}(s_C(e)) \in \{in, eff\} \wedge\ l_{E_C}(t_C(e)) = out\right) \wedge$ $u \in V_C - u | l_{V_C}(u) \in \{in, out, eff\}$:
   - $m_V(s_C(e)) = s_H(m_E(e))$,
   - $m_V(t_C(e)) = t_H(m_E(e))$,
   - $l_{E_H}(m_E(e)) = l_{E_C}(\mathrm{e})$,
   - $l_{V_H}(m_V(u)) = l_{V_H}(u)$,
2) $\forall$ e $\in E_C | l_{E_C}(s_C(e)) = in$:
   - $m_V(t_C(e)) = t_H(m_E(e))$,
   - $l_{E_H}(m_E(e)) = l_{E_C}(\mathrm{e})$,
3) $\forall$ e $\in E_C | l_{E_C}(t_C(e)) = out$:
   - $m_V(s_C(e)) = s_H(m_E(e))$,
   - $l_{E_H}(m_E(e)) = l_{E_C}(\mathrm{e})$,
4) *iff* $\exists$ e $\in E_C | l_{E_C}(s_C(e)) = eff$:
   - $m_V(s_C(e)) = s_H(m_E(e))$,
   - $l_{E_H}(m_E(e)) = l_{E_C}(\mathrm{e})$.

First of all, the Definition 6.5 of morphism is a general one in respect to technical processes allowing that graph C consists of more than one operation Op, thus 1) defining morphism of

interrelated operations with disregard to *in*, *out* and *eff* labelled nodes treating them as a special cases. Relations that have a source in *in* are explained with 2) by retaining only their target and label. How to connect the source is dependent on the set of connecting rules, hence if L has e $\in E_C | l_{E_C}(s_C(e)) = in$ then the same edge can change the source when being interconnected with the source graph's structure. The same principle but inversed as given by 3) applies for edges of type e $\in E_C | l_{E_C}(t_C(e)) = out$, where the source and label are being retained and where the target changes depending on the connection rules. Finally, for the effects 4) if they exist inside the rule than he source and label are conserved, while the target depends only on the contents of the right hand side of production rule. Again, for the effects the same assumptions are used as in Definition 6.3.

Match of L in G can be defined using morphism as given in Definition 6.5 [94], [95]:

DEFINITION 6.6 A match of L in host graph G is found by existence of morphism $m: L \rightarrow G$, with $m(L) \subseteq G$, thus satisfying contact conditions.

Match of L in host G supports arbitrary number of operations, however it is important to stress out that match applied within this thesis will always correspond to only one operation node that will be indetified in G as given in definitions 6.4 and 6.5. Hence, since CFG is applied at GE level then a mapping will be established from single Op to single BNF token. The right hand side of the rule R, can have more than one Op. Graphical interpretation of morphism m from Definition 6.5 and match $m(L)$ from Definition 6.6 that is applied to the individual operation Op_2 and its neighbourhood in the host graph is given by the following two pictures (Figure 6.4 and Figure 6.5):

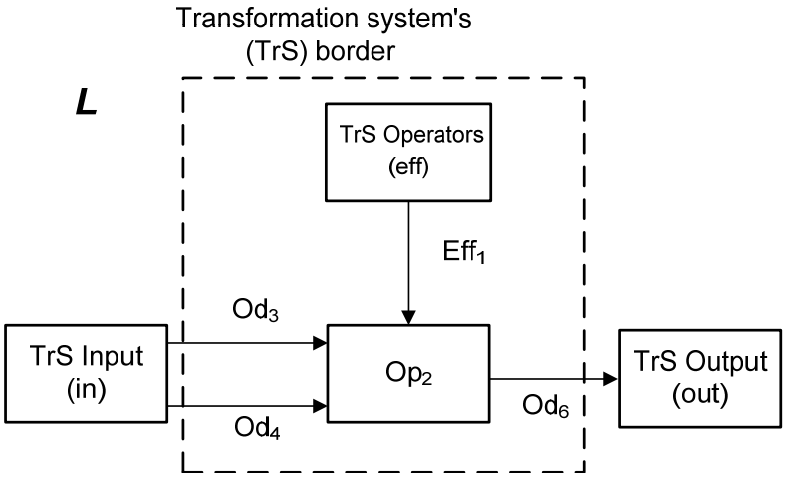


Figure 6.4 Example of rule *p* left hand side *L*

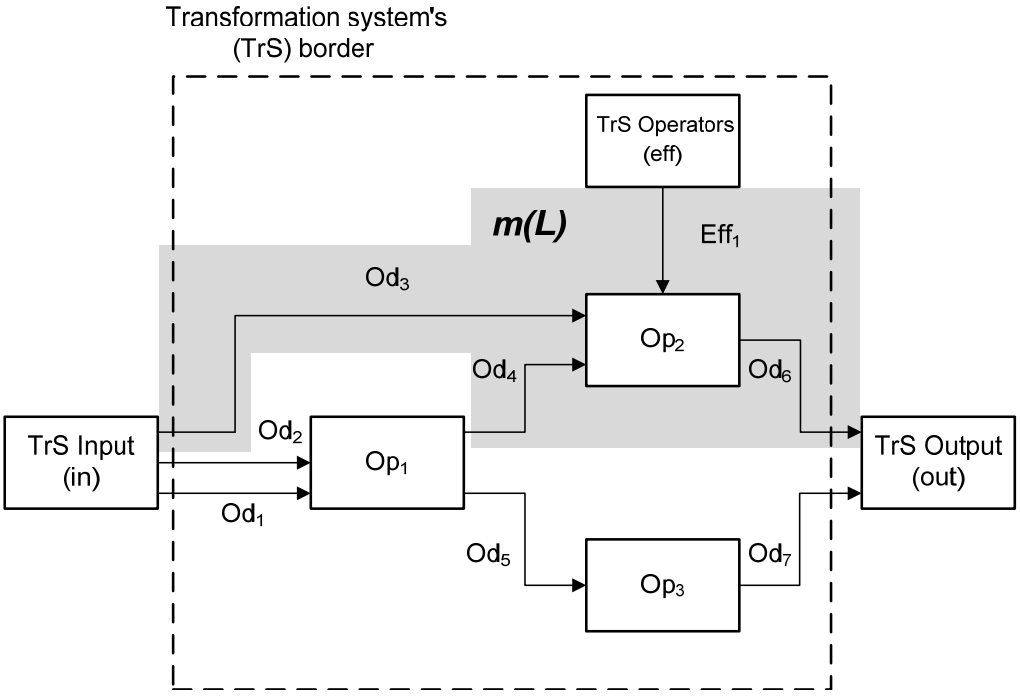


Figure 6.5 Identification of match of L as given in the Figure 6.4 using morphism $m(L)$ from Definition 6.5

Figure 6.5 depicts, marked in shaded area, an identification of match of L within a host graph. The extents of the morphism as given by Definitions 6.2 and 6.3 and example of L in the Figure 6.2 are clearly shown by excluding sources and targets of operand flows crossing the system's border, as well as the source of effects within transformation system. Hence, graphs of type L from $p: L \to R$ that are to be replaced by R are identified by $m(L)$ thus not paying attention to the connecting structure of host graph G. How to embed R within G will be regulated by connecting rule set ρ.

Applying $p: L \to R$ to graph G first identifies, then removes $m(L)$ from G and afterwards inserts R in its place, thus completing one step in the sequential derivation process. Insertion of R into G assumes the application of embedding mechanisms regulated by connecting rule set e_r. The derivation step is driven by the set of production rules p, morphism m and a set of connecting rules ρ is defined as follows [94], [98]:

DEFINITION 6.7 Using alphabet $\Delta \subseteq \Sigma_G$ since graph is labelled over Σ_G and taking finite non-empty set of production rules $p: L \to R$, then for every existing match $m: L \to G$ a direct derivation can be found stated as $G \overset{p,m,\rho}{\Longrightarrow} H$.

Following the Definition 6.7, generative sequential graph grammar of technical processes is defined as:

DEFINITION 6.8 A graph grammar of technical processes $\mathcal{GG}$ is defined as ordered triplet $\mathcal{GG} = (\mathcal{S}, \mathcal{P}_G, \Delta)$, with $\mathcal{S} \in \mathcal{G}_\Sigma$ as starting symbol, p as finite non-empty set of productions $p \in \mathcal{P}_G$ of type $p: L \to R$ and alphabet $\Delta \subseteq \Sigma_G$ over which graph is labelled.

Since grammar for describing technical processes is user-defined, it is logical to assume that it will be difficult to tell which elements will be variables and which will be terminals in advance. However, what can be stated is that terminal graph structure will contain terminal *TP entities* $\Delta \subset \Sigma_G$ since graph is labelled over Σ_G. Hence, terminal entities are operations from Σ_{Op}. All graphs composed of terminals is defined as as $\mathcal{G}_\Delta \subset \mathcal{G}_\Sigma$. A definition of language of technical processes $\mathcal{L}_{TP}$ generated by $\mathcal{GG}$ is stated directly from Definition 6.7:

DEFINITION 6.9 A language of technical processes $\mathcal{L}_{TP}$ generated by graph grammar $\mathcal{GG}$ is a set of graphs $G \in \mathcal{G}_\Delta$, which can be derived according to $\mathcal{GG} = (\mathcal{S}, p, \Delta)$ as:

$$\mathcal{L}_{TP}(\mathcal{GG}) = \left\{G | G \in G_\Delta, \mathcal{S} \Rightarrow^*_{\mathcal{GG}} G\right\}$$

6.4 Map between Chomsky's grammars and graph grammars

Generative grammars according to Chomsky and his proposed hierarchy are defined for linear strings of symbols. Linear strings in genotype and phenotype representation are also used within grammatical evolution to produce sentences in some formal language whose syntax has been defined within BNF. Since derivation of a sentence is nothing more than tree structured process, an attempt to generalise string structures to represent decomposition and synthesis of technical processes becomes reasonable and well justified. According to the literature [94], Chomsky grammars can be translated into string graph grammars. Thus if a string is composed as a sequence of symbols $\alpha_1\alpha_2\alpha_3 \dots \alpha_n$, $n \in \mathbb{Z}$, $|\alpha| = 1$ with symbols being elements of some given alphabet $\alpha_n \in \Sigma$, then it is possible to construct a string graph consisting of $n + 1$ nodes and n arcs or edges. Of course each of the edges that connect two consecutive nodes is labelled as prescribed by the $\alpha_n \in \Sigma$ [94].

Method presented within this thesis is node rewriting with underlying grammatical evolution working with BNF grammar, thus creating linear sentences. What is tried to be argued here is

that each derivation step under BNF generative grammar has to correspond to one decomposition step of technical processes in graph-grammar. In fact this is a necessity in order to create technical process decomposition step which is composed of a number of successive rewritings. As an example, Figure 6.6 shows BNF derivation process and its map to graph grammars:

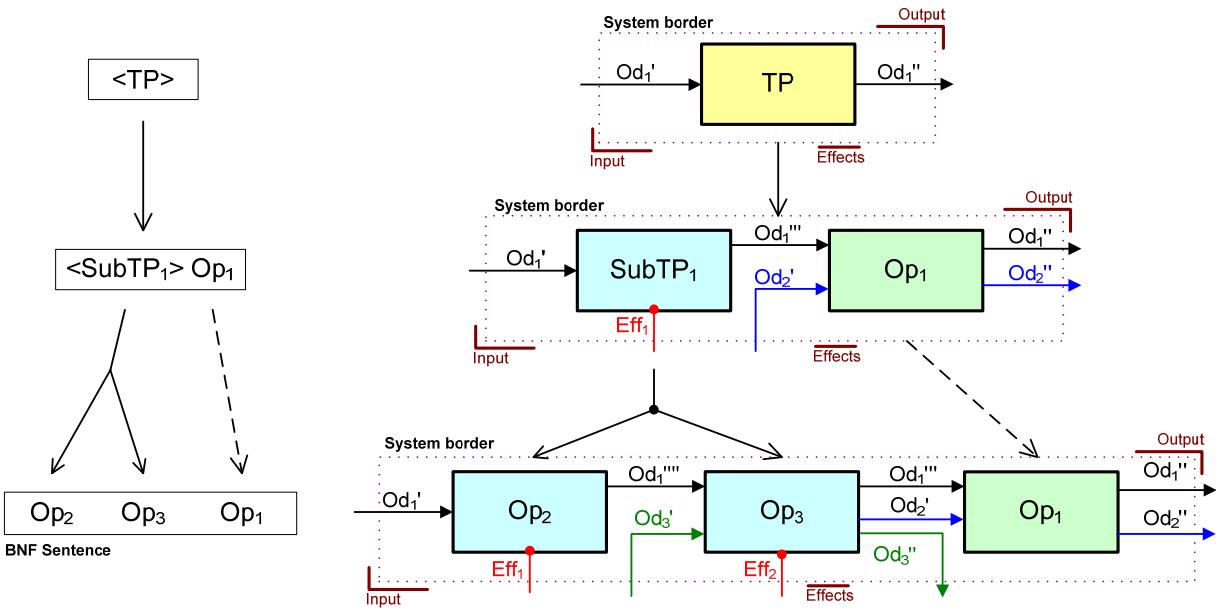


Figure 6.6 Example of the BNF derivation process and its map to graph grammars

Definition of graph grammar productions of type $p: L \rightarrow R$ requires first a definition of rule building blocks by the user; $\Sigma_{Op} \cup \Sigma_{Od} \cup \Sigma_{Eff}$ that is. Than according to Definition 6.3 each side of the rule is defined by designer as multidigraph labelled over Σ_G. In that way the relation $D_{\mathcal{L}_{TP}(\mathcal{GG})} : \mathcal{L}_{TP}(CFG_{TP}) \rightarrow \mathcal{L}_{TP}(\mathcal{GG})$ as given in (5.23) is only partially established since it is bounded only to the knowledge describing how individual operation can be decomposed without addressing of how each of these decompositions can be integrated with one and another. Context-free grammar of technical processes CFG_{TP} and its language of $\mathcal{L}_{TP} = \mathcal{L}_{TP}(CFG_{TP})$ are defined as follows:

DEFINITION 6.10 A context-free grammar of technical processes CFG_{TP} expressed in BNF is as a quadruple $(\Sigma_s, \mathcal{V}_s, \mathcal{S}_s, \mathcal{P}_s)$ where: $\Sigma_s \subset \Sigma_{Op}$ is a finite non-empty set of terminals belonging to operations, $\mathcal{V}_s \subset \Sigma_{Op}$ is a finite nonempty set of non-terminal symbols or variables satisfying $\Sigma_s \cap \mathcal{V}_s = \emptyset$, $\mathcal{S}_S$ is a starting symbol or axiom with $\mathcal{S}_s \in \mathcal{V}_s$, and $\mathcal{P}_s$ is a finite nonempty set of production rules of the type $\alpha \rightarrow \beta$ where: $\alpha \in \mathcal{V}_s$ and $\beta \in (\Sigma_s \cup \mathcal{V}_s)^*$.

DEFINITION 6.11 A formal language $\mathcal{L}_{TP} = \mathcal{L}_{TP}(CFG_{TP})$ generated by grammar $CFG_{TP} = (\Sigma_s, \mathcal{V}_s, \mathcal{S}_s, \mathcal{P}_s)$ is defined as:

$$\mathcal{L}_{TP}(CFG_{TP}) = \{\omega | \omega \in {\Sigma_s}^*, \mathcal{S}_s \Rightarrow^*_{CFG_{TP}} \omega\}$$

Hence, two steps are required to fully establish relation (5.23), one performed by designer when creating productions, and one automated by the computational systems when these productions are applied and need to be interconnected. Linear sentences generated with CFG_{TP} provide layout which should be followed by the computational system when patching up technical process provided by the right hand side of productions. On the left hand side of Figure 6.6 a derivation tree in CFG_{TP} is presented. Rewritings of $SubTP_1$ into Op_2 and Op_3, and TP into Op_1 are predefined by designer, however connecting together Op_2 and Op_3 with Op_1 is determined by predefined connecting procedure ρ embedded within computational system. Although a single node is always being rewritten it is still necessary to establish morphism $m(L)$ to determine node's neighbourhood within the host graph G. Hence, the information about edges and how to reconnect them after the node has been rewritten can only be provided by the graph grammar $\mathcal{GG} = (\mathcal{S}, \mathcal{P}_G, \Delta)$ and its connection procedure ρ. As an example, on the right hand side of Figure 6.6 it can be clearly seen how operand $Od_2{}'$ enters the transformation from outside the system in second derivation step. Finally, at the last derivation step it was determined by the connection mechanism that $Od_2{}'$ flow could be provided by Op_3 instead, thus eliminating one unnecessary flow. Finally, it is important to stress out that graph-grammar inherits the orderings of operations Op as presented within CFG_{TP} generated string. Since operations within processes can be performed in parallel taking into account the influence of knowledge formalisation map (5.23) and predefined connection procedure, then resulting structure obtained from linear string $\alpha_1\alpha_2\alpha_i \ldots \alpha_n$ can take any structure of the multigraph from $\mathcal{G}_\Sigma$, like in examples shown in Figure 6.7 (for sake of simplicity operand flows are represented as single arcs, effects, *in*, *out* and *eff* are omitted):

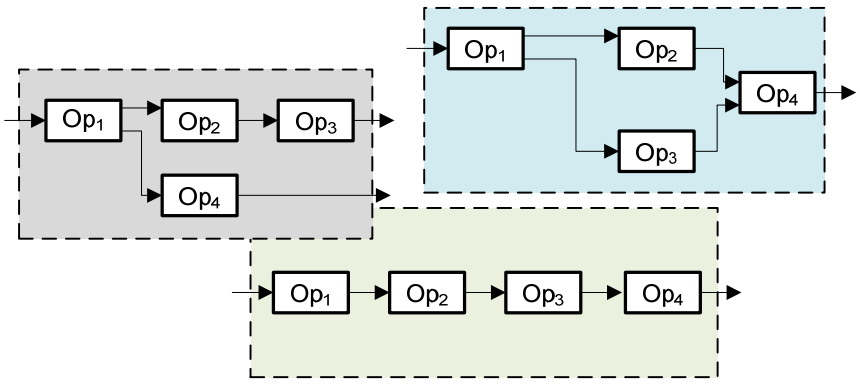


Figure 6.7 Example of resulting TP structures after map from BNF to graph grammar language

Providing the existence of (5.23) the orderings of the nodes inside incidence matrix r_{mn} also inherit orderings from linear BNF strings. Each derivation step is complete after all n rewritings of original $\alpha_1\alpha_2\alpha_i \dots \alpha_n$, $[i] \in \{1, \dots, n\}$ strings in BNF have been accomplished breadth-first. Identification of the insertion place k_i in r_{mn} thus $r_{m=k,n=k}$ for *i-th* rewriting by production $\alpha \to \beta$ is given as follows (size of individual token $|\alpha_i| = 1$):

$$k_i = 1 + \sum_{j=1}^{n}(|\alpha_j|\,|\alpha_j \in \Delta) + \sum_{j=1}^{i}(|\beta_j|\,|\,\beta_j \in \Sigma_{Op}) \tag{6.1}$$

The orderings of arcs e_{mn} stored in dependencies, a cell of r_{mn} that is (see Table 6.1), are also respected when applying embedding procedures. Since (5.23) exists than the place of rewriting i in $\alpha_1\alpha_2\alpha_i \dots \alpha_n$

With addition of connecting procedure ρ the mapping as given in (5.23) is complete thus defining graph-grammar language of technical process $\mathcal{L}_{TP}(\mathcal{GG})$ by that rendering synthesis of technical processes to be run computationally. In the upcoming section multidigraph's transformation algorithm as well as connecting procedure ρ will be defined formally.

6.5 Transformation algorithm and connection procedure ρ

Two types of embedding principles can be considered: *connecting* and *gluing* [98]. In case of connecting production rules of type $p: L \to R$ contain embedding rules ρ which are to be applied to integrate R into structure that emerged after $m(L)$ was subtracted from G, or put succinctly after $G - m(L)$. The principle difference between two approaches is that connecting performs creation of edges that have to abe dded to connect R with $G - m(L)$, whereas in gluing searches to identify which elements are present both in L and R to reuse them as much as possible when integrating R into $G - m(L)$. In fact only the elements that are unique in R are added anew, while elements both present in L and R are preserved. This thesis will use connecting approach thus defining connection procedure ρ.

Algorithm for transformation of graph $G = (V, E, s, t, l_E, l_V)$ under the set of production rules of type $p: L \to R$ with $|V_L| = 4$ according to Definitions 6.3 and 6.4 specifying what technical process graph is and with embedding mechanism defined by connection procedure ρ, is given as follows:

1) First a match $m: L \rightarrow G$ of the rule p left hand side L must be established in the host graph G according to Definition 6.6. The rewriting is performed always by replacing only a single Op labelled node of the host graph G with structure in R consisting of an arbitrary but greater than two number of operations.

2) After the match $m(L)$ has been established and the place of insertion specified, the Op labelled node u_{Op} such that $L \ni u_{Op} \in G$, $u_{Op} | l_V(u_{Op}) \in \Sigma_{Op}$ present both in the left hand side L of production rule p and in the host graph G has to be subtracted from the host graph G thus creating an intermediate structure as $G^- = G - u_{Op}$. The G^- is left without u_{Op} node thus resulting in number of dangling edges, either one of these edges is deprived of only one source or target but not of both at the same time. For sake of being pragmatic it will be assumed that these sources and targets which are left empty simply point to *nil* thus yielding with the following set of interfaces $E_{G_i^-} = ((u, nil) \wedge (nil, u) | u \in V_{G^-})$.

