A Framework For Optimization Problem (re)formulation Using A Cognitive Computing Concept

A FRAMEWORK FOR OPTIMIZATION PROBLEM (RE)FORMULATION USING A COGNITIVE COMPUTING CONCEPT

by

Choat Inthawongse

Presented in the Graduate and Research Committee
of Lehigh University
in Candidacy for the Degree of
Doctor of Philosophy

in
Industrial and Systems Engineering

Lehigh University

May 2019

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____

Date

_____

Dr. George R. Wilson
Dissertation Advisor

_____

Accepted Date

Committee:

_____

Dr. George R. Wilson, Chairman

_____

Dr. Héctor Muñoz-Avila

_____

Dr. Derya Pamukcu

_____

Dr. Eugene Perevalov

For my Grand Mother, Khunya Nam Intawongse

Acknowledgments

First and foremost, I am profoundly grateful to my advisor Dr. George R. Wilson, for mentoring me throughout my unusually lengthy Ph.D. study. Without his profound belief in my work, continuous support, confidence, ingenious ideas expertise, and patience, none of this could have been possible. I would like to thank my Ph.D. thesis committee Dr. Derya Pamukcu, Dr. Eugene Perevalov, and Dr. Héctor Muñoz-Avila for being on my committee, providing constructive criticism, and invaluable insight into my work.

At Lehigh, I'm greatly in debt to many people. From the P.C. Rossin College of Engineering, Brianne Lisk, I very much appreciate for unparalleled support. In the ISE department, I owe so much to all ISE faculty members for extensive knowledge and advice. Special thanks to Dr. Tamás Terlaky and Dr. Luis Nunes Vicente, for granting me several years of the teaching assistant funding. I'm also grateful to ISE staff, Ana Quiroz, Rita Frey, Kathy Rambo, Edwin Force, and Abby Barlock for being helpful and supporting when need. The Library and Technology Services, particularly the help-desk members that gave me invaluable help from the started. I also thank my very best friends Dr. Pornsak Thasanarapan, Suresh Bolusani, Dr. Elçin Çetinkaya, Kámil Çifti, Dr. Pacharmon & Dr. David Fuhy, Dr. Hiva Ghanbari, Dr. Yuhai Hu, Dr. Dan Li, Dr. Matt Menickelly, Dr. Chatree Saiyasombat, Dr. Mohammadreza Samadi, Dr. Wilawan Thongda, Dr. Benjamaporn Wonganu, and Dr. Wei Xia for their pure moral support whenever I needed.

Heartfelt thanks go to the Thai Government for the initial 5 years funding of my doctoral study. All the staff at Office of Educational Affairs, Royal Thai Embassy D.C., especially Ms. Somthavil Thongprasert, I thank you very much for the support, collaboration, and promptly replying to my endless emails throughout the years.

Lastly, I owe so much thanks to my beloved family, Dr. Khosit, my uncle and god father, Dr. Kosol, my father, Mrs. Sunun, my mother, Dr. Rungkarn, my sister for unconditional loving, understanding and helping me on concurring this journey. I would also like to extend my deepest gratitude to my teacher and US.mom, Dr. Jane Desnouée, especially for her kindness and unwavering guidance.

Contents

List of Tables

List of Figures

LIST OF FIGURES

Abstract

An Operations Research (OR) analyst is usually presented with a plethora of modeling choices including formats to represent optimization problems, various optimization components, and various solvers. Mathematical models are a medium for implementing optimization, by gathering the appropriate information in a suitable form. Model building is composed of interrelated steps performed recursively. As the solving phase became more advanced with improved algorithmic theory and computing technology, the formulation phase that involves an analyst became a hindrance to the efficiency of the overall optimization process. Certain problem characteristics have a much more significant impact on the ease of optimization than others. The form of the problem constraints can have an enormous effect on the ease of solution. For these reasons, it is sometimes advisable to (re)formulate a model into an appropriate form. This research outlines a novel optimization problem formulation and reformulation process using a cognitive computing approach. The cognitive system interacts with an OR analyst in plain, but restricted natural language, acting as a modeler's trusted advisor to ensure that all facets of modeling are considered. The framework proposes a form of optimization model workbench based on Ontology where an OR modeler may query the system for advice, and where a new experience becomes part of the existing training set. The aim is to provide a bridge between the practitioner and model elements to enable an OR analyst to design models more effectively than would otherwise be possible. Underlying the framework is an extension of Geoffrion's Structured Modeling concepts that facilitate the model's semantics. The framework establishes a decomposition

of an optimization model into smaller elements, each communicating on its own with a necessary optimization tool. The framework draws on a synergy between machine learning, information theory, ontology engineering, and operations research. The method developed is general and may be applied to any other problem domain, if a sufficiently accurate characterization of the domain can be manipulated within the framework.