3) Graph on the right hand side R of the production rule p first has to be deprived of nodes labelled as *in*, *out* and *eff*. The subtraction $R^- = R - V_{TrS}$, with set of nodes defined as $V_{TrS} = (u_R \in R | l_{V_R}(u_R) \in \{in, out, eff\})$, creates a set of edges deprived of only one source or target but not of both at the same time. Again, interface edges of R^- are defined as $E_{R_i^-} = ((u, nil) \wedge (nil, u) | u \in V_{R^-})$.

4) To complete the transformation, R^-must be added to the G^- and reconnected to the remaining structure according to connection procedure ρ, thus $G_T \overset{\rho}{\leftarrow} G^- + R^-$. Matching of R^- edges with the edges that have emerged as a result of creation of G^- and with respect to order of edges within dependencies of the host graph's G^- incidence matrix $r_{mn_{G^-}}$ proper interfaces will be created. Interface matching function $Inter(e_{G_i^-}, e_{R_i^-})$ is defined as follows:

$$\begin{aligned} &Inter\left(e_{G_i^-}, e_{R_i^-}\right) = true \ iff: \\ &\quad l_E\left(e_{G_i^-}\right) = l_E\left(e_{R_i^-}\right) \wedge s\left(e_{G_i^-}\right) = u \in V_{G^-} \wedge s\left(e_{R_i^-}\right) = nil, \\ &\quad \vee \\ &\quad l_E\left(e_{G_i^-}\right) = l_E\left(e_{R_i^-}\right) \wedge t\left(e_{G_i^-}\right) = u \in V_{G^-} \wedge t\left(e_{R_i^-}\right) = nil. \end{aligned} \tag{6.2}$$

If $Inter\left(e_{G_i^-}, e_{R_i^-}\right)$ yields truth, then $e_{G_i^-}$ will take for source/target the node v from R^- for which it holds the following $v = (v \in R^- | \, s\left(e_{R_i^-}\right) = v \vee t\left(e_{R_i^-}\right) = v)$. Edge reconnection procedure $Reconn\left(e_{G_i^-}, e_{R_i^-}\right)$ is defined as follows:

$$Reconn\left(e_{G_i^-}, e_{R_i^-}\right) = \begin{cases} iff\ s\left(e_{R_i^-}\right) = nil, & t\left(e_{G_i^-}\right) \leftarrow t\left(e_{R_i^-}\right) \\ iff\ t\left(e_{R_i^-}\right) = nil, & s\left(e_{G_i^-}\right) \leftarrow s\left(e_{R_i^-}\right) \end{cases} \tag{6.3}$$

Transformation algorithm required for $G_T \overset{\rho}{\leftarrow} G^- + R^-$ for *k-th* rewriting of derivation step providing G, $p_s: \alpha \to \beta$, $p: L \to R$ is given with the following pseudo-code (connection procedure ρ is defined in lines 10-14):

Input: $G, p_s: \alpha \to \beta, p: L \to R$

Output: G_T

1 For a_j calculate insertion in $r_{mn\mathrm{G}}$ providing $p_s: \alpha \to \beta$

2 $u_{Op} \leftarrow m(L)$;

3 $G^- \leftarrow G - u_{Op}$;

4 $E_{G_i^-} \overset{i}{\leftarrow} G^-$;

5 $R^- \leftarrow R - V_{TrS}$; (6.4)

6 $E_{R_i^-} \overset{i}{\leftarrow} R^-$;

7 $G_T \leftarrow G^- + R^-$;

10 **while** $(|E_{R_i^-}| > 0)$ **do**

11 $c \leftarrow 1$; $e_{G_i^-} \leftarrow E_{G_i^-}[\mathrm{c}]$; $e_{R_i^-} \leftarrow E_{R_i^-}[1]$;

14 **while** $(Inter\left(e_{G_i^-}, e_{R_i^-}\right) \neq True)$ **do** c++; **od**

13 If $Inter\left(e_{G_i^-}, e_{R_i^-}\right) \to (Reconn\left(e_{G_i^-}, e_{R_i^-}\right)$, Delete $E_{R_i^-}[1])$;

14 **od**

Example of the transformation process performed according to (6.4) is depicted in the following figure:

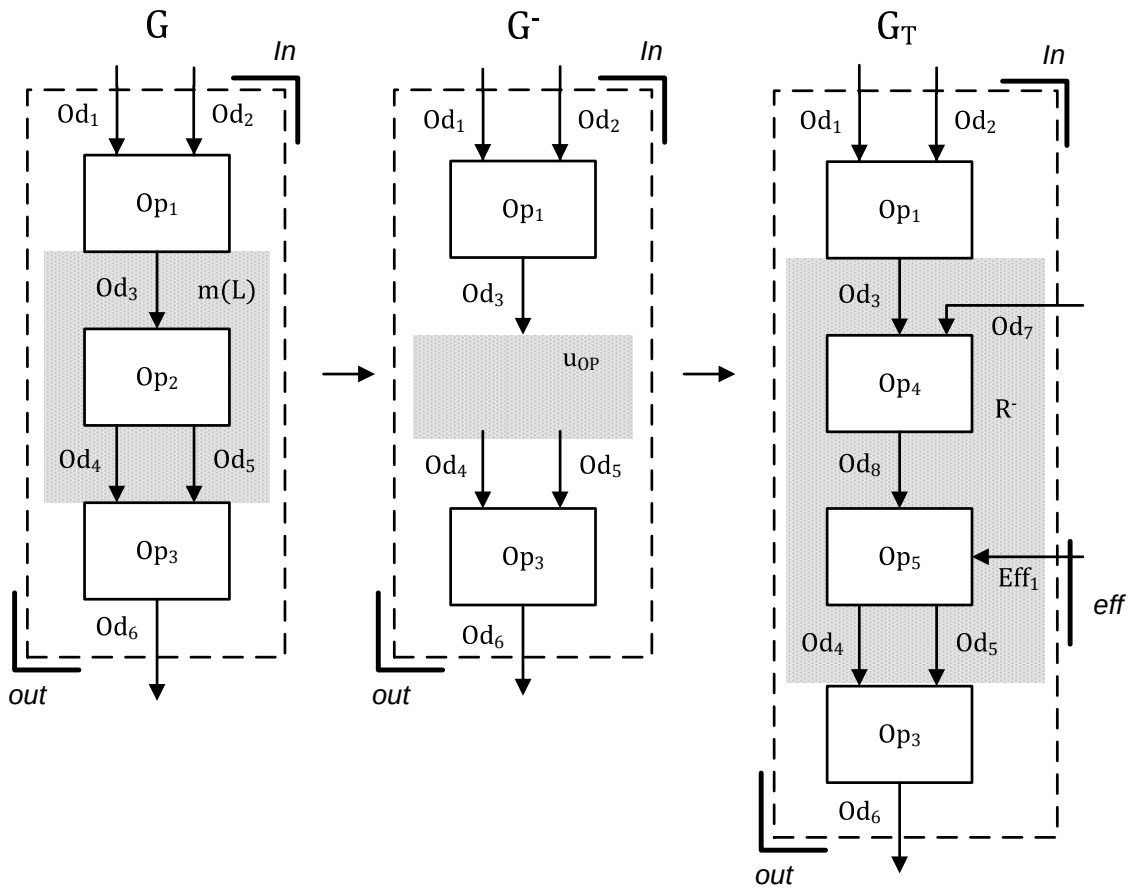


Figure 6.8 Transformation steps $G \rightarrow G^{-} \rightarrow G_T$

6.6 Implications to the knowledge formalisation

Formal system as presented within this Chapter has it rules thus affecting the knowledge that has to be formalised. Violating of these rules, is not allowed by the system, otherwise the system would not be able to operate. The summary of the most important implications is given here bellow:

1) Technical processes are defined as a labelled multigraph with operands, operations and effects $G = (V, E, s, t, l_E, l_V)$ according to Definitions 6.1 and 6.3, with $|V| \geq 4$.
2) For operation $Op \in \Sigma_{Op}$ it is said that it transforms operand $Od \in \Sigma_{Od}$ only if there exist input and output flows from such operation.
3) Derived from 2), for nodes labelled as *in*, *out*, and *eff* it is said that they only participate in transformation, since nodes *in*, and *out* provide only the source and target crossing the transformation systems border, and *eff* provides necessary effects for support of transformation.

4) Principle operands Od and the transformation of these are represented by initial graph $\mathcal{S}$ (principle transformation marked with asterisk in Figure 6.9)
5) Left hand side of the rule $p: L \rightarrow R$ with $|V_L| = 4$ is defined according to Definitions 6.3 and 6.4.
6) Right hand side of the rule $p: L \rightarrow R$ with $|V_R| \geq 4$ should contain at least two operations from which at least one operation should transform operands as given in 3); according to Hubka's law secondary flows and effects can appear as shown in Figure 6.9 (principle operands marked are marked with asterisk, *in*, *out* and *eff* nodes omitted for the sake of simplicity). Special cases where $|V_R| = 4$ are allowed only if change of operation label posses significant semantic meaning.

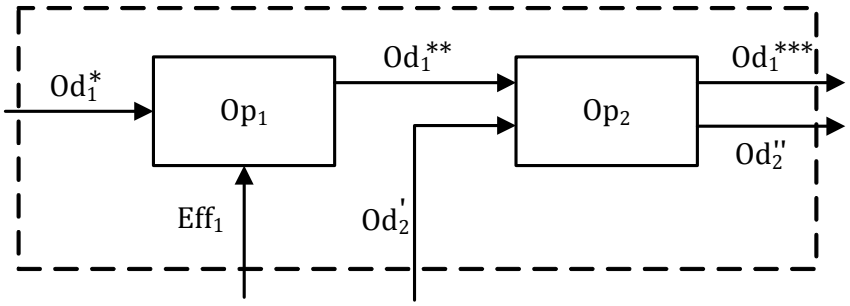


Figure 6.9 Emergence of secondary flows (principle transformation marked with asterisk) in right hand side of rule $p: L \rightarrow R$

7) Unless specified otherwise, the decomposition of technical process stops when all possible rewritings for the corresponding derivation tree have been exhausted.

The selection of formalism applied came to choosing a labelled multi-digraph. Multigraph is on the line with engineering institutive rule of a thumb reasoning resulting in the immediate mapping of operations to nodes and operands to arcs. However, a slightly more advanced concept of hypergraph came into consideration. In distinction to multigraphs, hypergraphs [97] are generalisation within graph theory, thus every graph is a hypergraph. Moreover, hypergraphs allow relations that are able to connect by definition, any number of graph's node. That set based approach for definition of relations, although at first glance somewhat awkward and distant to engineering applications and modelling, offers an easy way of implementing double and single pushout approaches to graph transformation, what is in contrast to hereby applied node rewriting principles. Utilisation of concept of hypergraphs will be left for the further research efforts.

7. KNOWLEDGE FORMALISATION AND EXAMPLE

In the beginning is the relation (Martin Buber, philosopher, 1878 – 1965, from I And Thou)

7.1 Formalization of the knowledge about technical process

Making knowledge both understandable and consistent is of paramount importance for design of knowledge-driven computational reasoning systems. The system's overall performance will be determined by the boundaries of knowledge formalisation that has been applied; either it will be hindered by it or the system would be robust and efficient. Thus, creation of the computational system that applies automated reasoning consists of two equally important parts; a strictly formal part dealing with the logic behind method's definition involving definition of means to make knowledge computationally understandable and of a subjective part when knowledge from the selected application domain was described and represented as proscribed by the method [101].

The method was elaborated in the previous Chapter presenting a formal part in the creation of a computational reasoning system - a graph-grammar rule based transformation system that is being applied to perform the decomposition of technical processes. The outcome of decomposition should be synthesised operand transformation variants. These have to clearly depict the necessary effects emerged under technological principles and resulted as a consequence of the process of product's usage, thus consequently imposing requirements to the function of technical system that is ought to be designed. Knowledge representation involves definition of objects and relations between these objects in order to provide semantics thus facilitating inference processes. Production systems manage to achieve semantics by using set of rules, thus bringing in relation different concepts. Inference is then conducted by rule application according to type of grammar applied. The subjective part refers to an actual event when designer must describe some of his or hers knowledge within the application domain using means provided by the computational tool. To clarify, the same concepts can be interpreted and related in many different ways and contexts depending on the viewpoint taken. Design know-how used to achieve required result within conceptual design phase depends on designers knowledge and experience in the field, thus varying from person to person both in viewpoints and in depth. Although such diversity in solution finding is precisely what is tried to be captured, computational reasoning systems must maintain a

degree of consistency of formalized knowledge in order to produce meaningful results. It can be assumed to an extent that among designers working and collaborating together there exist a shared understanding about concepts they deal with as a part of a daily routine. However if taking into account designers within domain that do not collaborate or even among different engineering domains, than the shared understanding would be reduced for sure. For large-scale knowledge systems the experience tells that before actual formalization, the content of knowledge requires careful systematisation to facilitate a maximum out of systems capabilities [100].

This Chapter will provide findings that are required to define foundations for knowledge formalisation about technical process. Although the generalisation of technical process entities is not yet supported within the boundaries of developed method for generation of operand transformation variants, it is still necessary to at least suggest guidelines for knowledge formalisation which should be followed when defining production rules. For that purpose online lexicon of the English language WordNet [104], The Suggested Upper Merged Ontology (SUMO) as the largest open ontology [107], and recommendations for reconciliation of product function related terms accepted by the NIST (National Institute for Standards and Technology of US) [22] where examined. The Chapter will conclude with two examples of synthesis of technical processes using method as developed within this thesis.

7.2 Taxonomy and ontology

The knowledge driven computational method for the generation of operand transformation presented in Chapter 6 considers creating and expanding production rules database by designer. Unlike productions that are operated only by the computer, suggestions how to formalise the knowledge are intended for designers in order to produce structured knowledge organized in such manner that is “best-fit” for the developed method. In fact these suggestions should at least include a taxonomy which can be utilized further also by the computational system to help maintain consistency and reduce the overall number of rules thus creating solid foundations for method’s application. Taxonomies denote subclass relations among the objects of the domain of interest, or more broadly entities that include concepts, attributes and relations and are the first step in rendering of the body of knowledge to computational environment [101], [102]. Understanding of taxonomic relations among concepts emerge as a result of the observations, although the surrounding world might appear random and unordered, there is a way how to assign concepts that share same properties to their respective

class. As a result the generalisation is created focusing just at the relevant entities attributes and relations. To put the latter the other way around, it is mandatory to add terms that could be sufficiently well told distinct, otherwise a bias that occur in respect to the meaning of individual terms may hinder the inference capabilities of the whole system.

A step further is the introduction of ontology which allows a multitude of various types of n-ary relations among the domain objects to facilitate a more extensive computational inference process. In a broad sense, definition of ontology states that it is the study of the categories of things that exist or may exist in some domain [33]. Put more specific, the ontology can be described as an explicit specification of a shared conceptualization, which can be taxonomically or axiomatically based [103]. In general, to define ontology three parts must be specified: concept definitions attribute definitions, and further inference definitions like backward chaining rules, path grammars, and so forth [102]. If taking the intended purpose viewpoint, then ontology can be recognized as two general types: the problem solving ontology and domain ontology [101]. The first involves the activity of identifying, formulating and obtaining a solution to the problem, and the latter corresponds to the domain as distinct from the problem or tasks in that domain [33]. However, recent and ongoing research efforts in the field of knowledge engineering including both general-purpose and low-level engineering specific applications do not present straight as it is the definition in respect to the basic technical process taxonomy [100], [104] and [107]. At the moment only the Design Ontology [17], [102], [104] provides product knowledge vocabulary as given within the Theory of Technical Systems. Vocabulary contents are classified into six main subcategories divided between physical and abstract world with the categorization of the relations based upon logical properties of symmetry, reflexivity, and transitivity. True, the Design Ontology offers the definition of high level concepts in respect to TTS and technical process, however for computational applications lower more concrete process related concepts should be also provided.

In fact, there is a strong division between ontology based problem solving approach and the Computational design synthesis directed mainly to the application of formal grammars. Reasons for being that so lay in the complexity and interdisciplinarity of both approaches requiring immense effort in order to create a unified and even more robust framework to support early product development. For example, production rules require generalisation and broader scope thus introducing complex mathematical structures involving both type graphs and typed graphs which are nothing more than taxonomies and specific mid-level ontology.

In order to model technical process formally it was necessary to introduce related technical process entities such as operands, effects and operations namely, into multi-digraph's structure. *TP entities* are considered as labels of nodes and arcs, with operations mapped to graph's nodes, and operands and effects mapped to arcs. All of these *TP entities* are members of vocabulary set Σ_G (Defintion 6.1). However, what have not been addressed are relations between these objects which can be established as consequence of technical processes understanding. This thesis will not embark the course to extend the rewriting system to include attributes, types and inheritance or to define operations as a part of algebraic system over operands. Instead it will only try do define the principles as recommendations that should be obeyed when defining production rules of technical processes. Full generalisation of production rules by creating *TP entity* taxonomies thus providing inheritance over instances of classes, both at computational and knowledge engineering levels will be left for the further research.

7.3 Foundations for the knowledge formalisation

On epistemological level of modelling for a particular domain of interest, the knowledge formalisation is performed first by identifying basic and generic terms and possible relations that could be raised between these terms [17]. Of course, one might than define a body of knowledge that has been formalized as a set of terms/objects that are connected with different type of relations thus altogether providing higher semantic meaning. The usual outcome of knowledge formalisation is the creation of an abstract high level model that captures basic and wide spread common sense knowledge. However, most often the engineering or scientific application opts for a more detail and specific definitions demanding the loss of bias that is present with the general knowledge. Thus, on the account of the underlying formalisation two cases can occur [42]: knowledge is formalised completely without bias and it is domain specific - a formal language that is, or bias in semantics can occur since the knowledge being formalised tends to be close to a natural language. Selection depends how close the computational system will have to come close to emulating human reasoning. The truth is, a strictly formal engineering formalisation can produce coherent results, but because of the one-to-one mappings the system will be deprived of possibilities to use more knowledge intensive reasoning techniques like establishing of analogies. In that way creating out-of-box reasoning which is sometimes cited as the source of creativity within cognitive linguistics remains unreachable [26]. This section will in its continuance provide examination of some existing

high and mid level taxonomy and ontology and work that has been done in order to formalise functions of technical system in order to suggest recommendations for the definition of production rules considering decomposition of technical processes. Lexicon of the English language *WordNet* [104] will be compared to the largest publicly available engineering ontology or SUMO [106] and [107], in order to propose additions to product function's taxonomy to be reused as guiding lines for production rule definition [22].

7.3.1 WordNet

The *WordNet* [104] is the Internet based lexical database of the English language developed at the Princeton University. It is a lexicon where nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms or synsets. In order to capture expressiveness of a natural language synsets are interrelated by means of conceptual-semantic and lexical relations. The result is creation of a conceptual network which can be navigated following conceptual-semantic and lexical relations by using the Internet browser. Although predominantly oriented to natural language processing *WordNet* offering more than one to one mappings between terms, it can still be utilized for the purposes of formalisation of engineering knowledge about technical processes. Figure 7.1 presents a portion of the *WordNet* lexicon that can be utilized when designer is about to define production rules for decomposition of technical processes. The structure shown is in almost taxonomic relationship; above of a chosen term is denoted as term's inherited hypernym. Consequently all the terms bellow are a troponym of the chosen term. Pure inheritance structure is not exact, thus semantic, since troponym only partially fulfil type-of relationship. In general, troponym of a verb bears a lot of semantics since it is applied to expresses a more specific meaning of a verb that it is to replace. In Figure 7.1 (if further decomposition exists a three-doted element is related to the term, thicker line denotes no more decomposition possible).

Two distinct trees which can be observed in Figure 7.1 which are of special interest to TP's; the one with (shaded) root specifying change as to undergo or experience a change and the other with its root denoting a change as a cause. The change as *undergo a change* is a viewpoint taken when considering operands with their respective change of state and change as *a cause to change* denotes viewpoint that has to be taken when considering operations, a process that is. Change of operand attributes thus include change of form or shape, change of state, change of internal physical properties as change of integrity, conversion, change of magnitude including addition and finally a division of objects in parts (Figure 7.2).

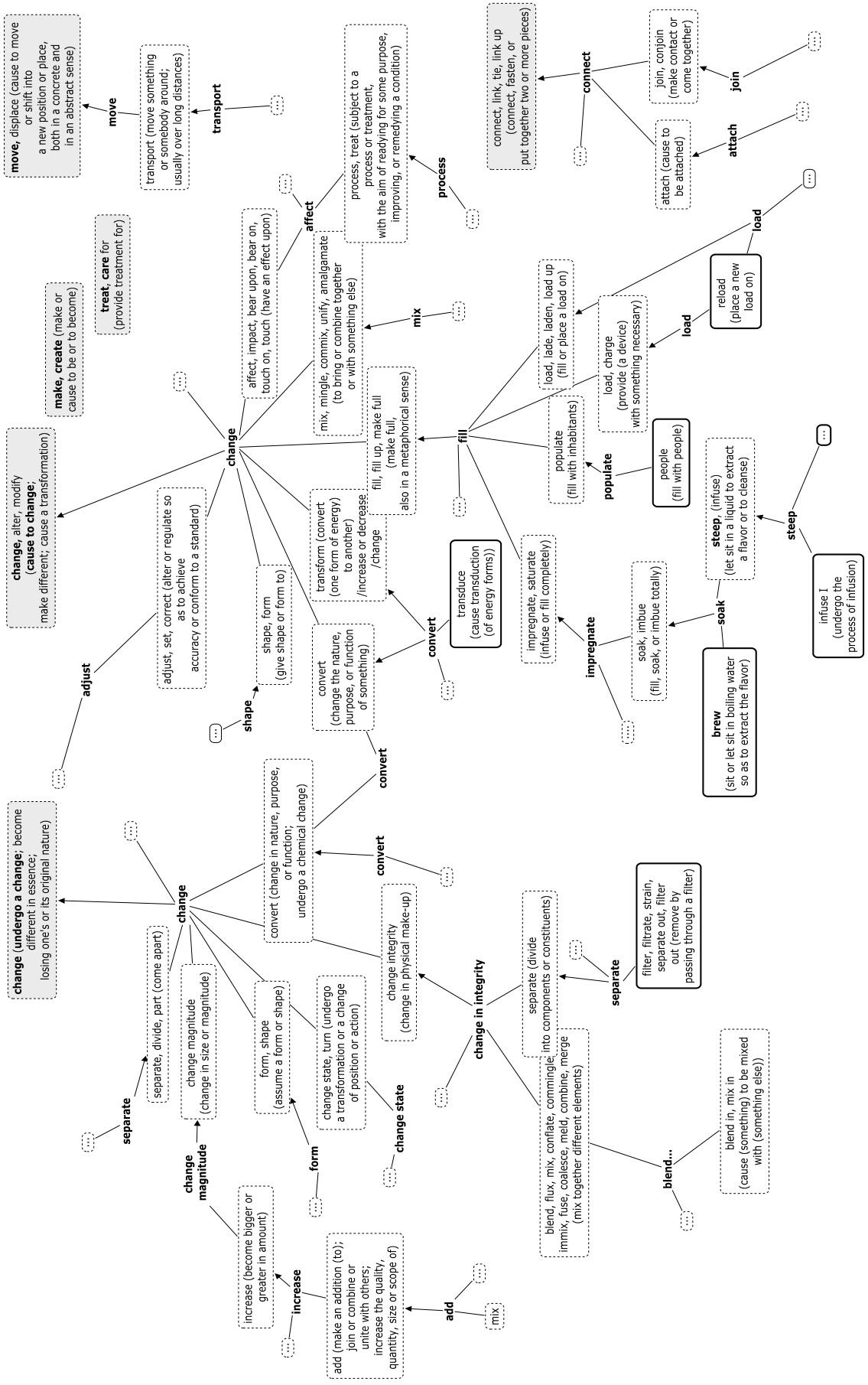


Figure 7.1 Semantic links between terms related to TP's according to *WordNet* [104]

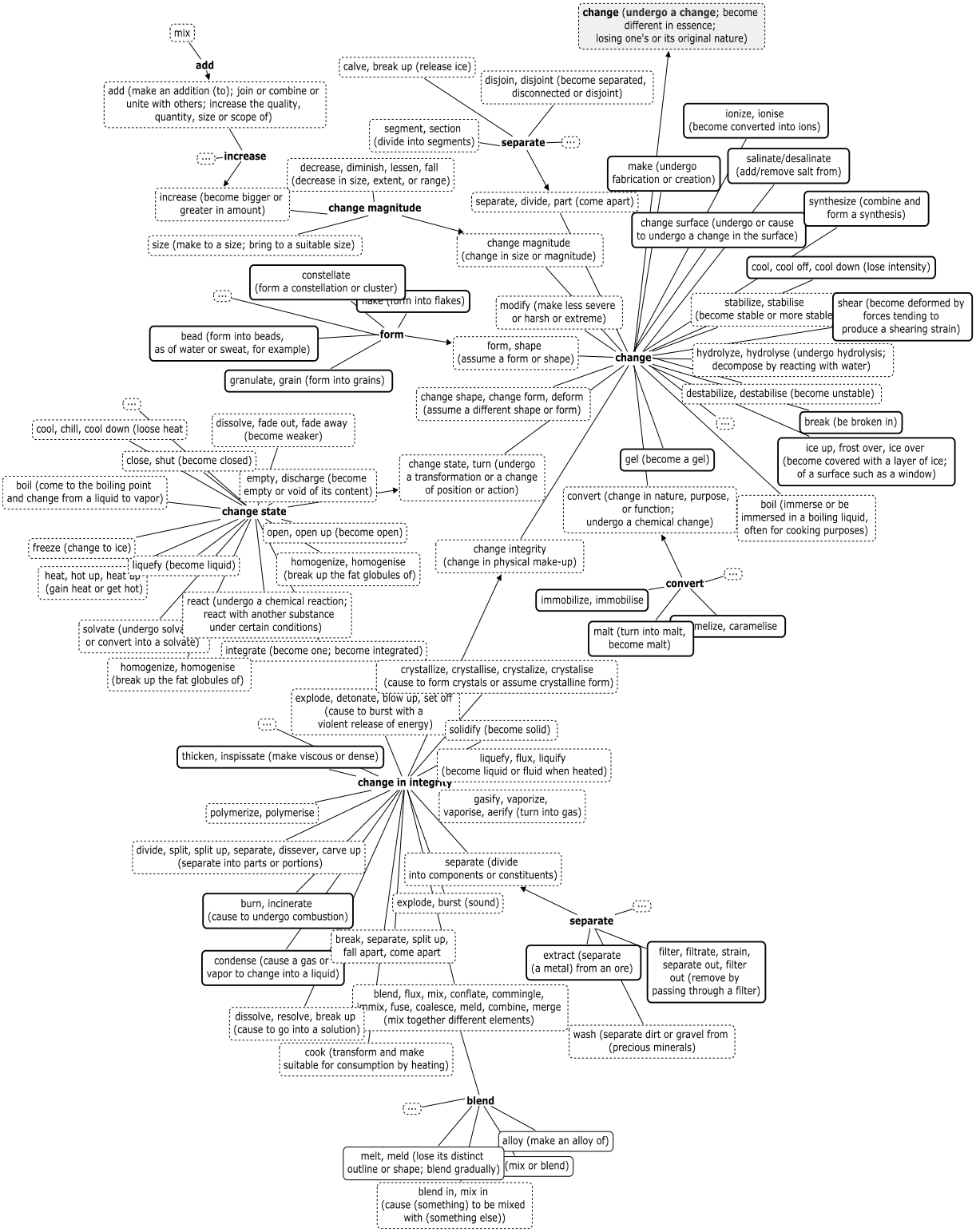


Figure 7.2 Detail semantic links in the context of operand transformation as in *WordNet* [104]

Each of these terms can be divided further (omitted in Figure 7.1 because of simplicity). Process viewpoint assumes a cause of change, involving regulation and adjustment, affecting with processing, mixing, converting and so on. It is necessary to emphasize that processing or cause of change as a root is understood in much broader sense making it less usable to

engineering applications. Roots involving moving of objects, connecting them or treating of living beings are presented separated of root that involves *causing of a change*. For a natural language processing this holds, however it is necessary to include at least some of these as processes which makes perfect sense in engineering application domains, thus opting for considering more engineering oriented taxonomies.

7.3.2 SUMO

Like the WordNet, The Suggested Upper Merged Ontology (SUMO) is open for public and being available for browsing over the Internet [107]. SUMO is the largest formal public ontology in existence today [107]. It is written in SUO-KIF language and it achieved a complete mapping over the *WordNet* lexicon of the English language, thus bringing together formal and natural content in an acceptable way. SUMO ontology is axiomatic in its core which allows an automated inference process, both offering both taxonomy and ontology over related terms. Since all of the terms are formally defined their meanings are independent of inference procedures applied and in that way create a robust inference system [107].

Most upper levels of the SUMO ontology are defined starting with the term *entity* thus denoting both physical and abstract that exists in our surroundings (Figure 7.3). Such approach is coherent with the application in the field of artificial intelligence, for example with the general ontology as proposed by Russell and Norvig [101]. At this point of research only the branch that is composed of physical entities is of interest. The latter encompass both objects and processes. According to SUMO, an object is defined as a tangible and visible entity. Some of its derived subclasses, i.e. agent, are closely related to the inference processes built in SUMO. Although these objects might prove handy for the generation of operand transformation operands they exceed the scope of the method presented within this thesis. Thus, the self-connected object with its instances is of interest, however as it will be shown later in this section, the taxonomy of operands will be adopted as given within engineering product function reconciliation, adopted as a part of NIST [22]. Moreover, SUMO specifies objects as of biological origins, as material, content bearing objects as information container and collections [17], [107]. Such specification is somewhat coherent with the one given by TTS where operands are categorised as being of biological organ, material, information and energy. Process in SUMO is defined as a sustained phenomenon or one marked by gradual changes through a series of states. Such definition of process is coherent with engineering understanding of process and technical processes as given by the TTS, thus more closely

related to the scope of this thesis than the definition provided by *WordNet* stating that process denotes subject to a process or treatment, with the aim of readying for some purpose, improving, or remedying a condition.

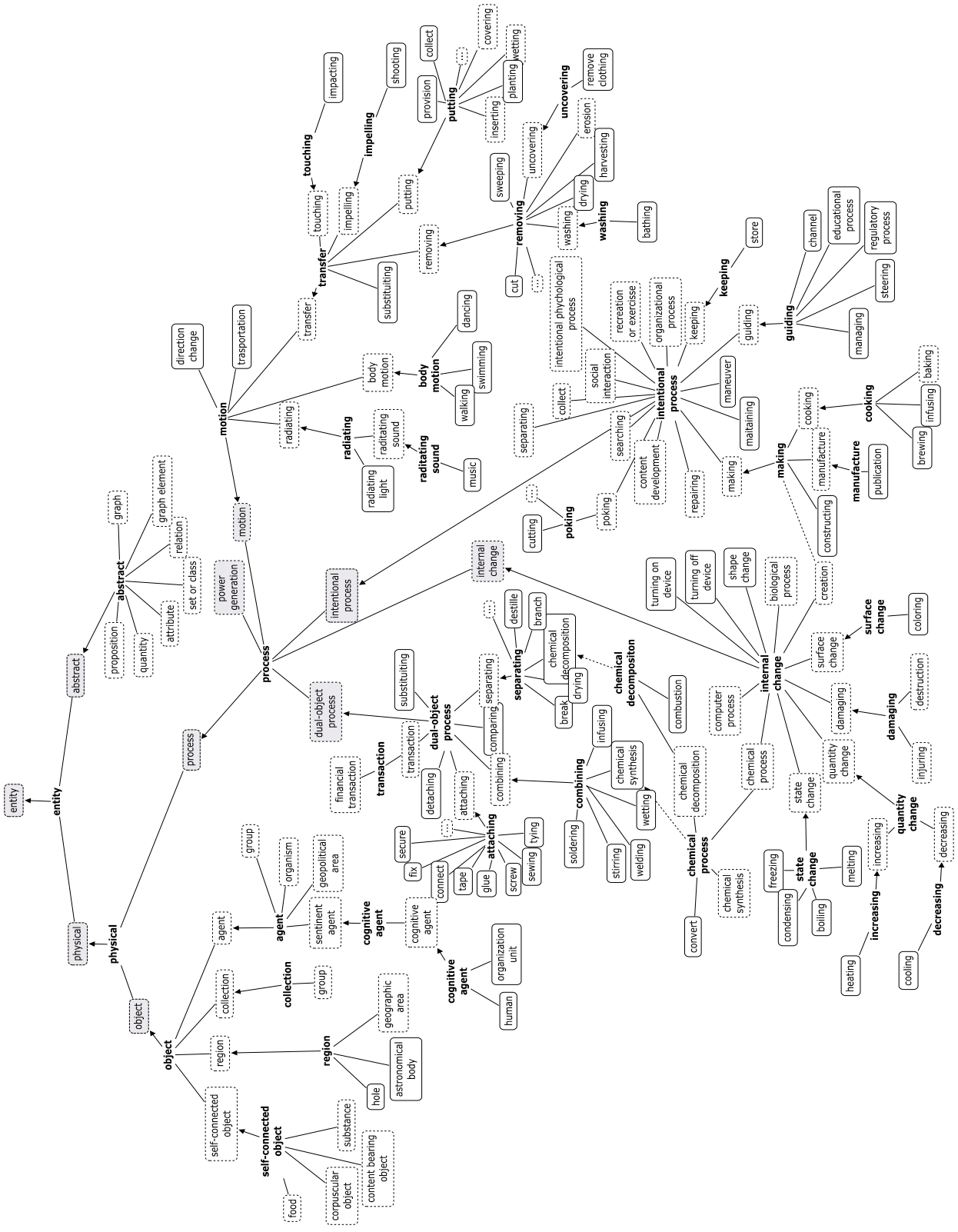


Figure 7.3 Upper ontology as proposed by SUMO [107]

Definition of technical processes is given by [17] as a process of technical product usage in which necessary effects are need for purposeful transformation of operands; hence, technical product is also an operator within technical process. Only the root node *cause of a change* in Figure 7.1 as provided by the *WordNet* specifies process like definition as cause to change, make different and cause a transformation provided in much broader sense than required by the engineering applications.

Process branch of SUMO (see shaded terms in Figure 7.3) with its derived instances bringing together motion, transfer, internal change, internal process as making or creation and dual-process defined as any process that requires two, non-identical patients (in the scope of this thesis patient equals operand), creates a rational basis for engineering applications. In Figure 7.3 only the initial taxonomy of SUMO with some extensions in respect to processes is shown.

7.3.3 Functional basis for engineering design

It was shown [21], [22] that is necessary to reach common and shared understanding of product's functions in order to enable unbiased communication between product development process participants. Functional decomposition that is performed by designer, like decomposition of TP's, requires at least a taxonomy, or standardisation of terms being used that is, to enable understanding between participators of the product development process. The need for standardization is even more emphasized especially if considering computational reasoning systems. Unlike the *WordNet* which was constructed based on the epistemology of the general terms as natural language, or as in the case of the SUMO which extended the *WordNet* further towards more specific engineering domains, defining functional basis was driven only by the present design methodologies and research papers in the field of engineering design [22]. Differences and similarities identified within considered methodologies like Systematic Design [4] and TRIZ [5], and research papers manage to result with product function taxonomy. The expected impact of creating functional basis was intended towards lessening of the ambiguities which occur at the function modelling level as a result of similarities between terms applied for description of the same function. By specifying vocabulary the efficiency of the function modelling could be increased in respect to the effort necessary to process, interpret and facilitate the exchange of information about technical product. If bearing in mind artificial reasoning systems utilized for any form of automation of design than synonyms which are regular and understandable within a context of

a natural language could as such present an impeccable barrier to inference system of a computer. Thus, at least with a tendency to follow principle of parsimony choosing a minimalist approach to define vocabulary of function modelling in spite of issues which may emerge since not all of the concepts could be described in the way only as prescribed. Embracing a standard vocabulary a long term repositories of technical products' function models could be established. Consequently the whole project was accepted by the NIST as a standard.

Table 7.1 shows taxonomy of operands, or as more commonly referred to as taxonomy of operand flows, which are as such being proposed by functional basis [21], [22]. Operand taxonomy is composed as a result of extensive in-domain scientific literature review including reconciliation of functional modelling terms adopted from known design methodologies like Systematic design from Pahl and Beitz [4] and Altshuller's TRIZ [5]. The truth is that the taxonomy presented in Table 7.1 is not directly derived from TTS, however the TTS itself follows Systematic approach as its natural precursor [4] but only addressing operand taxonomy in its basics specifying operands as materials, energy or signal without further more detail specification. Further extension of taxonomy presented in Table 7.1 is possible if for each of the operands additional attributes would be considered and then defined following the SUMO and the *WordNet* respectively. These would be able to accept their change of states according to predefined type graph structure. This thesis, however, will adopt state change and internal change as proposed by SUMO (Figure 7.3) in conjunction with taxonomy of operands as in Table 7.1 only in a form of suggestion required when defining production rules. The classification of operands remains the same whether they are transformed within technical process as a requirement to satisfy existing market needs or within technical product itself as required by technical process to deliver the necessary effects. Label formation is adopted as proposed by [21] and [22], where term applied must contain level of interest combined with its class root, e.g. optical energy where the former is a tertiary and the latter is a class/primary term. Moreover, a power conjugate is provided for bond-graph system modelling driven by propagation of energy flows, thus offering more precise description of energy type.

Table 7.2 shows taxonomy of technical products' functions. Similarly to operand flows, the taxonomy of functions emerged from the reconciliation of different taxonomies as proposed by Pahl and Beitz, Hundal and Altshuller [22]. Approach involved an analysis of each of the terms in respect to already constructed structure.

Table 7.1 A taxonomy of operands (operand flows) as accepted by NIST [22]

Class (Primary)	*Secondary*	*Tertiary*	*Correspondents*
Material	Human		Hand, foot, head
	Gas		Homogeneous
	Liquid		Incompressible, compressible, homogeneous
	Solid	Object	Rigid-body, elastic-body, widget
		Particulate	
		Composite	
	Plasma		
	Mixture	Gas-gas	
		Liquid-liquid	
		Solid-solid	
		Solid-Liquid	Aggregate
		Liquid-Gas	
		Solid-Gas	
		Solid-Liquid-Gas	
		Colloidal	Aerosol
Signal	Status	Auditory	Tone, word
		Olfactory	
		Tactile	Temperature, pressure, roughness
		Taste	
		Visual	Position, displacement
	Control	Analog	Oscillatory
		Discrete	Binary
Energy	Human		
	Acoustic		
	Biological		
	Chemical		
	Electrical		
	Electromagnetic	Optical	
		Solar	
	Hydraulic		
	Magnetic		
	Mechanical	Rotational	
		Translational	
	Pneumatic		
	Radioactive/Nuclear		
	Thermal		

If a term overlaps and if a term is not a synonym of an existing term and it is a subset of that term, than it would be placed lower in the taxonomy; the other way around occurs when a new term presents a superset, thus placing the term above in the hierarchy the corresponding structure. Synonyms adjoined rather than added as new terms in the hierarchy. According to TTS [1], the functions of technical process are equated in one-to-one mapping to processes.

Processes that are performed as vertical transformation, or vertical action chain that is (see Figure 2.7), are those which are performed within and only by technical product itself. It is possible only to speculate why have the authors of the TTS choosen to put focus on product functions instead of technical system bound processes although they are equivalent and following Pahl and Beitz legacy might be one plausible explanation.

Table 7.2 Taxonomy of technical products' functions as accepted by NIST [22]

Class (Primary)	*Secondary*	*Tertiary*	*Correspondents*
Branch	Separate		Isolate, sever, disjoin
		Divide	Detach, isolate, release, sort, split, disconnect, subtract
		Extract	Refine, filter, purify, percolate, strain, clear
		Remove	Cut, drill, lathe, polish, sand
	Distribute		Diffuse, dispel, disperse, dissipate, diverge, scatter
Channel	Import		Form entrance, allow, input, capture
	Export		Dispose, eject, emit, empty, remove, destroy, eliminate
	Transfer		Carry, deliver
		Transport	Advance, lift, move
		Transmit	Conduct, convey
	Guide		Direct, shift, steer, straighten, switch
		Translate	Move, relocate
		Rotate	Spin, turn
		Allow DOF	Constrain, unfasten, unlock
Connect	Couple		Associate, connect
		Join	Assemble, fasten
		Link	Attach
	Mix		Add, blend, coalesce, combine, pack
Control Magnitude	Actuate		Enable, initiate, start, turn-on
	Regulate		Control, equalize, limit, maintain
		Increase	Allow, open
		Decrease	Close, delay, interrupt
	Change		Adjust, modulate, clear, demodulate, invert, normalize, rectify, reset, scale, vary, modify
		Increment	Amplify, enhance, magnify, multiply
		Decrement	Attenuate, dampen, reduce
		Shape	Compact, compress, crush, pierce, deform, form
		Condition	Prepare, adapt, treat
	Stop		End, halt, pause, interrupt, restrain
		Prevent	Disable, turn-off
		Inhibit	Shield, insulate, protect, resist
Convert	Convert		Condense, create, decode, differentiate, digitize, encode, evaporate, generate, integrate, liquefy, process, solidify, transform
Provision	Store		Accumulate
		Contain	Capture, enclose
		Collect	Absorb, consume, fill, reserve
	Supply		Provide, replenish, retrieve
Signal	Sense		Feel, determine
		Detect	Discern, perceive, recognize
		Measure	Identify, locate
	Indicate		Announce, show, denote, record, register
		Track	Mark, time
		Display	Emit, expose, select
	Process		Compare, calculate, check
Support	Stabilize		Steady
	Secure		Constrain, hold, place, fix
	Position		Align, locate, orient

However, the latter does not diminish issues which appear when technical processes are not being considered, it is quite the contrary since being unaware of underlying reasoning can hinder the search even more. Of course, technical processes are broader in scope since they involve more types of technological principle related operations, but still the classification of basic or common operations is the same no matter which action chain is considered. This is an explanation of why to reuse work done to create basis for technical product's function

modelling and then to broaden it with process specific operations (as in Figure 7.1 and Figure 7.3) in the form of suggestion or guidelines for production rules definition.