Chapter 1

Introduction

> "To solve a problem, whether simple or complex, is to make the best choice from among the available courses of action." (Ackoff, 1962)

1.1 Overview

Mathematical models have a fundamental role in Operations Research (OR). Models provide distilled description and explanations of the systems that they represent. Mathematical models are very useful, but it is important to recognize that the value of models and modeling approaches extend way beyond the realm of mathematics for decision and control in OR applications. To find the solution for mathematical programming models, if we pass from a proposed condition to a new condition equivalent to it, we still have the same solutions. But if we pass from a proposed condition to a narrower one, we lose solutions. And if we pass to a broader one we admit incorrect solutions that have nothing to do with the proposed problem.

OR analysts and practitioners spend a good amount of time developing and using models as a core instrument for their analysis. Models are not only instruments for representing the best (or at least good) course of action from a set of actions, but also an effective tool to explore the structure of a problem and to uncover choices that were overlooked. One of the most important tasks for an OR researcher is to diagnose a problem and determine a strategy to be used for information acquisition: What questions should be asked, what tests should be run, what data can safely be ignored, and what algorithms are absolutely necessary to obtain desired solutions? Once the right questions have been asked and the right data obtained, solving such problems becomes obvious to even the least experienced analyst. What distinguishes an expert is an ability to sense important exclusions in the data that can often be fulfilled by simply asking the right questions, noting that the value of such information may not always be correlated with the cost of obtaining it.

The real challenge in dealing with a complex, real-world problem is to have a proper mind-set and to identify and extract the critical problem elements. Different players in the OR discipline may view optimization differently. For instance, considering a mathematical programming element, a modeler may see an abstract representation to be analyzed and understood. Secondly, for application developers, a mathematical program is a concrete instance to be represented, communicated, and solved. Lastly, users may not realize they are using an optimization system but, rather, are often focused on the outcome (goals), such as maximizing profit or minimizing costs. An OR analyst is usually faced with a plethora of decision combinations including formats to represent optimization problems, diverse optimization components, and various solvers that could be employed. In addition, these different optimization components are usually located on different platforms which use different programming and modeling languages. The number of variables and parameters in

a problem often exceed human simultaneous manipulation capabilities, with the number of dimensions exceeding human visualization capacities, as well. The problem definition and its optimization model each change dynamically as the OR analyst's understanding of the problem evolves. This discovery of alternative courses of action may often be the most important use to which the model can be put.

Among others, Nadler [82] addressed the serious deficiencies of the most problematic classification schemes (also appear in Evans [35], page 89).

(1) how one decides where a particular problem fits within a classification scheme;

(2) classification schemes are descriptive at best; a few classification schemes point to a specific problem-solving approach; and

(3) some classification schemes lead to analytical techniques that limit the solution space and overlook critical thinking.

As a practical matter, many problems cannot be pigeonholed into a fixed classification scheme. In fact, the classical classification will not work with them. Thus, this biased perception problem causes an inappropriate problem formulation and, ultimately, a wrong solution to the intended problem. However, a model formulation is still a mystery relegated to the not yet well-understood process of creativity. This research sets forth a new paradigm in optimization model formulation, utilizing cognitive computing. The cognitive system interacts with an OR analyst in plain, but restricted natural language, acting as a modeler's trusted adviser to ensure that all facets of modeling are reckoned with.

This research proposes a form of optimization model workbench where an OR modeler queries the system for advice and every new experience becomes part of the existing train-

ing set. The aim is to provide a bridge between the practitioner and model elements to enable an OR analyst to design models more effectively than would otherwise be possible. This research studies a framework that will assist an OR analyst to recognize multiple model formulations, instead of one formulation. The fundamental notion of inferences and representation from problem context addresses how much model fidelity and model sensitivity can be expected in a model recommendation by this framework to an OR analyst.