7.4 Examples of methods application

The knowledge formalisation about technical processes, technological principles and necessary effects shown within these examples is conducted using functional basis expanded with process related terms as suggested by the *WordNet* and SUMO. Examples of different levels of technical systems complexity will be considered. The two examples that are presented here are intended to serve as a proof of concepts showing all of the possibilities and drawbacks of developed method for generation of operand transformation variants. These findings will provide foundations for the future research.

7.4.1 Tea-brewing process

The decomposition of the technical process of tea-brewing is adopted from the literature [7] and will serve as a first example of the method's application. The formulated task is the design of an automated tea-brewing machine [7]. The task assumes that the energy needed for the heating of water is provided by the technical system that is ought to be designed. The black-box process formulated according to requirements is shown in the following figure:

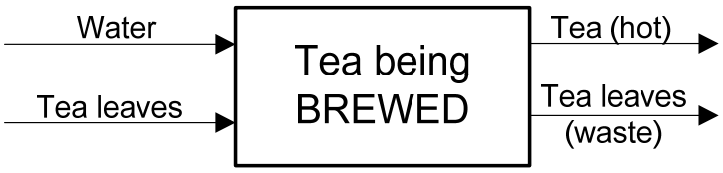


Figure 7.4 Tea-brewing black-box process formulation

The possible technologies for tea preparation are numerous some of which are very well known. Thus the search performed by the method developed within thesis probably will not yield an innovative solution, but still the results provided will gave insights to designers what effects are required to sustain operand transformation process. In Figure 7.4 the input to the search is specified in the process' black-box representation. Operands in their required input states are water and tea leaves, and the desired output states are tea (hot) and tea leaves (waste). Provision of energy for heating by the machine is also checked during the decomposition. The search objective function is formulated as the minimization of the number of operations needed to accomplish the required transformation. Knowledge about technical processes, technological (working) principles and necessary effects is formalized as proposed

by the knowledge formalization foundations provided within this Chapter (Section 7.3). Formalization guidelines included the Functional Basis (Table 7.1, Table 7.2), the *WordNet* (Figure 7.1) and the SUMO (Figure 7.3).

What designer has to do first is to formalise knowledge about tea-brewing process thus creating tea-brewing graph-grammar. First the method requires definition of *TP entities* Σ_G. Then, designer has to map these to labelled multidigraph for representation of technical processes (Definition 6.3) of size $|V| = 4$, thus creating set of rule building blocks. For an example the black-box process formulation as shown in Figure 7.4 can be used (*in*, *out* and *eff* nodes omitted from picture). These building blocks can be used to specify left hand sides of graph-grammar productions $p: L \rightarrow R$ of size $|V_L| = 4$, or in combination with other building blocks to form right hand sides with $|V_R| \geq 4$. Table 7.3 shows context-free grammar CFG_{TP} of tea brewing processes expressed in BNF with its alphabet Σ_{Op} (Definition 6.10). Grammar was enumerated in accordance with Section 5.4 and expressions in (5.14) to be able to apply grammatical evolution.

Table 7.5 shows graph-grammar $\mathcal{GG} = (\mathcal{S}, \mathcal{P}_G, \Delta)$ of tea-brewing which is defined in accordance to Definition 6.8. Operands $Od \in \Sigma_{Od}$ mapped to arc $e \in E$ are depicted ontop of black head arrows, where effects $Eff \in \Sigma_{Eff}$ (*Human force*, *regulation* and *energy*) are shown over default arrows. For simplicity reasons left hand side of the productions $p: L \rightarrow R$ are represented only as tokens as represented in Table 7.3, where the right hand sides of productions are shown in full as multigraphs, thus operation of brewing defined with token $< brewed >$, in multifigraph representation equals black-box formulation as given in

Table 7.4. That holds for each of the tokens. The graph transformation algorithm for decomposition of technical processes performs as defined in Chapter 6 and within pesudo-code given in (6.4):

- rewritting procedure for each of decomposition steps is followed as prescribed by token sequence (Table 7.3),
- each of the $p: L \rightarrow R$ is applied by first indetifying soroundigns of L by determining match $m(L)$ in host graph (current derivation step) as defined in Defintion 6.6,
- L is replaced with R,
- conecting procedure ρ is applied to connect R to host graph's structure (Section 6.5)

Technical process synthesis is depicted as derivation tree in Figure 7.5. Production application sequence all possible theoretical variants that can be created providing grammars as defined in Table 7.3 and Table 7.5. Optimal variant with minimal number of operations is shaded.

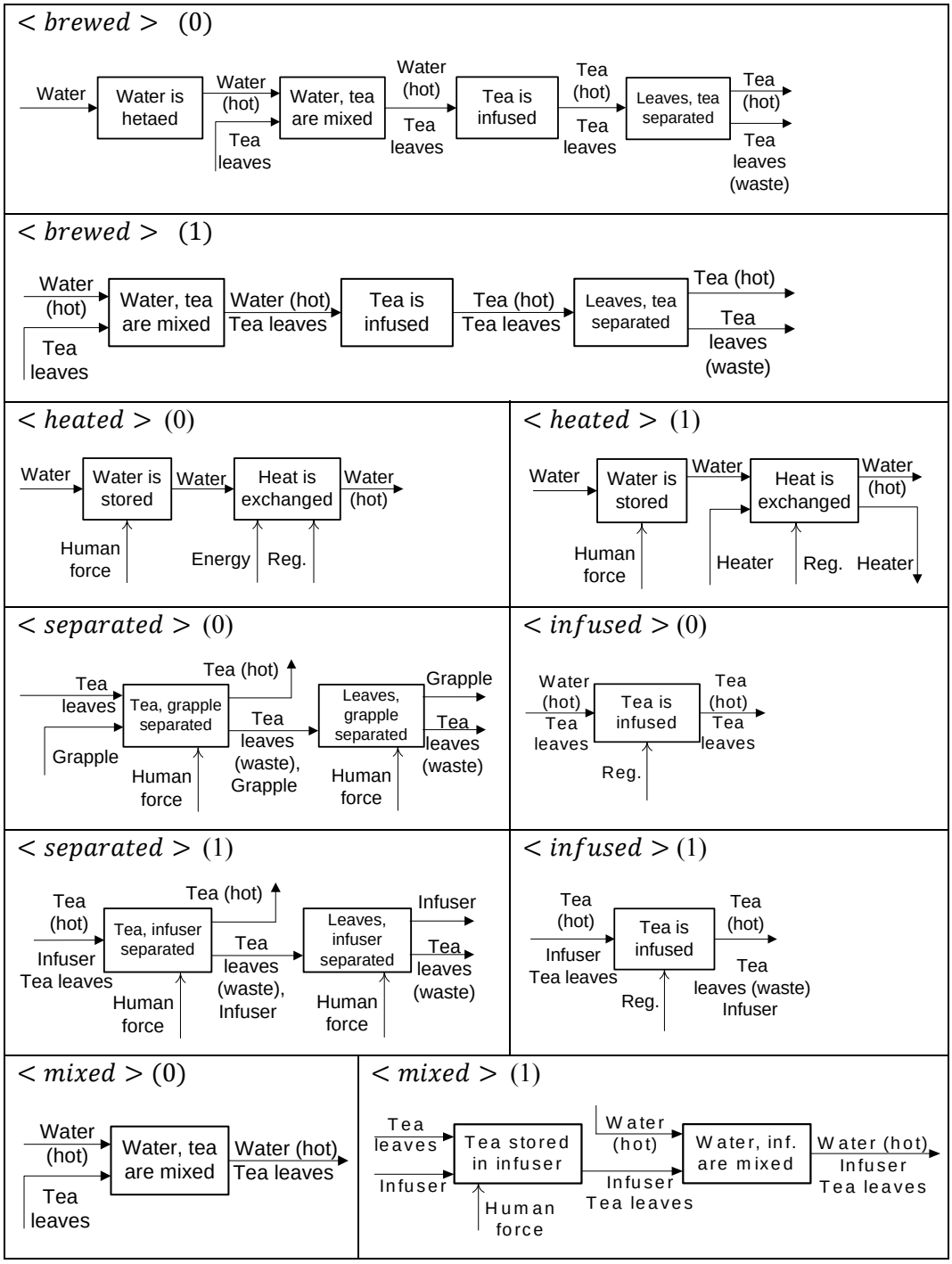
Table 7.3 Context-free grammar CFG_{TP} of tea-brewing process in BNF

			Rule alternatives
	$\Sigma_s = \left\{ \begin{matrix} \_infused, \_infused', \_stored, \_stored', \\ \_exchanged, \_exchanged', \_mixed, \\ \_stored, \_separated, \_separated', \\ \_separated'', \_separated''' \end{matrix} \right\}$		
	$\mathcal{V}_s = \left\{ \begin{matrix} brewed, heated, infused, mixed, \\ separated, stored \end{matrix} \right\}$		
	$\mathcal{S}_S = < brewed >$		
	Production rule set $\mathcal{P}_S$:		
	$< brewed >$	$::= < heated >< mixed >< infused >< separated >$	(0)
		$\mid < mixed >< infused >< separated >$	(1)
	$< heated >$	$::= \_stored \_exchanged$	(0)
		$\mid \_stored \_exchanged'$	(1)
	$< mixed >$	$::= \_mixed$	(0)
		$\mid \_mixed\_stored'$	(1)
	$< separated >$	$::= \_separated\_separated'$	(0)
		$\mid \_separated''\_separated'''$	(1)
	$< infused >$	$::= \_infused$	(0)
		$\mid \_infused'$	(1)

Table 7.4 Correspondence between CFG_{TP} and TTS

$\boldsymbol{CFG_{TP}}$	**TTS**	**Remark**
Σ_s	Operations	No more decompositions possible
$\mathcal{V}_s$	Sub-processes	Can be decomposed further
$\mathcal{S}_S$	Technical process	Starting point
$\mathcal{P}_S$	Formalized knowledge about TP -showing only operation sequences	Set of productions $\alpha \rightarrow \beta$

Table 7.5 Graph-grammar of tea-brewing



To explain the derivation process in Figure 7.5, triggered rules are labelled by their left-hand-side with the applied rule alternative following in brackets. Rewriting rules $< brewed > (0)$ and $< brewed > (1)$ denote two different processes, the first carries out automated heating of the water and the second assumes the water is already heated when it enters the process. For this reason, given that water (hot) is not an input to the starting black box model (Figure 7.5), the sequence on the right hand side in Figure 7.5 is infeasible, thus purely theoretical.

Branch $< heated > (1)$ is determined to be unfeasible as well, since it requires heat from an external object and not from the product itself.

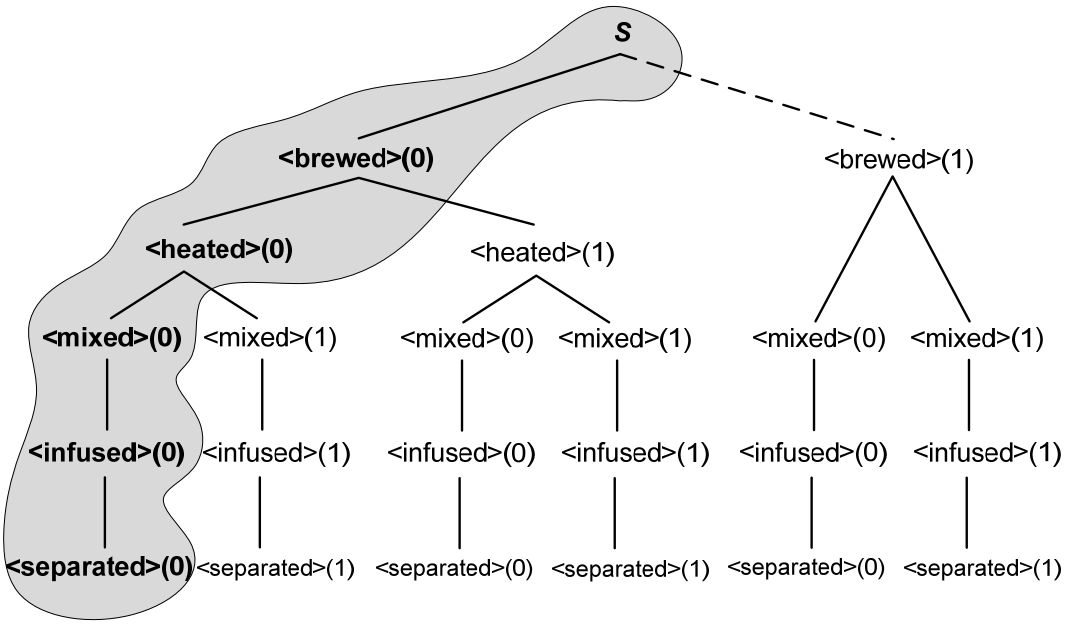


Figure 7.5 Production application sequence of tea-brewing process (only left hand sides of applied production is shown, unreachable branch expressed as dashed)

Assumption is that the heat is provided as an effect from the product. Finally, since both of the branches under the $< heated > (0)$ sub-process are feasible under the given requirements, the $< mixed > (0)$ symbol has one transformation less, therefore designating optimal rewriting sequence as shown in Figure 7.5. Optimal working principles under given criteria and based on black-box in Figure 7.4 would consist of the following set of operations: $\_stored$, $\_exchanged$, $\_mixed$, $\_infused$, $\_separated$, $\_separated'$. Identified effects beside the specified energy provision are the regulation of the infusion and the heating process, allowing a human operator to store the water and use an outside object to hold the tea leaves.

7.4.2 Design of stiffened panel assembly line

The formulated task is the design of an automated assembly line that is able to deliver stiffened panels. What designer needs to gain are insights about working principles on which transformation of operand is performed, as well as necessary effects that need to be provided to sustain the transformation. Within this example's grammar, the process of stiffened panel assembly is divided within three logical steps: step one is positioning of steel plates and their assembly and welding, step two comprises of cutting of panel to desired dimensions and then,

surface cleansing and setting markings for placing of stiffeners, and final the step three comprises of stiffener transport and their positioning and welding. Step 3 is concluded with further distribution of welded panel. Design of an assembly line is a complex process involving solutions which may contain multitude of different technical systems, depending on the required effects.. Black-box of such process as it might be specified by designer with operands in their initial and desired states is given at in the following picture:

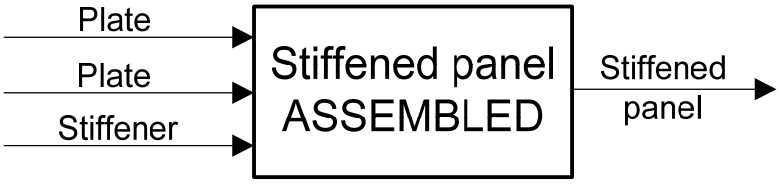


Figure 7.6 Stiffened panel assembly black-box process formulation

In Figure 7.6 the input to the search is specified in the process' black-box representation. Operands in their required input states are both plates and a stiffener, and the desired output is a stiffened panel. Effects are rendered as unknowns. The search objective function is formulated as the minimization of the number of operations needed to accomplish the required transformation; goal is set to find automated procedure involving pneumatic based securing of the panel structural elements. Additional stooping rule is introduced if iteration exceeded 100 derivations. It was necessary to apply such condition since recursive rule have been applied (see Table 7.6 at $< plateWeld > (0)$, which represents two sides welding procedure involving intermediate stage of turning the steel plate. Two types of welding are considered manual arc welding (see Table 7.6 and Table 7.7 at $< plateWeld^1 > (0)$ and) and submerged fully automated arc welding under a granulate flux (sand) (see Table 7.6 and Table 7.7 at $< plateWeld^1 > (1)$) Knowledge about technical processes, technological (working) principles and necessary effects is formalized as proposed by the knowledge formalization foundations provided within this Chapter (Section 7.3). Formalization guidelines included Functional basis (Table 7.1, Table 7.2), *WordNet* (Figure 7.1) and SUMO (Figure 7.3).

Table 7.6shows context-free grammar CFG_{TP} of stiffened panel assembly process expressed in BNF with its alphabet Σ_{Op} (Definition 6.10). Grammar was enumerated in accordance with Section 5.4 and expressions in (5.14) to be able to apply grammatical evolution. A correspondence between CFG_{TP} and TTS is given in Table 7.6. It is assumed that the knowledge was formalised prior a designer actually applies a tool

Table 7.6 Context-free grammar CFG_{TP} of stiffened panel assembly process in BNF

			Rule alternatives
	$\Sigma_s = \left\{ \begin{matrix} \_platePos, \_panelRelease, \_panelTurn, \_plateSec', \\ \_plateSec'', \_plateWeld', \_plateWeld'', \_maw, \_saw \\ \_granRemoved, \_plateSec', \_plateSec''', \_plateSec'''' \\ \_panelCut, \_stfPos, \_brush, \_blast, \_abrSeparated' \\ \_panelPos, \_stfSec', \_stfSec'', \_stfSaw, \_abrSeparated'' \end{matrix} \right\}$		
	$\mathcal{V}_s = \left\{ \begin{matrix} spa, assembled, treated, stiffened, plateWeld, plateSec, plateWeld^1 \\ treated, dirtRemoved, stfWeld, stfSec \end{matrix} \right\}$		
	$\mathcal{S}_S = < spa >$		
	Production rule set $\mathcal{P}_S$:		
	$< spa >$	$::= < assembled >< treated >< stiffened >$	(0)
	$< assembled >$	$::= \_platePos < plateWeld >$	(0)
	$< plateWeld >$	$::= < plateSec >< plateWeld^1 > \_plateRelease$	(0)
		$\mid < plateWeld > \_panelTurn < plateWeld >$	(1)
	$< plateWeld^1 >$	$::= \_maw$	(0)
		$\mid \_saw\_granRemoved$	(1)
	$< plateSec >$	$::= \_plateSec'$	(0)
		$\mid \_plateSec''$	(1)
		$\mid \_plateSec'''$	(2)
	$< treated >$	$::= \_panelCut < dirtRemoved > \_stfPos$	(0)
	$< dirtRemoved >$	$::= \_blast\_abrSeparated'$	(0)
		$\mid \_brush\mid$	(1)
	$< stiffened >$	$::= \_panelPos < stfWeld >$	(0)
	$< stfWeld >$	$::= < stfSec > \_stfSaw\_abrSeparated''$	(0)
	$< stfSec >$	$::= \_stfSec'$	(0)
		$\mid stfSec''$	(1)

Table 7.7 Graph-grammar of stiffened panel assembly (part I)

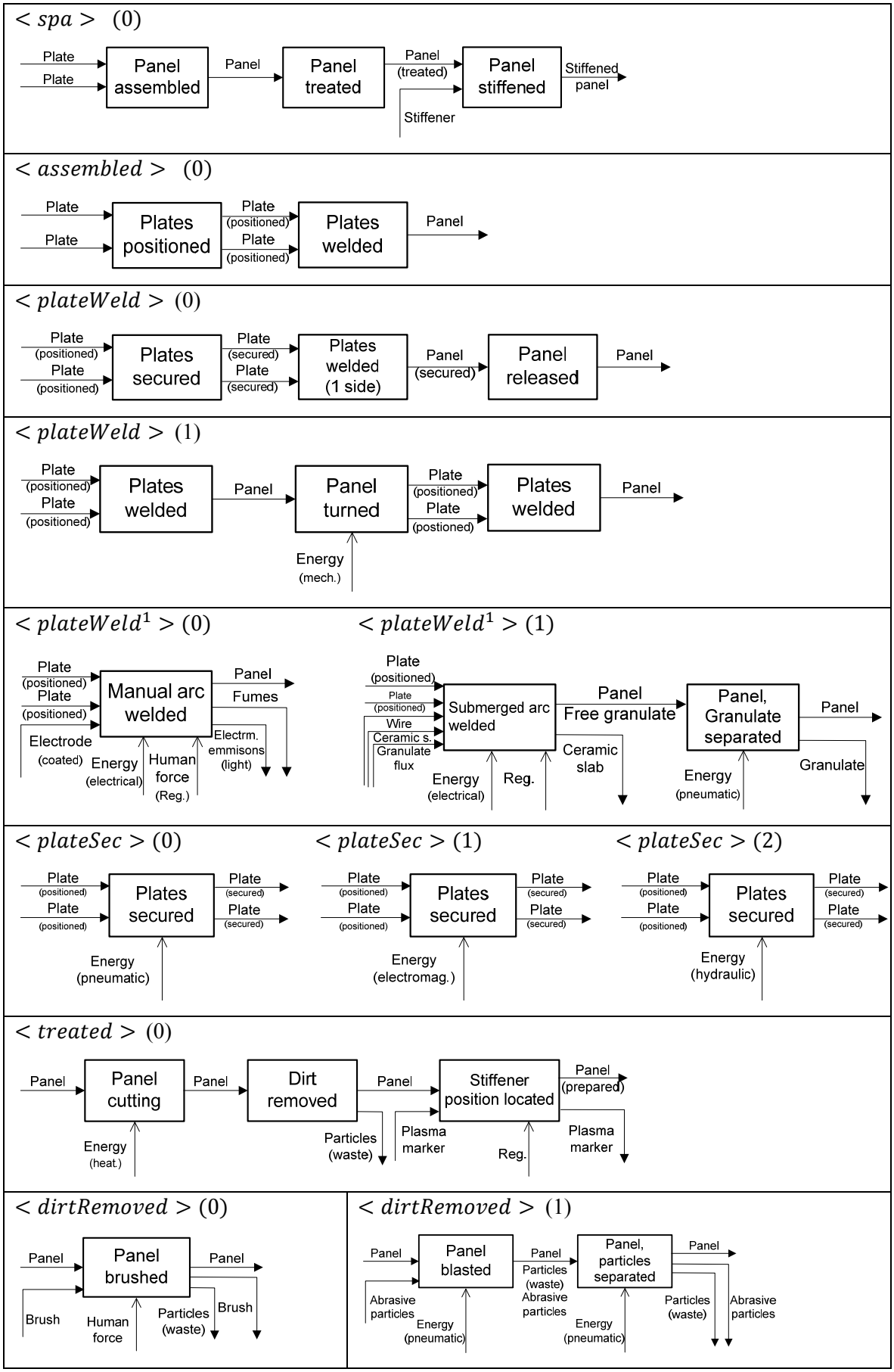
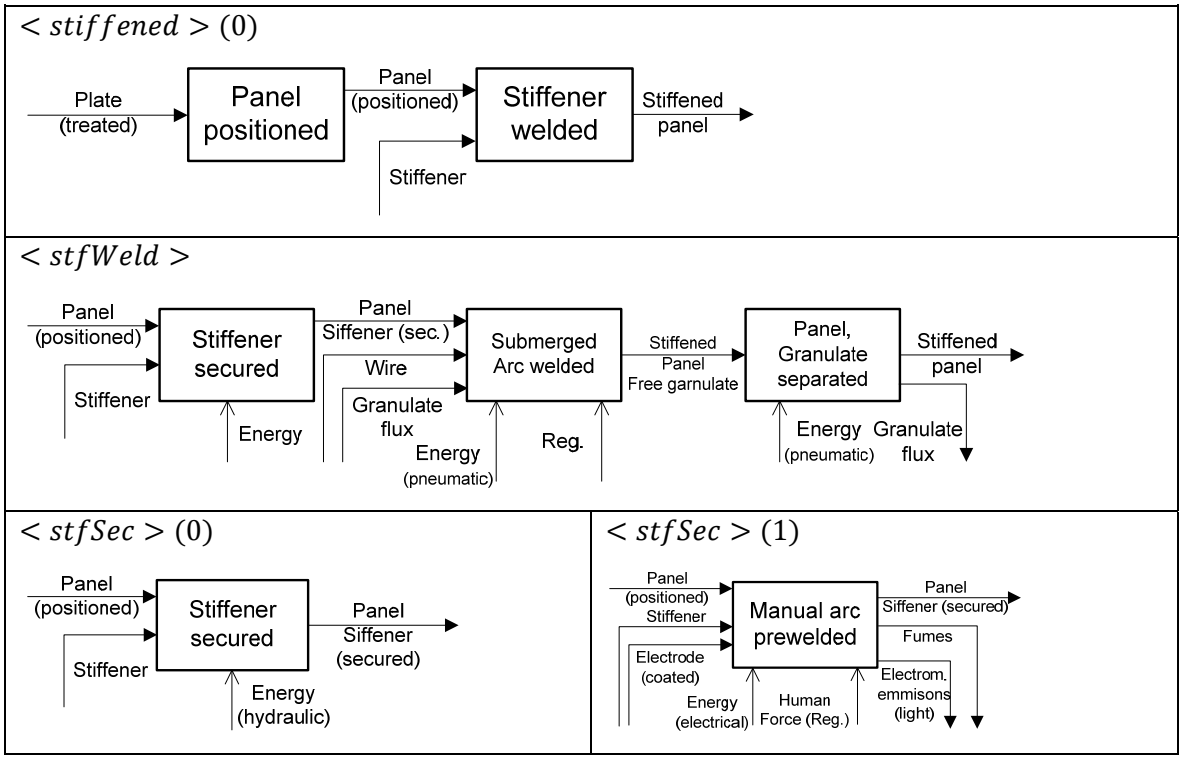


Table 7.8 Graph-grammar of stiffened panel assembly (part II)

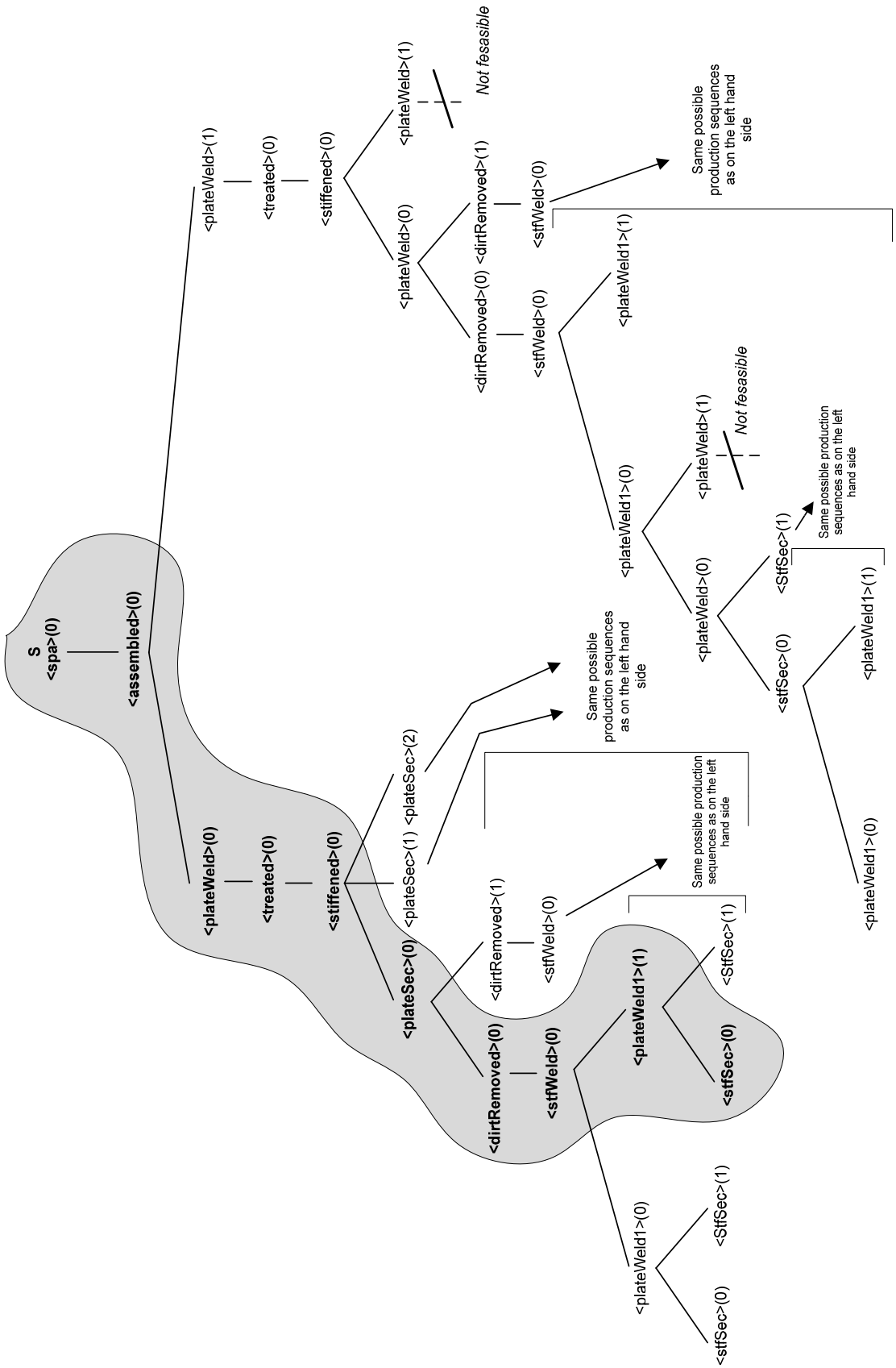


Technical process synthesis is depicted as a tree in Figure 7.7 showing all theoretically possible production application sequences. all possible theoretical variants that can be created providing grammars as defined in Table 7.6, Table 7.7 and Table 7.8. Optimal variant with minimal number of operations under the given criteria is gray-shaded.

7.5 Implications to this thesis

The method for generation of operand transformation variants presented within this thesis is focused on the in-domain knowledge thus creating an extensible and robust problem solver. The same knowledge can often be used in several ways; extending and intertwine at the knowledge level is often simpler than doing the same to the programming code. Thus, the focus can be put just at formalisation, what is truly necessary for and relevant to the domain of interest. Knowledge driven systems are easily adaptable for the domains exhibiting similar solution modelling principles, and since the early design stages opt towards graph representation it is intended to as a part of the further research efforts to cover the whole of the conceptual design phase. To maintain an amount of the consistency among the production rules at least a shared understanding in respect to technical process related terms must exist. In order to define production rules the definitions about terms used must be clear to users in order to be able for the computational system to produce coherent results in the end.

Figure 7.7 Production application sequence of tea-brewing process (only left hand sides of applied production is shown, recursive branches expressed as dashed, goal gray-shaded)



Thus, a prerequisite for knowledge generalization is at least having taxonomy of related terms thus creating possibilities to organize knowledge more efficiently and if necessary to apply predicate calculus of order as felt fit. Generalisation enables to put forward only what is necessary to describe each of the objects, thus eliminating the irrelevant details and maintaining the production rule redundancy. Moreover, the proposed method offers possibilities for induction of new grammar if desired. Both the extension of the proposed model to other stages of product development and utilisation of grammar induction to create and add bottom-up navigation possibilities thus creating an iterative search process are set as aims of the future research. The examples shown within this Chapter are label bounded having no knowledge generalisation possibilities what so ever. As defined in Chapter 6 operands, effects and operations attached to multi-digraph's edges are technical process labels and not objects of their respective classes. The extension to include attributes, types and inheritance and to even define operations as a part of algebraic system over operands and effects would require definition of type graph and typed graph, thus creating a robust system. It is suggested to define type graph of technical processes based on the SUMO ontology (Figure 7.3) and extended with the operand taxonomy as proposed by reconciled functional basis as adopted by the NIST (Table 7.1). The definition of type graphs will enable to utilize the full power of stochastic search of grammatical evolution, since rules would not have to be defined so strict and label bound, but would be instead tied to types of objects.

8. COMPUTATIONAL TOOL'S ARCHITECTURE

In the computer field, the moment of truth is a running program; all else is prophecy (Herb Simon, *taken from The Shape of Automation: For Men and Management* [108]*).*

The aim of this thesis is to provide a support to technical process synthesis which is according to the TTS a step within conceptual development stage. Devising a graph-grammar based method for technical process synthesis provides theoretical fundaments for its implementation as a tool expressed within a computational environment. When completed such tool would offer designers the possibility to computationally explore solution space of technical processes in order to select the most feasible one in respect to existing market and societal needs. To be fast and efficient is a necessity in today's product development process, and computational tools can help provide these features to the overall design process. In the research projects conducted within the Design Science, realisation of a computational tool is considered as a practical part of the research effort, since it allows creation of results as it was once envisaged at the beginning of a project. Most often, the practical objective is a prime motivation behind the existence and realisation of theoretical research objectives. However, development of a computational tool up to a stage in which it is completed to an application-ready state is a daunting task for itself. Thus, the computational tool in this thesis that was completed up to a prototype stage driven by an intention to be somehow able to test and to produce results, however ending up to being much more than a sole computer implementation valuable notion about how to design the method itself have emerged. Hence, design methodology and experience from practice which show us how engineers design, or if referring even more generally than how do designers design, learns us that order of prescribed activities is most often not followed and that it depends both on the individual involved and the amass of external creation process related causes. What is tried to be argued here now is that the implementation of a method within a computational environment can and does teach us a lot about the method itself. Like design of artificial, thesis development is an endless aiming to perfection abductive process, where it's developed and involved concepts having mutual impacts on each other. Thus, the usual strict division [10] to theoretical objectives which encompasses development of a method and to practical objectives as method's implementation doesn't hold since both of them carry parts of each other as a result of their emergence from a conceptually overlapping and iterative process. The latter expresses the

precise way in which the famous Herb Simon's [108] statement put as an opening to this Chapter is viewed upon within this thesis.

This Chapter will try to present in brief the architecture of the computational tool that has been developed based on devised method for generation of operand transformation variants. Class diagrams of labelled multi-digraph and its corresponding data structures, as well as class diagrams of grammatical evolution which are a part of an extensive genetic algorithm framework are developed in-house by the author of this thesis. It was, and still is a part of an effort to create a general purpose engineering problem multi-objective optimiser. Developed optimisation framework together with labelled multi-digraph should provide solid foundations for achieving long term research objective (see Chapter 1) towards creating complete graph-grammar based computational design framework for early design. Computational tool architecture will be represented in brief. The basic data structures will be represented as well. Finally, a graphical user interface screen shots will be shown.

8.1 Architecture of computational tool

The kernel of the computational system that will perform the synthesis on the technical process level is shown in Figure 8.1. The BNF library is defined by the assistance of a designer through a visual builder interface which comprises a Preparation module. From the author's own experience it has been learned that designing production rule libraries by hand as text files becomes tedious and almost impossible to keep errorless especially when considering structures like labelled multi-digraphs. As a result of a visual user interface for production rule builder has been designed. This has provided an enhancement to overall functionality. BNF rule library should contain a large enough set of rules through which meaningful and useful results can be obtained. In Chapter 7 it was suggested to define production rules by extending functional basis with process related terms according to the *WordNet* and the SUMO ontology. It is a necessity for user to follow these suggestions in order to create a consistent rule database; however no automated consistency check in respect to these suggestions has been implemented yet. Preparation module both stores and retrieves production rules from the BNF rule library as a part of an ongoing rule development process and refinement.

Execution module as shown in Figure 8.1 contains grammatical evolution based search and optimiser that produces operand transformation variants. To create rule derivation sequences,

the solver must make a request to the BNF rule library to find an appropriate rewriting rule according to the expression from the previous section. Information flow between designer, library and solver is possible for the input of the problem formulation, addition of new rules and provision of assistance to the solver.

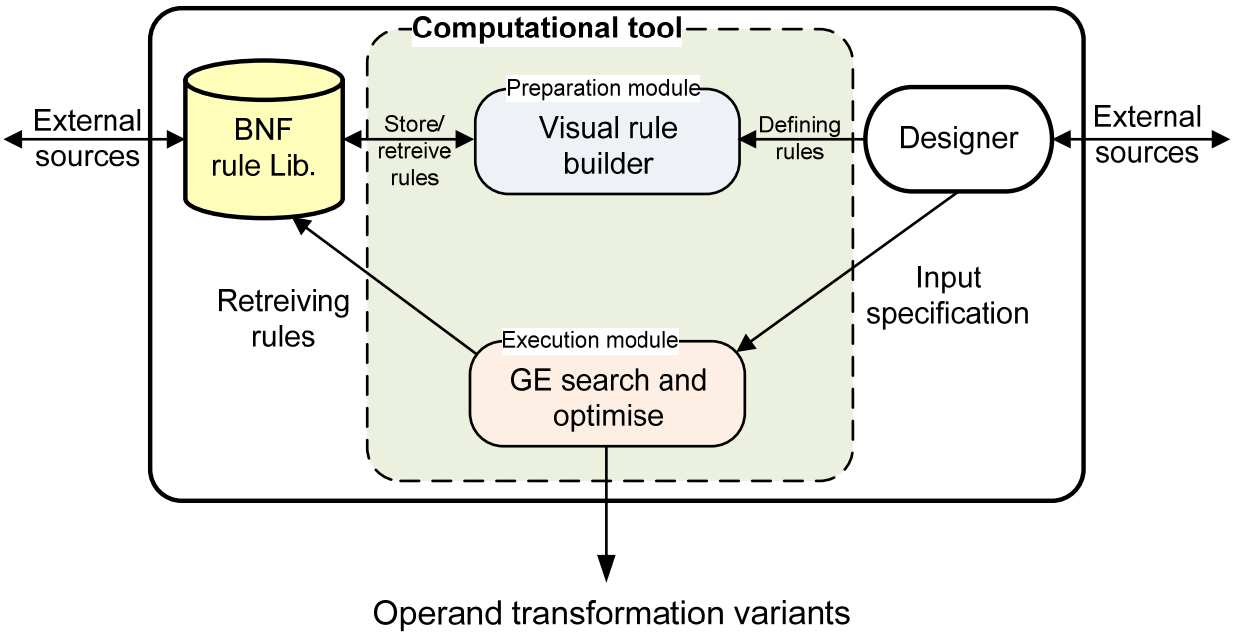


Figure 8.1 Schematics of the kernel of system for generation of operand transformation variants

Both the designer and the BNF rule library can communicate with *external resources* beyond the system boundary to pass or retrieve new rule definitions. External sources for the tool may well include other BNF rule libraries. By choosing the initial axiom symbol, which is a technical process, the designer partially formulates the input information from which the rewriting procedure begins. To fully define the necessary input data, the information about operands and their initial and desired states must be provided. The black box representation of the search formulation is defined by technical operands with input states which are transformed to output states through a general technical process.

Findings about design process, and technical system that has to be designed are equally important for a successful product development [1], [4], [7] and [8]. The Theory of Technical Systems uses same the modelling principles, transformation system paradigm, for technical processes and technical systems and design processes. Thus, design process is defined as a process of information transformation from design specifications based on the existing needs and requirements to complete technical specification of technical system [1]. Operators belonging to executive part of such transformation system are designer and various tools that

aid him or her during the course of design. The method embedded within the computational tool developed in this thesis participates in design process as shown in Figure 8.2:

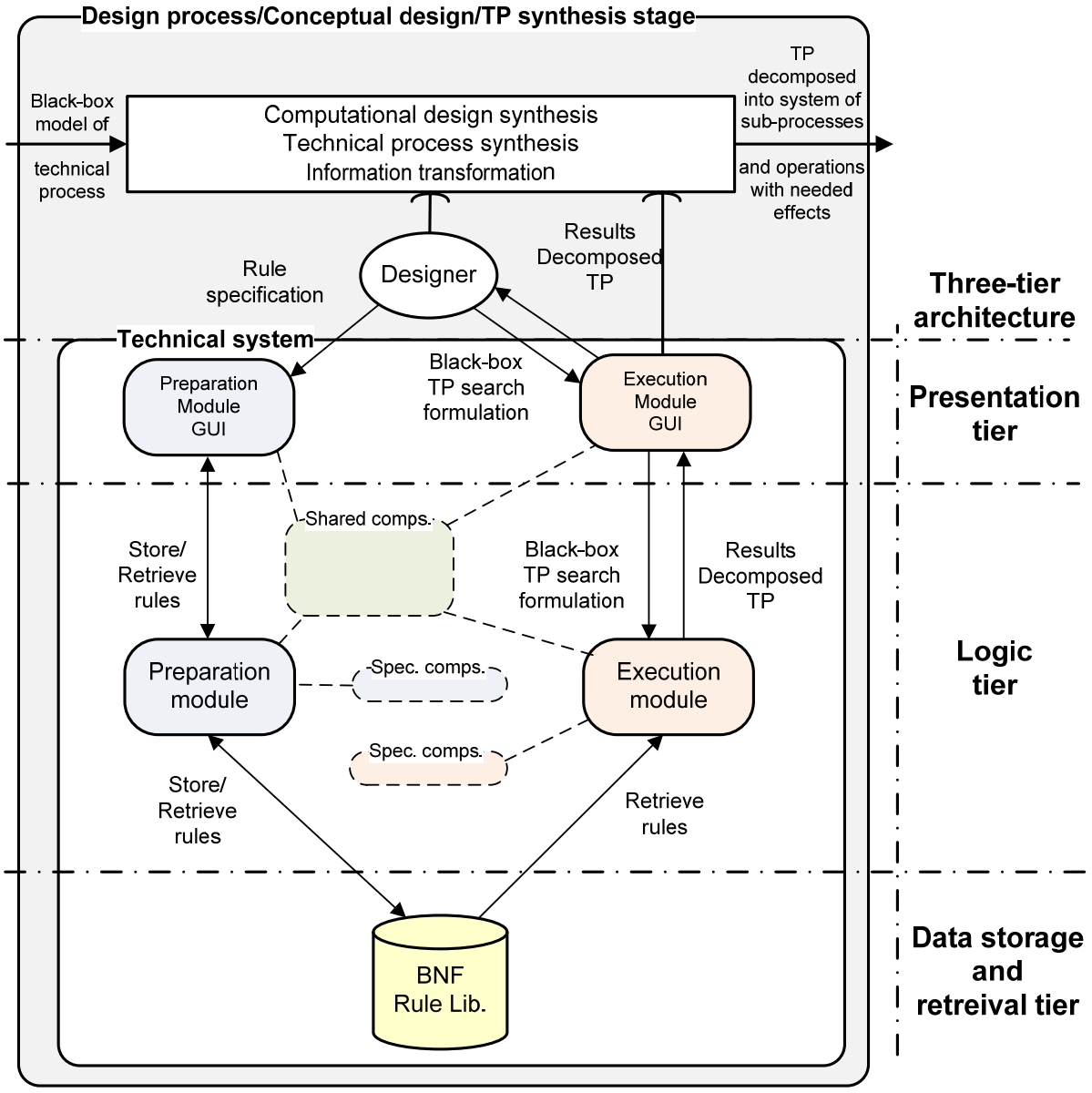


Figure 8.2. Three-tier computational tool's architecture with participation in design process

Figure 8.2 clearly models how designer interacts with the computational tool developed in this thesis, and how such interaction affects design process in order to improve the technical system being designed (see Chapter 1). The effects delivered by designer and computational tools are all of these activities which are necessary to perform computational synthesis of technical processes. It is clearly shown that designer was not replaced by computational tool. Thus, the results provided just advice designer to facilitate better search across the solution space of operand transformation. Tier modelling is often used for representing physical distribution among components of an application [109] in contrast to application's logical

groupings. Figure 8.2 shows three-tier modelling approach for design of computational tool within this thesis. Although at the current development stage computational tool and rule database are located completely at client side, future implementations consider relocating BNF Rule Lib. on server to allow multi-user access. At the moment rule database is developed in MS *Access* and computational tool is programmed in MS C# with the communication between the two done in SQL. The future implementation may consider graphical user interfaces and processing parts as separate applications within distributed client-server online system.