1.2 Operations Research and Decision Analysis

In a broad sense, OR is a branch of applied science for decision making, which employs advanced, analytical methods to help make better decisions when solving complex problems. Ulrich [102] defines an integrated perspective of OR that is linked to critical systems thinking. He pointed out the criticism that was made by Ackoff [3]:

> This obsession with techniques . . . reduced the usefulness of OR, a reduction that was well recognized by executives who pushed it further and further down in their organizations, to where such relatively simple problems arose as permitted the application of OR's mathematically sophisticated but contextually naïve techniques.

Greenberg gives a thoughtful discussion in *A Bibliography for the Development of An Intelligent Mathematical Programming System* [55], Why is a model built? The brief answer is to support decision-making, raising the need for supporting analysis and model management. In addition, the bibliography collection indicates that there have been fewer papers

focused on environments for analysis than for modeling. An OR analyst must balance between abstraction and fidelity, by interacting with domain users to build up a more precise definition of the problem, which is an iterative process that will ultimately form a model. Model building is generally composed of several interrelated steps that transform and combine the modeler's knowledge and experience into action involving exhaustive data analysis by selecting and constructing the most important information from the data. There are eight elements in an optimization system: (1) model, (2) modeling language environment (MLE), (3) instance representation (data representation), (4) communication agent (interface), (5) server/registry, (6) analyzer, (7) solver, and (8) simulation. Moreover, a triangular communication exists among interface, register, and solver. Solving problems with operations research technique requires six phases: (i) formulating the problems, (ii) constructing a mathematical model to represent the system under study; (iii) deriving a solution from the model; (iv) testing the model and the solution, which is derived from the model; (v) establishing controls over the solution; and (vi) implementing the solution.

In many instances, real-world problems (so-called messy problems [1, 3, 4], wicked problems [17], or complex problems) originate from unpredictable, uncontrollable, and sporadic events. Rather most mathematical modeling methodology requires a specific set of assumptions, which may not mesh with reality. Hence, the result from those unrealistic and naïve models may possibly be useless as they cannot be authenticated, leading to criticism in the OR community in 1970 [78]. As a result, OR researchers in Great Britain developed context-oriented techniques, referred to as "soft OR," tackling messy problems, while preserving the mathematically oriented approach as "hard OR".

1.2.1 Hard OR

"Hard OR" has been widely employed in the United States for many decades, becoming the primary paradigm for operations research in general [61, 78, 86]. It can be referred to as *functionalist* whereby the quantitative modeling becomes the focus of attention. In other words, an OR analyst is devoted to the models and their solutions rather than solving real-world problems. In fact, a clear and unarguable quantitative objective exists to be optimized for well-defined problems. The structure of such problems, despite being technically challenging, is well-known and renders itself to quantitative modeling; so solutions are clear-cut and easily implemented. For the sake of discussion, the general quantitative model will be specialized to optimization models. Specifically, the traditional methods of optimization fall into *two* distinct categories:

(i) algorithms that only evaluate complete solutions; and

(ii) algorithms that require the evaluation of partially constructed or approximate solutions.

For algorithms that only evaluate a complete solution, all of the decision variables are specified. When we find that a new solution has a better evaluation than the previous one, it replaces the prior solution. Such methods include exhaustive search (enumeration), local search, hill climbing, and gradient-based numerical optimization methods, as well as heuristic methods such as simulated annealing, tabu search, and evolutionary algorithms. Chief examples of such algorithms are:

*Linear Programming (LP)***:** Maximize an evaluation function, which is a linear combination of variables, that is constrained by linear equality and inequality constraints.

*Gradient-based***:** Minimize an evaluation function if it is smooth (differentiable) and, unimodal.

*Combinatorial problem***:** There are many different local operators that can be employed to take advantage of the characteristics that are inherent to developing an optimal solution to a circuit in the Euclidean plane; e.g., Traveling Salesman Problem (TSP).

Considering the latter category, for algorithms that require an evaluation of partially constructed or approximate solutions or incomplete solutions, an incomplete solution to the problem is originally posted. The hope is that in solving each of these easier problems, we can eventually combine the partial solutions to get an answer for the original problem, or a complete solution to a reduced problem.