8.1.1 Architectures of Preparation and Execution modules

Processing level is composed of preparation and execution module containing a number of libraries and components. Figure 8.3 presents component diagrams of Database management (DMU) and Graphic management (GMU) units:

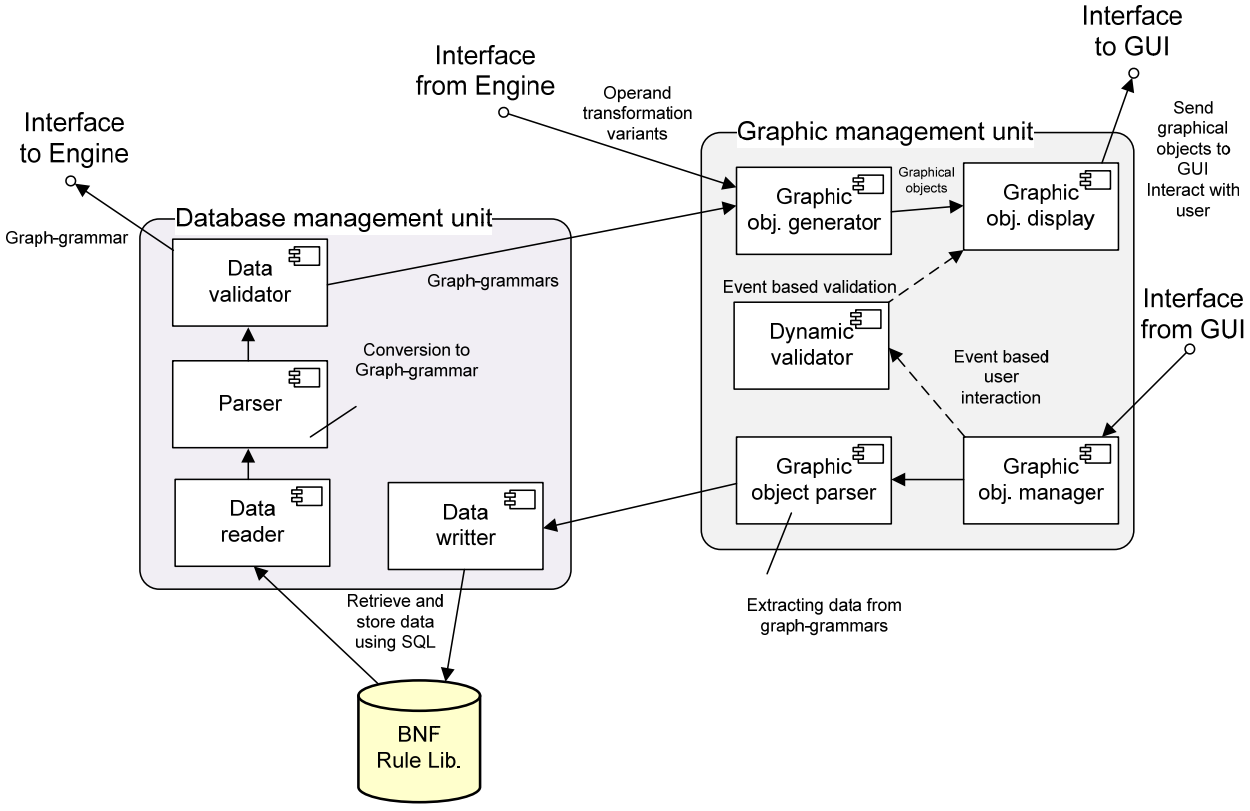


Figure 8.3 Component diagrams of DMU and GMU with data/object flow

Database management unit has a task to establish connection to the database and to retrieve and store production rules. All data that has been retrieved from the database is run through the *Parser* to obtain rule syntax and. Hence, based on data from DMU *Parser* creates appropriate graph-grammar representation using *Multidigraph* and *BNF* dynamic link

libraries. Afterwards, productions are to test the basic rule semantic in order to minimize possible errors which may end up stored inside the database. Errors might have appeared due to lost connection during write process or because of direct intervention within database. Data with errors is discarded as not valid. *Data validator* has an interface towards *Engine* which requires production rules in order to create operand transformation variants.

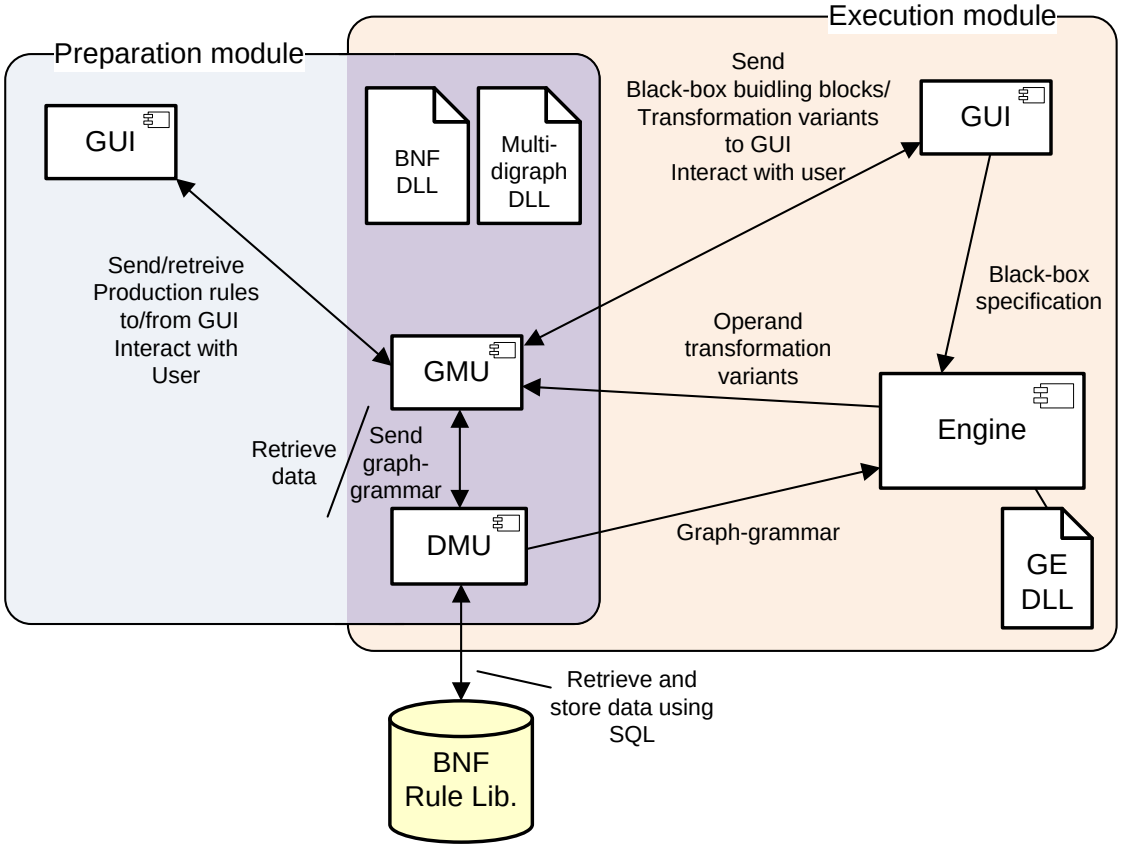


Figure 8.4 Architecture of Preparation and Execution modules with principle components, dynamic link libraries and dataflow

Unlike DMU which is completely procedural, Graphic management unit is partly event based since it interacts with the user. After the validation of the data retrieved from the DMU, the data is being sent to the Graphic objects generator. GMU has to prepare graphical representation whether they are production rules, technical processes or technical process decompositions thus creating proper instances and storing them in the memory. These are then read by the Graphic objects display component and sent via an interface to Graphical user interface (GUI) to be displayed. Hence, only Data writer in DMU and Graphic object parser in GMU are components only used by one module, Preparation module that is. The rest of DMU and GMU components are shared by both modules (Figure 8.4). Interaction with the user is monitored by Graphical object manager. All of the events within GUI are interpreted

and passed further to *Dynamic validator* which informs the user about his or hers actions through Graphic objects display for both modules. When the user decides to save the created rule (Preparation module only), the graphical rule representation is parsed into suitable form, accepted by DMU and written to the database. The architectures of Preparation and Execution modules are shown in Figure 8.4.

Figure 8.4 shows how components and dynamic link libraries are reused among modules. Database interaction through DMU and graphical representation by GMU are developed robust and therefore could be used in both of the main modules. *BNF* DLL for production rules formulation and *Multidigraph* DLL as data model shared by both modules; where *GE* DLL as grammatical evolution library is utilized only by Execution module. *BNF* DLL and *Multidigraph* DLL are used by all of the components.

Execution module consists of several interconnected components as shown in Figure 8.4. The central component of the whole system is simply referred to as the *Engine*. It has a task to generate operand transformation variants as specified by input black-box specification. All of the three available dynamic link libraries are used by this component. Up on execution module start graph-grammar is delivered to *Engine* by DMU. Simultaneously, GMU stores in memory and than delivers necessary graphical representations of TP *Entities* to GUI so that the user can specify black-box input to search. Based on the multidigraph node rewriting principle as specified in Chapter 6 and a set of given constraints and objective functions given universal virtues as specified within Chapter 3, variants are created. Node rewritings are accomplished through *Engine* component's built-in methods with the assistance of *Multidigraph's* own functionalities as specified in Figure 8.5. Operand transformation variants when obtained are passed to the GMU, and than deliver back to GUI.

8.2 Class diagrams

Class diagrams shown in Figure 8.5 are data models of technical process itself alongside all of the relevant technical process entities. The centre class is *MultiGraph* with its duty to represent and sustain transformation in a computational sense. In respect to its mathematical model already shown in Table 6.1, *MultiGraph* is a derived class out of topmost abstract class of *BaseArray<T>* and of *DependenciesFlow* respectively. *BaseArray<T>* is a dynamic matrix of size $m \times n$ composed of n lists of size m accepting via a template *T* any kind of object as an input to its cells. Using a list of lists to structure matrix as a collection of its rows

is just another way of matrix representation differing slightly from the usual array data type. Every row within *Rows* is of type *BaseRow<T>* with its corresponding attributes as shown in Figure 8.5. Although more complex than array, *BaseArray<T>* as structured as it shown allows the application of fast built-in list type methods which are altogether chained through a set of *BaseArray<T>'s* own row and column manipulation methods.

DependenciesFlow is an instance of *BaseArray<FlowArrow>* representing a matrix that has *FlowRow* for each row as an instance of *BaseRow<FlowRow>,* thus accepting the *FlowArrow* type within its cells (as defined within Table 6.1). *FlowArrow* is an instance of abstract *BaseArrow<FlowArrow>* representing a relation between nodes of graph when the *MultiGraph* is instantiated. *DependenciesFlow* was conceived within an intermediate development stage leading towards *MultiGraph* and although the question of programming pragmatics could be raised here, the structure of *DependenciesFlow* survived as a legacy part of an effort to design a *MultiGraph* class. *FlowArrow* represent a bag of arcs that can be set between two nodes of graph. Attributes of a *FlowArrow* class contain pointers to nodes within source and target for easier accessibility. *FlowArrow* is a collection of arcs between two nodes. Moreover, *FlowArrow* class accepts single or a collection of operands thus representing operand flows between consecutive operations. Since *DependenciesFlow* accepts collections of *FlowArrows*, thus it is possible to represent complex relations occurring within labelled multi-digraph. *Flow* class is biased as it represents both operands and effects depending on the value of the string within type property. Although such class construction should be avoided it will remain until next source upgrade. As shown class *flow* contains *state* as given by Table 7.2 as a publicly accessible collection defined in order to be able to track history of state transition, in case type property is set as operand. Operations are derived classes starting from abstract class of *BaseNode*. Ultimately two types exist; an operation and a *DummyNode* required for the modelling of source and target of operands coming towards and out of the transformation system. Likewise, the former is also utilised for modelling the source of effects. The similar applies for the source of effects. Finally, these entire aforementioned classes tie up together in *MultiGraph* class which is designed in order to create objected-oriented model of a transformation that is occurring within technical processes. Unlike classes from which it has been derived, *MultiGraph* class is a graph thus accepting a list of operators for its nodes. Correspondence between TP data model and method as defined within Chapter 6 is shown in the following table:

Table 8.1 Correspondence between TP data model and method as defined within Chapter 6.

Class	**TTS/method correspondent**	**Remark**
Operation Inherits *OperationNode* and *BaseNode*	Operation, *TP entity* - Definition 6.1, Graph labelling $l_V: V \rightarrow \Sigma_{Op}$, Definition 6.2.	If operation can be decomposed than it is referred to as sub-process. Top most sub-process is a process.
DummyNode Inherits *OperationNode* and *BaseNode*	Source and target of operand flows, Source of effects, Graph labelling $l_V: V \rightarrow \Sigma_{Op}$, Definition 6.2.	Add just for technical process modelling purposes to avoid dangling arcs within multidigraph.
Flow	Operand or Effect, *TP entity* - Definition 6.1, Graph labelling $l_E: E \rightarrow \Sigma_{Od} \cup \Sigma_{Eff}$, Definition 6.2.	Operand or effect depending on the value of *type* property.
FlowArrow Inherits *BaseArrow*	Bag of arcs e, Incidence matrix r_{mn} Table 6.1, Source and target mappings, $s: E \rightarrow V$, $t: E \rightarrow V$, Definition 6.2.	Accepts Flow(s) (Operand or effect)
FlowArrows	Row of incidence matrix, Table 6.1.	Accepts FlowArrow(s).
FlowRows Inherits *BaseRow*	Container of incidence matrix rows, Table 6.1	Accepts FlowArrows(s).
Multidigraph Inherits *DependeciesFlow* and *BaseArray*	Technical process, Multidigraph $G = (V, E, s, t, l_E, l_V)$, Definition 6.2, Left or right hand side of production rule $p: L \rightarrow R$, Section 6.3.	As described in Chapter 6, left and right hand side of productions are also multidigraphs. Accepts all of the classes.

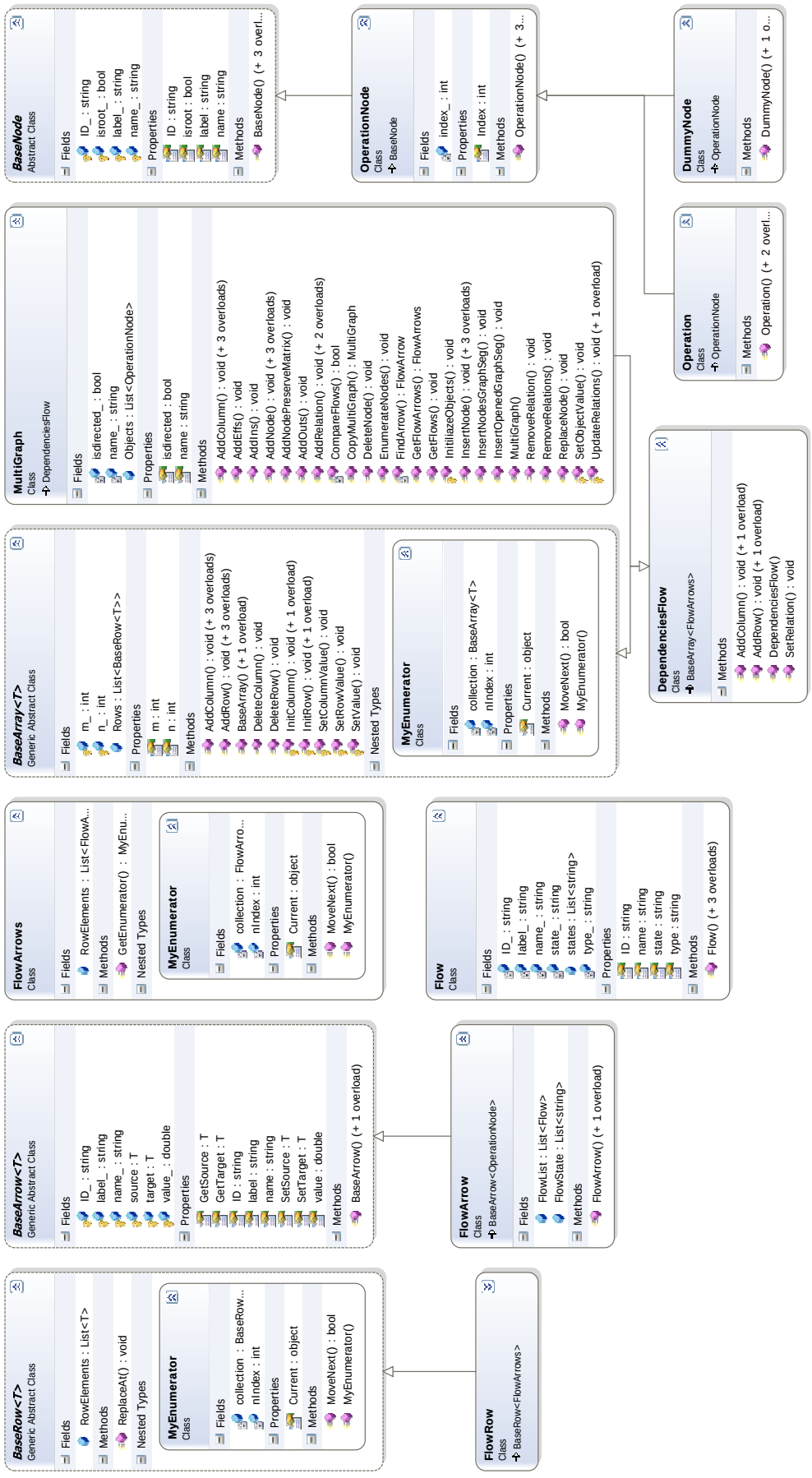


Figure 8.5 Class diagram of *TP Entities* and multi-digraph with operations, operands and effects

Figure 8.6 shows a class diagram of a grammatical evolution framework. Correspondence between GE search and optimisation classes and GE is shown in Table 8.2. Abstract class of individual solution to a problem at hand is a *BaseChromo<T>* again accepting any object through template *T* to become as a chromosomal gene. Attributes of *BaseChromo<T>* contain multiple lists, properties and methods that are used in this thesis or can be used when the results of further research will be implemented. *String_* is a string of *T* denoting a collection of genes that constitute chromosome. Thus, *BinaryChromo* is a class derived as *BaseChromo<byte>* representing common binary genetic algorithm chromosome. Again, the applicability and robustness of grammatical evolution and genetic algorithms can be reemphasized here by. To begin with, an excerpt of a larger MOGA library is applied for a new purpose, to drive grammatical evolution that is, and vice versa the whole grammatical evolution is built on top of an underlying genetic algorithm. Algorithms that are used for optimisation, like MOSGA [110] and NSGA-II [111] classes as shown in Figure 8.6 or single-objective ranking or sorting are thus directly reusable for both GA and GE. Finally, a *GrammarChromo* is derived from *BinaryChromo* with addition of special BNF language and related methods. It can be observed that the list phenotype of *GrammarChrom,o* is a new definition in respect to *BinaryChromo* since it contains a collection of *MultiGraphs* what was necessary to represent the course of technical processes synthesis which is conducted as a derivation procedure resulting in creation of single *MultiGraph* at each of the derivation steps (see example as given in Figure 6.6). *TokenBNF* is used to establish a connection between rules in BNF and graphs.

Table 8.2 Correspondence between GE search and optimisation classes and GE (as in Chapter 5).

Class	**GE/method correspondent**	**Remark**
GrammarChromo Inherits *BinaryChromo* and *BaseChromo*	Population member a as in (5.1) and Table 5.1/ satisfies relation (5.25)	Accepts Multidigraph / stores whole decomposition process
GrammarRecombination Inherits *BaseRecombination*	Binary recombination $P'(t) \leftarrow recombination(P(t), \Theta_r)$, as in (5.13)	
GrammarMutation Inherits *BaseMutation*	Binary mutation $P''(t) \leftarrow mutation(P'(t), \Theta_m)$, as in (5.13)	
GrammarPop Inherits *BasePop*	$P(t)$ as given within (5.1)	Parameters $\mu, \lambda, \Theta_s, \Theta_m, \Theta_r$ defined over population

Figure 8.6 Class diagram of grammatical evolution search and optimisation framework

BasePop<T> is an abstract class used for instancing of the population of individuals depending on template *T* linked to the types of chromosomes and their genes (Figure 8.6). The *Chromosome* list present a collection of chromosomes. Various population and overall algorithm related parameters like population size λ, matting population size, or offspring population size μ are raised to the level of population. Alongside population manipulation and ranking methods, an enumerator class has been encapsulated into population as well as a collection of pointers to the functions required for more robust and reduction in size code programming. *GrammarPop* is directly derived from *BasePop< GrammarChromo >*, since the relation between *BaseChromo* and *GrammarChromo* already exists. Initialization of population is embedded into the respective population class.

Genetic algorithm operators or EVOPs: recombination and mutation are also being shown in Figure 8.6. Abstract class *BaseRecombination<T>* consist of chunks or elements that were programmed as broad in scope, if possible, as permitted by abstract class, thus containing variants of crossover methods, random number generation and selection. *GrammarRecombination* class is instantiated from *BaseRecombination<byte>* as all of the operators take place at the level of an embedded genetic algorithm. Few special delegates used for programming function exchange within recombination methods where also defined. *Recombination* is a general purpose method for gene exchange, where as *CrowdingMatting* is required by the NSGA-II MOGA. Likewise, the *BaseMutation<T>* abstract class is used to derive *GrammaticalMutation* using byte type for *T*. Again, mutation as a randomized bit-flip occurs at the level of genetic algorithm.

What is left to be described in Figure 8.6 are the Pareto optimisation classes, single objective elitism principle class and few helping structures including interface comparers required to apply build-in quicksort over ranked individuals. *BinaryMOSGA* and *BinaryNSGAII*, as well as *BinaryElitism* can be applied directly to GrammarPop that accepts byte type via template. The BNF DLL that is shown in Figure 8.4 is used for production rule definition. It accepts *Multidigraph* as it is necessary to define left and right hand sides of productions.

8.3 GUI

As presented within Figure 8.4 it can be seen that the Preparation and the Execution modules each have their own graphical user interfaces. Since GMU is built event based towards the user than the interaction provided assures less effort for the user in achieving desired

computational support. Such approach enables capturing of the users actions and triggering of the appropriate system's response whenever a prescribed event has occurred. Paramount responsibility of the system is to assure correct and valid rule creation; otherwise the usefulness of the results produced might not be satisfactory. Since the error checking is being implemented at the GUI level by GMU event based *Dynamic validatior*, and within procedural DMU by *Data validator* during data retrieval, there is no need to perform error checking when the method is being implemented within *Engine* component in Execution module. Although not implemented yet, it is be most likely that different application utilities would be required not only to help manage rule syntax, but also to check out and verify the usefulness and impact to the consistency of the knowledge data-base if considered rule would be created.

Two separate modules with their own GUI's allow users to only actively participate in knowledge formalisation, just to conduct search, or both. In continuance two GUI screenshots are provided: in Figure 8.7 screenshot of interactive rule builder GUI in Preparation module through which productions are specified using TP Entities as building-blocks and in Figure 8.8 a screenshot of synthesized tea-brewing process (example 7.4.1) as shown in Execution's module GUI. In both figures presented objects are not just the static pictures, they are interactive thus responding to the user actions. Consistency of the graphical objects during users interaction is monitored by *Graphical object manager* which are then interpreted and passed to the *Dynamic validatior* (Figure 8.3).

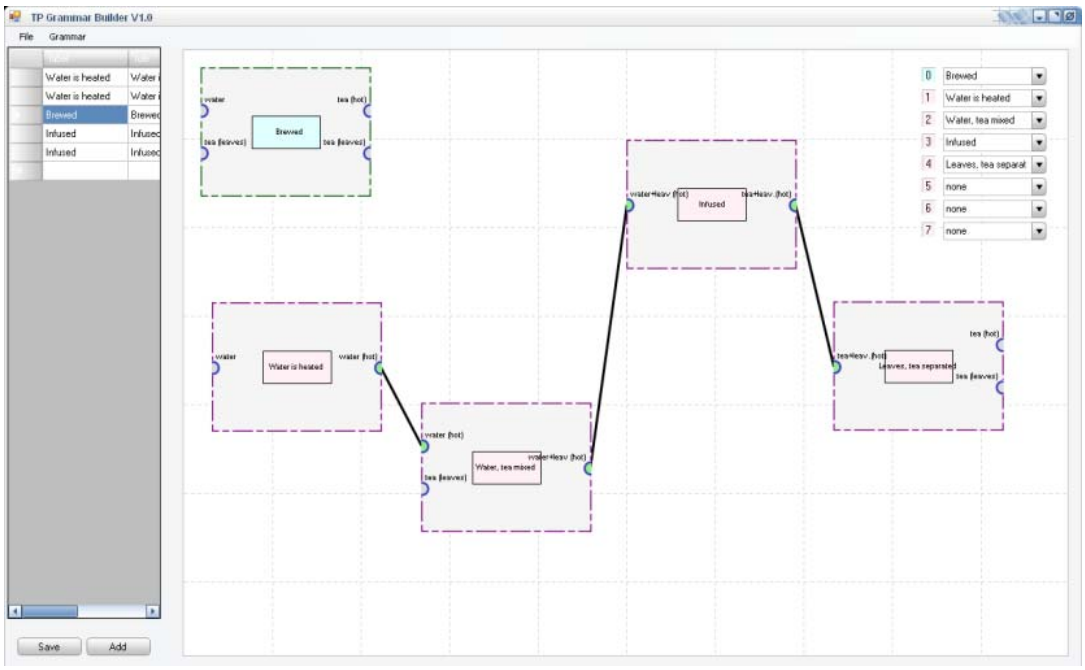


Figure 8.7 Screenshot of interactive rule builder GUI in Preparation module

Finally, when the user defines goals and constraints the generation of optimal variants can start. If no stopping criteria have been set, the user, can always stop the search process as felt fit.

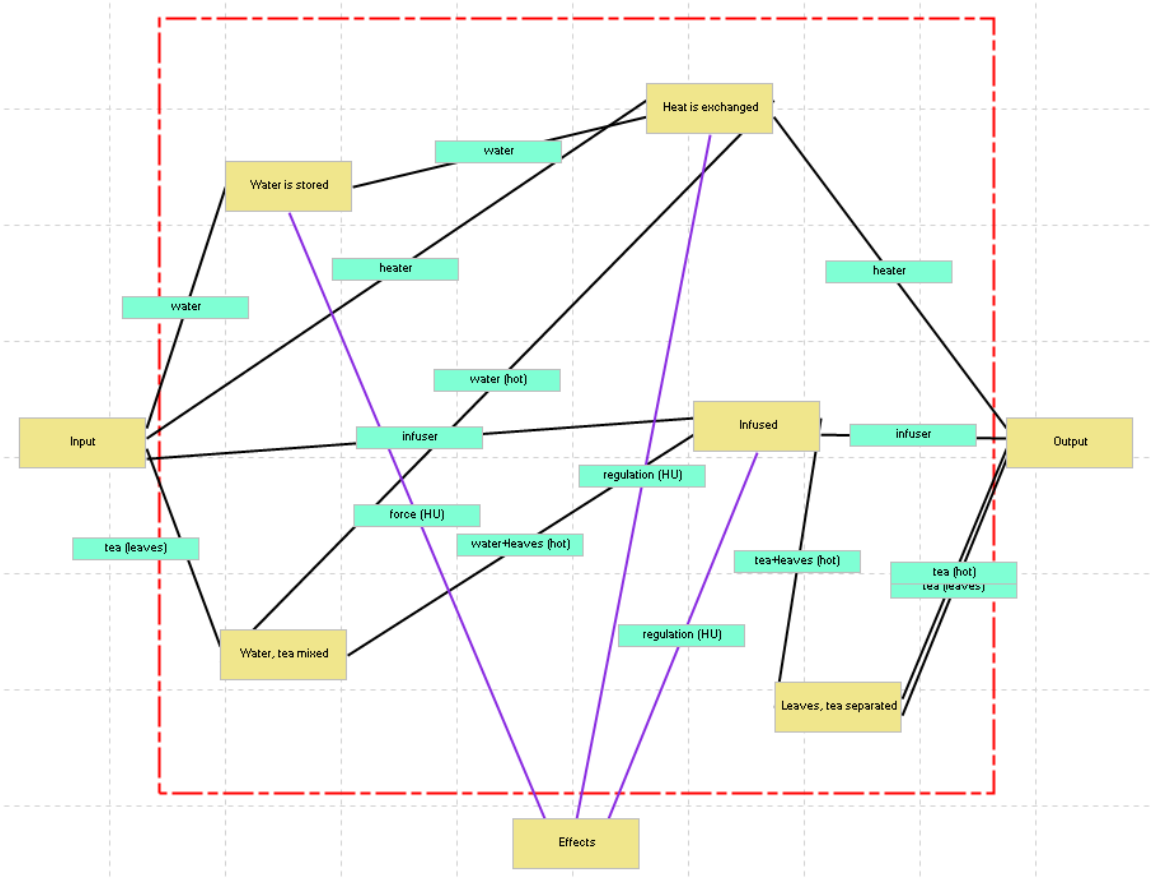


Figure 8.8 Screenshot of synthesized tea-brewing process (example 7.4.1) as shown in Execution's module GUI

8.4 Implications to this thesis

Success of computational tools most often depends on their visual interfaces which allow users better interaction in order to get required support. At the current level of tool's development with no application of advanced algorithms for graph visualizations, the result will be displayed in form as shown by example in Figure 8.8. Algorithms that are able to untangle and depict graph in a manner comprehensible to the user have not yet been implemented within the computational tool. Complexity management methods and tools like dependency structure matrix will also be considered for that purpose. Computational tool as presented in this Chapter is built to a prototype stages in order to be able to see and test whether the proposed modeling can deliver results. Object oriented architecture of presented computational tool permits reuse of its components and dynamic link libraries in order to

further develop computational support for other stages of conceptual design phase. Deciding to model three-tier architecture Figure 8.2 assures multi-user participation allowing access to the BNF rule library to construct rules further, and/or to conduct search based on these stored existing rules.

9. CONCLUSIONS AND FURTHER WORK

The concluding Chapter of this thesis will present a summary on findings that emerged as a result of this research. An outlook will be provided of the aims and objectives as defined within the introductory Chapter of this thesis in respect to what was actually achieved. For the closing future research directions and plans will be laid out.

9.1 Research summary

The aim of this thesis is formulated within its introductory Chapter as to provide a support to begging of the conceptual development stage by offering designers possibility to computationally explore operand transformation variants in technical processes. In a contrast to the conventional research conducted in the filed of Design Science, within computational design synthesis to which body of knowledge this work also belongs to, development of a method for design support always assumes a complementary development of computational tool. Thus, in order to accomplish the postulated aim it was proposed that following objectives have to be met: to devise a method for generation of operand transformation variants based on different technological (working) principles, and to implement that method within computational tool build to a completion stage that allows verification and testing of the research results. According to accepted Design research methodology, development of this research project can be summarized in the following four steps:

1. **Analysis and state of-the-art review**. Analysis consisted of multidisciplinary literature review in the field of engineering design synthesis focusing on the synthesis of technical processes and to an establishment of the state-of-the-art review on current research efforts in the filed of Computational designs synthesis. The literature review on engineering design synthesis provided findings necessary to generally understand the phenomenon of problem solving and cognitive aspects of solution synthesis as a part of problem solving activity. It was tried to be established what kind of logical models of engineering design synthesis exits and especially what is the role of technical process synthesis within design process. The state-of-the-art review on the Computational design synthesis served the purpose to determine theoretical and methodological fundaments of current CDS research efforts, to compare and systematize them in order to focus this research. Based on the findings it was

concluded that contribution to the filed of CDS could be accomplished if a method for technical process synthesis could be devised. To accomplish that goal it was required to propose and develop computational model of technical process as well as a method that could perform technical process synthesis on proposed method.

2. **Determination of theoretical and methodological foundations to this research**. It was necessary to determine the means necessary for devising of method for generation of operand transformation variants. Selection of theoretical fundaments was of course predefined, since only the Theory of Technical Systems and its related siblings like Domain Theory acknowledge the existence of technical processes. Nevertheless, teleological approach that TTS advocates proved to be new for CDS thus even more firmly determining the selected course of the research. Efforts where then turned to exploration of the existent mathematical concepts that could be used for modelling of technical processes and related synthesis methods. Based on the findings from the fields of computation, artificial intelligence and CDS it was concluded to conceive the method as knowledge oriented rather than problem oriented. Knowledge oriented methods achieve knowledge formalisation on extensible set of editable production rules which the method is using rather than encoding the knowledge fixed within the method itself. Based on the latter it was established that knowledge about technical processes, technological (working) principles and necessary effects can be formalised within set of production rules. Since the roots of TTS are drawn from The Systems Theory which opts for graph based system modelling, than it was concluded to model technical process as graph and to base the method for technical processes as production driven transformation system. Optimisation as a frequent engineering demand was also considered as a research question, and in order to enable optimization of technical processes it was found out that existing method of grammatical evolution combines productions and genetic algorithms to perform search proving to be ideal selection for considered rule based formalisation principle and graph based modelling.

3. **Contributing to the fields of Design Science and Computational design synthesis.** Based on all of the findings resulted in development of a graph-grammar based method for synthesis of technical process. For modelling of technical process itself a labelled directed multigraph with operands, objects and effect was created. Thus, the method was conceived as a breadth-first node rewriting based on the knowledge about

technical process that is formalised within set of productions. Grammatical evolution was applied for controlling and directing the goal based search. In addition a set of embedding mechanisms and connecting rules had to be defined in order to perform multigraphs decomposition.

4. **Verification of the research results.** Like in fundamental sciences, it is necessary to by any means verify the research results or at least to provide foundations on top of which verification could be conducted. Principle focus of this research, and one of the expected contributions to the research filed was development of method for decomposition of technical processes. It was hypothesised that if the method would be production rule based, than it is possible to formalise engineering knowledge within set of production rules. Later it was shown that these productions will be graph-grammar based. Thus, for purposes of research results verification a computational tool was conceived and realised on foundations of developed method for generation of operand transformation variants. In Chapter 7 of this thesis two examples were presented in which graph-grammars of tea brewing and stiffener assembly welding were constructed. Using developed method it was shown that technical processes can be synthesized fast and efficient using formalised knowledge within set of productions. Knowledge about TP's was formalised on foundations of online lexicon of English language WordNet [104], The Suggested Upper Merged Ontology (SUMO) as the largest open ontology [107], and recommendations for reconciliation of product function related terms accepted by the NIST [22].

9.2 Results discussion

Problems, issues and prospects related to the creation of computational support for synthesis of technical processes which motivated the author for his research where postulated in introductory Chapter of this thesis (Section 1.3.). Here by the findings on these questions are summarized as follows:

- Based on the findings it was concluded to devise a graph-grammar based method for development generation of operand transformation variants (Figure 6.1). Multitude of methods in CDS today follow the same knowledge–driven approach which offers knowledge driven approach to development of computational design synthesis methods and tools. The increase in performance and range of problem that can be tackled depends on the extents of knowledge that has been formalised, rather than in

change of principle algorithm's mechanisms. By applying graph grammar transformation using set of predefined rules following breadth-first node rewriting principle it is possible to conduct decomposition of technical process starting from black-box goal formulation and ending in synthesized operand transformation process. In order to achieve embedding of each of the rewritings into host graph structure special connection procedures had to be devised. Algorithm of technical process decomposition as well as accompanied connecting procedures is given as pseudo-code in (6.4). Knowledge based graph-grammar methods do not emulate human cognitive processes and reasoning, they do enable application of advanced high-level computational processes like machine induction or grammar inference, what can be considered in future.

- For modelling of technical processes as labelled multi-digraph with operations, operands and effects was created (Definitions 6.2 and 6.3). Multigraph permits bag of relations between each of the nodes what was necessary for technical process modelling (see Table 6.1). Following object-based approach operations, operands and effects are being mapped to the graphs nodes and arcs. A graph as a carrier structure is invariant to complexity of the objects that have been assigned thus creating possibilities for further development. Since the method for decomposition is defined over the same multigraph types, than it is also invariant to type of objects being assigned to graph's nodes and arcs. Operations, operands and effects are process related objects defined as TP entities (Definition 6.1) constitute graph's vocabulary. At the current stage of development only their labels are used by the system.
- Operand transformation variants can be created as starting from black-box representation and than creating all possible rewritings providing graph-grammar and transformation algorithm in (6.4) thus in the end in fact showing to designer whole of the language of considered technical process. To generate only optimal variants within this thesis a grammatical evolution is applied ((5.14), (5.21)-(5.24)). Since at the current stage of development *TP entities* do not permit more attributes which would require technical process knowledge generalisation and systematization, it is not possible to construct useful metrics that could describe universal virtues of technical process as defined by Domain Theory [8] (see Section 2.3). Although multi-objective support exists within *GE LIB*. (Figure 8.6) a constrained goal formulated search is maximum what can be achieved.

- Since the proposed method for operand transformation variants is knowledge-driven it was necessary to explore requirements what need to be met in order to formalise the knowledge about technical processes within set of production rules. It is important to stress out that aims of this thesis did not include research about the content of knowledge about technical processes in respect to its systematization and generalisation. It was intended to provide means to formalise that knowledge within set of production rules and to utilise productions by developed method to generate operand transformation variants. During the research it was found out that knowledge about technical processes still does not exists in accessible open taxonomies or ontologies as per se (Chapter 7). Thus it was necessary to at least suggest guidelines for knowledge formalisation which should be followed when defining production rules. This is an explanation of why to reuse work done to create basis for technical product's function modelling and then to broaden it with process specific operations (as in Figure 7.1 and Figure 7.3) in a form of suggestion or guidelines for production rules definition.

Success of the research that involves deployment of new method and computational tool is most often measured by comparing it to other related scientific work. However, theoretical foundations as well as the level of modelling using multidigraphs or applying GE for optimisation still do not exist within CDS. TTS being rather unknown to CDS community was for the first time introduced as theoretical foundation within this thesis, which altogether makes difficult to evaluate this work in comparison with the others. The intention of this research was to lay foundations for development of complete graph-grammar based framework for support of early design phases. GE as a robust problem solver which can be applicable as a search method for any of the phases in early design assuming that engineering knowledge is formalized using graph grammars. Almost all of design theories are following systemic reasoning thus resulting with early design modelling opting towards transformation systems. When developing computational support it was thus natural to merge graph-grammars with grammar based stochastic search algorithm. It is to be assumed that full impact of these and similar tools to design process and technical products being designed can be evaluated more complete support frameworks will appear in real life engineering environments. It can be concluded that based on the presented findings and achieved research objectives and goals that the hypothesis as postulated in the beginning of this thesis is verified.

9.3 Further work

Expected long-term objectives would be aimed at further development and improvement of computational tool for generation of operand transformation variants in technical processes and possible integration of developed method in the existent frameworks for Computational design synthesis and application in industry, and ultimately creation of an overall graph grammar and grammatical evolution based computational framework for early design support. Moreover, proposed method offers possibilities for induction of new grammar if desired. Both the extension of the proposed model to other stages of product development and definition of type graph and typed graph based decompositions are set as aims of the future research. Future work will also include introduction of taxonomies and ontologies or linking to existing ones in order to introduce types to graph's structure be able to facilitate higher semantic reasoning.

10. REFERENCES

[1] *Hubka, V., Eder, W.E.: "Engineering Design: General Procedural Model of engineering Design", Springer-Verlag Berlin Heidelberg, 1992.*

[2] *Cagan, J., Campbell, M.I., Finger, S., Tomiyama, T.: "A Framework for Computational design synthesis: Model and Applications, Journal of Computing and Information Science, ASME - 2005.*

[3] *O'Neill, M.; Ryan, C.: "Grammatical evolution, IEEE Transactions on Evolutionary Computation", Vol. 5 (4), pp. 349-358, 2001.*

[4] *Pahl, G., Beitz, W.: "Engineering Design – A Systematic Approach", Springer Verlag, 1988.*

[5] *Savransky, S.D.: "Engineering of Creativity - Introduction to TRIZ: Methodology of Inventive Problem Solving", CRC Press, Boca Raton, New York, 2000.*

[6] *Suh, N.P.: "Axiomatic Design, Oxford University Press", New York, 2001.*

[7] *Hubka, V., Andreasen, M.M., Eder, E.W.: "Practical Studies in Systematic Design", Butterworth & Co., London, 1988.*

[8] *Hansen, C.T., Andreasen M.M.: "Two approaches to synthesis based on the Domain Theory", Chakrabarti, A. (Ed.), Springer-Verlag, London, 2002.*

[9] *Lin, Y., Shea, K, Johnson, A., Coultate, J., Pears, J., 2009, A Method and Software Tool for Automated Gearbox Synthesis, ASME, IDETC/CIE 2009.*

[10] *Jensen, T.: "Functional Modelling in a Design Support System – Contribution to Designer's Workbench, Department of Control and Engineering Design", Technical University of Denmark, 1999.*

[11] *Goldberg, D.E.: "Design of Innovation", Kluwer academic Publishers, 2002.*

[12] *Hubka, V., Eder, W.E.: "Theory of Technical Systems and engineering design synthesis", Chakrabarti, A. (Ed.), Springer-Verlag, London, 2002.*

[13] Blessing, L.T.M, Chakrabarti, A.: "DRM, a Design Research Methodology", Springer Verlag London, 2009.

[14] Simon, H.A., "The Sciences of the Artificial, The MIT Press", Third Edition, Cambridge, Massachusetts, London, England, 1996.

[15] Holland, J.H., Holyoak, K.J., Nisbett, R.E. Thagard, P.R.: "Induction – Process of inference, Learning and Discovery", The MIT Press, Cambridge, Massachusetts, London, England, 1986.

[16] Carnap, R.: "An Introduction to the Philosophy of Science", Gardner, M. (ed.), Dover Books, 1995.

[17] Štorga, M.: "Model rječnika za računalnu razmjenu informacija u distribuiranom razvoju proizvoda", Doctoral dissertation, Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, 2005.

[18] Reich, Y.: "Synthesis and theory of knowledge: general design theory as theory of knowledge, and its implication to design", Chakrabarti, A. (Ed.), Springer-Verlag, London 2002.

[19] Chomsky, A. N.: "Syntactic Structures"; Mouton; The Hague; 1957.

[20] Naur, P.: "Revised report on the algorithmic language ALGOL 60", Communication ACM, 6(1), pp. 1-17, 1963.

[21] Stone, R.B., Wood K.L.: "Development of a Functional Basis for Design", Journal of Mechanical Design, Vol. 122, pp. 359-370, 2000.

[22] Hirtz, J., Stone, R.B., McAdams D.A., Szykman, S., Wood, K.L.: "A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts", NIST Technical report, 2002.

[23] Roozenburg N.F.M.: "Defining Synthesis: on the senses and the logic of design synthesis", Chakrabarti, A. (Ed.), Springer-Verlag, London, 2002.

[24] Andreasen M. M., Hein, L.: "Integrated Product Development", IFS Publications, 1987.

[25] Stein, K: "The Genius Engine - Where Memory, Reason, Passion, Violence, and Creativity Intersect in Human Brain", John Wiley & Sons, Inc., Hoboken, New Jersey, 2007.

[26] Weisberg, R.W.: "Creativity: Understanding Innovation in Problem Solving, Science, Invention, and the Arts", John Wiley & Sons, Hoboken, New Jersey, 2006.

[27] Sawyer, K.R.: "Explaining Creativity: The Science of Human Innovation", Oxford University Press Inc., New York, 2006.

[28] Polya, G.: "How To Solve It - A New Aspect of Mathematical Method", Princeton University Press, Princeton, New Jersey, 1973.

[29] Von Bertalanffy, L: "General System Theory - Foundations Development, Applications", George Braziller, New York, 1969.

[30] Takeda, H., Veerkamp, P., Tomiyama, T., Yoshikawa, H.: "Modeling Design Processes", AI Magazine, Vol. 11 (4), 1990.

[31] Pereira, F.C.: "Creativity and Artificial Intelligence: A Conceptual Blending Approach", Walter de Gruyter GmbH & Co., Berlin, New York, 2007.

[32] Capra, F.: "The Web of Life", Anchor Books, A Division of Random House, Inc., New York, 1996.

[33] Sim, S. K., Duffy, A.H.B.: "Towards an ontology of generic engineering design activities", Research in Engineering Design, Vol. 14., pp. 200-223., DOI 10.1007/s00163-003-0037-1, 2003.

[34] Bäck, T., Fogel, D.B., Michalewicz, Z.: „Evolutionary Computation 1, Basic Algorithms and Operators"; Institute of Physics Publishing; Bristol and Philadelphia; 2000.

[35] Goldberg, D.E.: "Genetic Algorithms in Search Optimization and Machine Learning", Addison Wesley Longman Inc., 1989.

[36] Kicinger, R., Arciszewski, T., De Jong, K.A.: "Evolutionary computation and structural design: a survey of the state of the art", Computers & Structures, Vol. 83, pp. 1943-1978, 2005.

[37] Frontiers of Evolutionary Computation, Menon, A. (ed.), Kluwer academic Publishers, 2004.

[38] Shea, K., Starling, A.C.: "From Discrete Structures to Mechanical Systems: A Framework for Creating Performance-Based Parametric Synthesis Tools", American Association of Artificial Intelligence, Technical Report, 2003.

[39] Hutcheson, R.S., Jordan R.L., Stone R.B.: "Application of a Genetic Algorithm To Concept Variant Selection", In Proceedings of the ASME DETC, 2006.

[40] Bryant, C.R., Stone, R.B., McAdams, D.A., Kurtoglu, T., Campbell, M.I.: "Concept Generation from the Functional Basis of Design", In the Proceedings of the ICED05, 2005.

[41] Campbell M., Cagan J., Kotovsky K.: "A-Design: Theory and Implementation of an Adaptive, Agent based Method of Conceptual design", Artificial Intelligence in Design '98, Kluwer Academic Publishers, Netherlands, 1998.

[42] Gips, J., Stiny, G.: "Production systems and grammars: a uniform characterization", Environment and Planning B, Vol. 7, pp 399-408, 1980.

[43] Shea, K.: "Essays of Discrete Structures: Purposeful Design of Grammatical Structures by Directed Stochastic Search", Doctoral dissertation, Carnegie Mellon University, 1997.

[44] Shea, K., Cagan, J.: "Generating Structural Essays from Languages of Discrete Structures", Artificial Intelligence in Design, Kluwer Academic Publishers, pp. 365-384, 1998.

[45] McCormack, J.P., Cagan, J.: "Speaking the Buick language: capturing, understanding, and exploring brand identity with shape grammars", Design Studies Vol. 25 No., Elsevier, Great Britain, 2004.

[46] Pugliese, M.J., Cagan, J.: "Capturing a rebel: modeling the Harley-Davidson brand through a motorcycle shape grammar", Research in Engineering Design 13 (2002) 139-156, DOI 10.1007/800163-002-0013-1, 2002.

[47] Hoisl, F., Shea, K.: "Exploring the Integration of Spatial Grammars and Open-source CAD Systems", ICED'09, Stanford CA, USA, 2009.

[48] Starling, A.C., Shea, K.: "A Parallel Grammar for Simulation Driven Mechanical Design Synthesis", ASME, IDETC/CIE, 2005.

[49] Umeda, Y., Tomiyama, T.: "FBS modelling: modelling scheme of function for conceptual design", Proc. Working Papers of the 9th Int. Workshop on Qualitative Reasoning About Physical Systems, pp. 271-278, 1995.

[50] Erden, M. S., Komoto, H., van Beek, T.J., D'Amelio, V., Echavarria, E., Tomiyama, T.: "A review of function modeling: Approaches and applications", Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Vol. 22, pp. 147-169, 2008.

[51] Bolognini, F., Seshia, A.A., Shea, K.: "Exploring the Application of a Multi-domain Simulation-based Computational Synthesis method in MEMS Designs", ICED'07, Paris, France, 2007.

[52] Wu, Z., Campbell, M.I., Fernandez, B.R.: "Bond Graph Based Automated Modelling for Computer-Aided Design of Dynamic Systems", Journal of Mechanical Design, Vol. 130, ASME, 2008.

[53] Kurtoglu, T., Campbell, M.I.: "A graph Grammar Based Framework for Automated Concept Generation", In the Proceedings of the International Design Conference DESIGN 2006, Dubrovnik, Croatia, 2006.

[54] Jin, Y., Li, W.: "Design Concept Generation: A Hierarchical Coevolutionary Approach", Journal of Mechanical Design, Vol. 129, pp. 1012-1022, 2007.

[55] Schmidt, L.C., Cagan, J.: "GGREADA: A Graph Grammar-Based Machine Design Algorithm", Research in Engineering Design, Vol. 9, pp. 195-213, 1997.

[56] Siddique, Z., Rosen, D.W.: "Product Platform Design: A Graph Grammar Approach", DETC99/DTM-8762, ASME, 1999.

[57] Schmidt, L.C., Shetty, H., Chase, S.C.: "A Graph Grammar Approach for Structure Synthesis of Mechanisms", ASME; Journal of Mechanical Design, Vol. 122, pp. 371-376, 2000.

[58] Helms, B., Shea K.: "Object-Oriented Concepts For Computational design synthesis ", In the Proceedings of of DESIGN 2010, Dubrovnik, Croatia, 2010.

[59] Geiß, R., Batz, G., Grund, D., Hack, S., Szalkowski, A.: "GrGen: A Fast SPO-Based Graph Rewriting Tool, Graph Transformations, Springer, Berlin, 2006.

[60] Wyatt , D.F., Wynn, D.C., Clarkson, P.J.: "A Computational Method To Support Product Architecture Design", Proceedings of ASME-IMECE, 2009.

[61] Rihtaršič, J., Žavbi, R., Duhovnik, J.: "Sophy – Tool for Structural Synthesis of Conceptual Technical Systems", In the Proceedings of of DESIGN 2010, Dubrovnik, Croatia, 2010.

[62] Mitchell, T.M.: "Machine Learning", McGraw-Hill Science, 1997.

[63] Stanković, T., Shea, K., Štorga, M., Marjanović, D.: "Grammatical Evolution of Technical Processes", ASME-DETC, San Diego, CA, USA, 2009.

[64] Stanković, T., Bojčetić, N., Marjanović, D.: "Developing Computational Tool for Generation of Operand Transformation Variants In Technical Processes", Proceedings of the DESIGN 2010, Marjanović, Dorian (ur.). Zagreb, Glasgow, The Design Society; FSB, 2010, 1409-1420, 2010.

[65] Turing, A.M.: "Computing machinery and intelligence", Mind, 59, pp. 433-460, 1950.

[66] Jiang, T., Li, M., Ravikumar, B., Regan, K.W.: "Formal Grammars and Languages, Algorithms and Theory of Computation Handbook"; edited by Atallah M. J., CRC Press, Boca Raton, New York; 1999.

[67] Goldstein, I., Papert, S.: "Artificial Intelligence, Language, and the Study of Knowledge", Cognitive Science, Volume 1, Issue 1, pages 84–123, DOI: 10.1207/s15516709cog0101_5, January, 1977.

[68] Gips, J., Stiny, G.: "Production systems and grammars: a uniform characterization", Environment and planning B, Vol.7, pp. 399-408, 1980.

[69] MIT Encyclopaedia of the Cognitive Sciences, Edited by Wilson, R. A. and Keli, F. C., A Bradford Book, The MIT Press, 1999.

[70] Solomonoff, R.J.: "An inductive inference machine", IRE National Convention Record, pt. 2, , pp. 56-62, 1957.

[71] Minsky M.L.: "Steps toward artificial intelligence" Proceedings of the Institute of Radio Engineers, 1961.

[72] Levet, W.J.M.: "An Introduction to the Theory of Formal Languages and Automata", John Benjamins Publishing Company, Amsterdam, Philadelphia, 2008.

[73] Minsky, M.L.: "Computation: Finite and Infinite Machines", Prentice Hall, Series in Automated Computations, 1967.

[74] Hopcroft, J.E., Motwani, R., Ullman, J.D.: "Introduction to Automata Theory, Languages, and Cpomputatio", Addison-Wesley Publishing Company, 2001.

[75] Slonneger, K. Kurtz, B.L.: "Formal Syntax and Semantics of Programming Languages – A Laboratory Based Approach", Addison-Wesley Publishing Company, 1995.

[76] Duke University Medical Center (2008, August 14), How DNA Repairs Can Reshape Genome, Spawn New Species. ScienceDaily, from http://www.sciencedaily.com-/releases/2008/08/080813144407.htm, retrieved September 13th, 2010,

[77] Stanković, T., Žiha, K., Pavković, N.: "Back-tracing Technical Progress with Evolutionary Algorithm", Proceedings of the 29th International Conference on Information Technology Interfaces, ITI, 2007.

[78] O'Neill, M.; Ryan, C. "Grammatical Evolution by Grammatical Evolution: The Evolution of Grammar and Genetic Code", Lecture Notes in Computer Science, Springer Berlin/Heidelberg, pp. 138-149, 2004.

[79] O'Neill M., Brabazon A.: "Recent Adventures in Grammatical Evolution", Computer Methods and Systems CMS'05, Krakow, Poland, pp. 245-253, 2005.

[80] Tsoulos I., G., Lagaris I.E., "GenMin: An enhanced genetic algorithm for global optimization", Computer Physics Communications, DOI:10.1016/j.cpc.2008.01.040, 2008.

[81] Bäck, T.: "Evolutionary Algorithms in Theory and Practice"; Oxford University Press; New York; 1996.

[82] Merkle, L.D., Lamont, G.B.: „A Random Function Based Framework for Evolutionary Algorithms"; Proceedings of the Seventh International Conference on Genetic Algorithms, San Mateo, CA; Morgan Kaufmann; 1997.

[83] Coello, C.A.C, Lamont G.B., Van Veldhuizen, D.A.: „Evolutionary Algorithms for Solving Mult-Objective Problems"; 2nd Edition; Springer Science+Business Media, LLC; 2007.

[84] Bentley, P.: "Evolutionary Design by Computers"; Morgan Kaufmann; 1999.

[85] Geertzen, J.: "String Aligenment in Grammatical Inference, Induction of Linguistic Knowledge Research Group"; Technical Report ILK-0311; Tilburg University, 2003.

[86] Prusinkiewicz, P., Lindenmayer, A.: "The Algorithmic Beauty of Plants"; Springer-Verlag, New York Ltd.; 1996.

[87] O'Neill, M., Ryan, C., Keijzer, M., Cattolico, M.: "Crossover in Grammatical Evolution, Genetic Programming and Evolvable Machines", Kluwer Academic Publishers, Vol. 4, pp. 67-93., 2003.

[88] Bäck T., Fogel D. B., Michalewicz Z.: "Evolutionary Computation, Advanced Algorithms and Operators", Institute of Physics Publishing, Bristol and Philadelphia, 2000.

[89] Yockey, H.: "Information theory, Evolution and the Origin of Life", Cambridge University Press, 2005.

[90] Darwin, C.: "The Origin of Spicies", 6th Edition, Online Literature Library, 2004.

[91] Weisstein, E.W.: "The CRC Encyclopedia of Mathematics", CRC Press, 2009.

[92] Wolfram, S.: "A New Kind of Science", Wolfram Media, 2001.

[93] Kurtoglu, T., Campbell, M.I., Bryant, C.R., Stone, R. B. McAdams, D.A.: "Deriving a Component Basis for Computational Functional synthesis", ICED 05, Melbourne, Australia, 2005.

[94] Kreowski, H., Klempien-Hinrichs, R., Kuske S.: "Some Essentials of Graph Transformations, Recent Advances in Formal Languages and Applications", Esik, Z. et. al., Springer, 2006.

[95] *Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: "Fundamentals of Algebraic Graph Transformation", Monographs in Theoretical Computer Science – An EATCS Series, eds. Brauer, W.et. al., Springer, Berline, Heidelberg, New York, 2005.*

[96] *Rosen, K.H., Michaels, J.G., Gross, J.L., Grossman J.W., Shier D.R.: "Handbook of discrete and combinatorial mathematics", CRC Press LLC., 2000.*

[97] *Berge, C.: "Hypergraphs - Combinatorics of a finite sets", North-Holland, Elsevier Science Publishers, 1989.*

[98] *Kniemeyer, O.: "Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling", Dissertation, Fakult¨at f¨ur Mathematik, Naturwissenschaften und Informatik, der Brandenburgischen Technischen Universit¨at Cottbus, 2008.*

[99] *König, B.: "Analysis and Verification of Systems, with Dynamically Evolving Structure", Habilitationsschrift zur Erlangung der Venia Legendi in Informatik, Institut für Formale Methoden der Informatik, Universität Stuttgart, 2004*

[100] *Kitamura, Y., Mizoguchi, R.: "Ontology-based systematization of functional knowledge", Journal of Engineering Design, Vol. 15, No.4, pp. 327-351, August 2004.*

[101] *Russell, S.J., Norving, P.: "Artificial Intelligence- A Modern Approach", Prentice Hall, Englewood Cliffs, New Jersey, 1995.*

[102] *Ahmed, S., Štorga, M.: "Merged ontology for engineering design: Contrasting empirical and theoretical approaches to develop engineering ontologies", AI EDAM, Volume 23, Issue 04, November 2009, pp. 391-407, 209*

[103] *Gruber, T.: "A translation approach to portable ontology specifications", Knowledge Acquisition 5(2), 199–220, 1993.*

[104] *Štorga, M., Andreasen, M.M., Marjanović, D.: "The design ontology—contribution to the design knowledge exchange and management", Journal of Engineering Design, Journal of Engineering Design, 1466-1837, Volume 21, Issue 4, Pages 427 – 454, 2010.*

[105] *Princeton University "About WordNet." WordNet. Princeton University. 2010. http://wordnet.princeton.edu.*

[106] Niles, I., and Pease, A.: "Towards a Standard Upper Ontology", In Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001), Chris Welty and Barry Smith, eds., Ogunquit, Maine, October 17-19, 2001.

[107] SUMO - Suggested Upper Merged Ontology, http://www.ontologyportal.org., 2010.

[108] Simon, H.: "The Shape of Automation: For Men and Management". New York: Harper & Row, 1965.

[109] Microsoft Developer Network, MSDN Online Library, Microsoft Corporation, http://msdn.microsoft.com/en-us/library/ee658109.aspx,, 2010.

[110] Anderson, J.: "Multiobjective Optimization in Engineering Design, Applications to Fluid Power Systems", Ph.D. Thesis, Linköpings universitet, Sweden, 2001.

[111] Deb, K. Multi-Objective Optimization using Evolutionary Algorithms, John Wiley & Sons Ltd, England; 2001.

CURICUULUM VITAE

Tino Stanković was born in 1975. in Kassel, Germany. After returning to Croatia he completed his primary and secondary education (High school for mathematics and computing). In 1993 he enrolled in the study of naval architecture at the Faculty of Mechanical Engineering and Naval Architecture University of Zagreb where he graduated in 2002. From 2004 to 2005 he was employed by Factory for rolling stock "TŽV Gredelj" where he worked as a production engineer. Since 2005 he is employed at the Faculty of Mechanical Engineering and Naval Architecture University of Zagreb as a Ph.D. research student at Chair of Design and Product Development. His research is founded by Ministry of Science, Education and Sports of Republic of Croatia as a part of the research project 120-1201829-1828 "Model and Methods of Knowledge Management in Product Development".

In summer of 2005 he attended an international two-week PhD seminar: “Ph.D. Course - Design Methodology” organised by the Danish Technical University, Technical University of Berlin and University of Saarland. Since 2006 he took an active part in the the European global product development course (E-GPR) organised in cooperation with Technical University of Delft and other international partners. In academic year 2008/2009, on account of grant funded by Ministry of Science, Education and Sport of the Republic of Croatia, Tino Stanković spent six months as a visiting researcher at the Technical University of Muenchen.

He is the author or co-author of several scientific and professional papers published in Croatia and abroad.

ŽIVOTOPIS

Tino Stanković rođen je 1975. godine u Kasselu u Njemačkoj. Po povratku u domovinu, te nakon završene osnovne i srednje škole (XV. gimnazija u Zagrebu), 1993. godine upisuje Fakultet strojarstva i brodogradnje u Zagrebu. Diplomirao je 2002. godine na usmjerenju "Brodogradnja" sa temom "Interaktivno modeliranje dijelova brodske konstrukcije parametriziranjem geometrije osnovnih sklopova". U periodu od 2004. do 2005. zaposlen u TŽV „Gredelj" gdje je radio kao tehnolog u procesu proizvodnje. Od 2005. godine zaposlen je pri Katedri za konstruiranje i razvoj proizvoda Fakulteta strojarstva i brodogradnje u Zagrebu kao znanstveni novak na projektu Ministarstva znanosti i tehnologije Republike Hrvatske broj 120-1201829-1828 "Modeli i metode upravljanja znanjem u razvoju proizvoda".

Tijekom ljeta 2006. godine završio je međunarodni doktorandski seminar "Ph.D. Course - Design Methodology" u organizaciji Danskog tehničkog sveučilišta, Tehničkog sveučilišta u Berlinu, te Tehničkog sveučilišta Saarland. U periodu od 2006. do 2009. godine aktivno sudjeluje u izvođenju E-GPR programa međunarodne nastave koji se održavao u suradnji s Tehničkim sveučilištem u Delftu i drugim sveučilištima iz inozemstva. U srpnju 2008. godine, Ministarstvo znanosti i tehnologije Republike Hrvatske, dodijelilo mu je stipendiju za boravak na Tehničkom sveučilištu u Minhenu gdje je u trajanju od šest mjeseci tijekom akademske godine 2008./09. proveo dio istraživanja u okviru izrade disertacije.

Od 2006. aktivno sudjeluje i u organizaciji međunarodnih znanstvenih skupova iz serije DESIGN, koji se s dvogodišnjim ciklusom održavaju u Dubrovniku. Od 2008 djeluje kao recenzent TMCE međunarodne konferencije. 2010. godine postaje član međunarodnog znanstvenog odbora ICED međunarodne konferencije

Kao autor ili koautor objavio je više znanstvenih i stručnih radova u Hrvatskoj i inozemstvu.