Modeling typically means the construction of models which can be used for detection of insights or for a presentation of perceptions of systems. Model building is generally composed of several interrelated steps that transform and combine the modeler's knowledge and experience into action involving exhaustive data analysis by selecting and constructing the most important information from the data. The very act of modeling consists of a selection and construction, workmanship, an analogy conclusion or other derivations on the model and its relationship to the real world, and preparation of the model for its use in systems (accounting for future evolution and change). Wilson [107] presented a view of models and the distinction between models and modeling:

> Models (of any kind) are *not* descriptions of the real world; they are descriptions of *ways of thinking* about the real world.

Models may be inherently incomplete, biased, and ruled by their creator, by bounding attention to parts of the applications that are under examination while ruling out any other

parts that have never been under consideration. This idea motivated us to find ways that can help others become better modelers, design a good modeling process, and explore what makes a good model in general. In fact, finding the right solver, selecting the computing resources, invoking the software, and interpreting the solution through a systematic process is this research's ultimate goal.

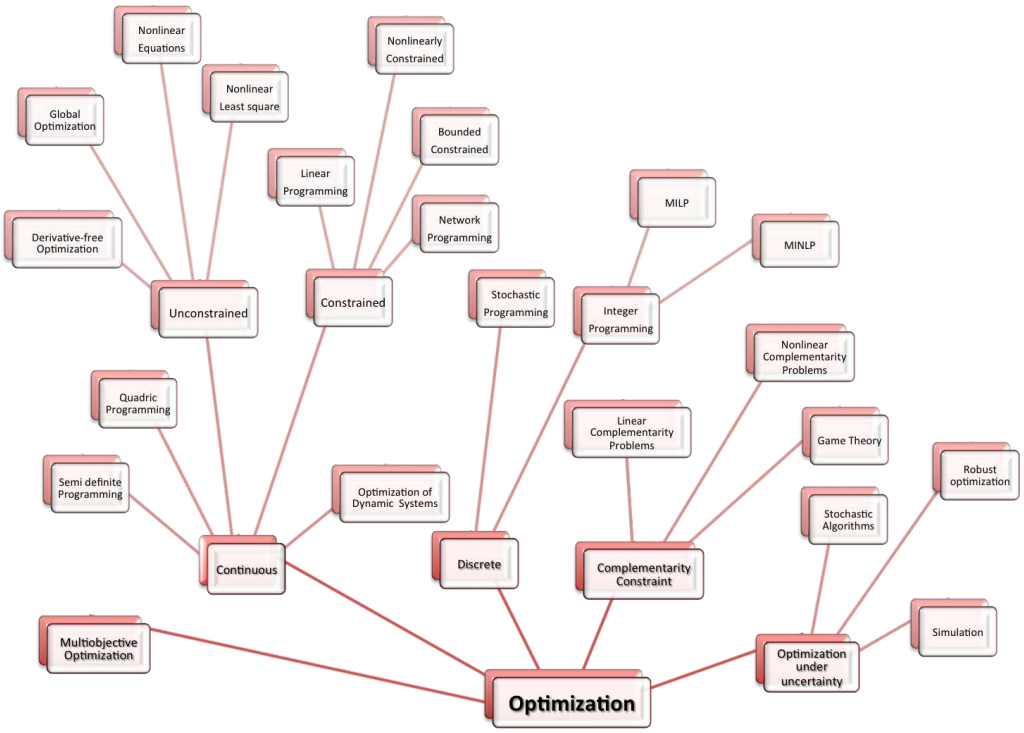

Figure 1.2.1: Taxonomy of Optimization Tree (http://neos-guide.org)

It is a daunting task to categorize optimization [76]. There are many possible perspectives for the optimization taxonomy. For example, one may start by considering the type of variables in the problem, yielding two optimization types: continuous and discrete; while another may consider the type of functions, yielding another two optimization types: linear, and nonlinear. Mathematicians may view optimization by their constraint types, yielding

three optimization types: generally constrained, bound constrained, and unconstrained; while the statistician may look at mainly three optimization types: (i) deterministic, (ii) non-deterministic (uncertain), and (iii) multi-objective optimization. Since algorithms for solving optimization problems are tailored to a particular type of problem, it is important to correctly classify optimization models. A neos-guide provides guidelines for Optimization Taxonomy as shown in the optimization tree (Figure 1.2.1). However, there exist some optimization problems that is a cross between these initial categorizations, Combinatorial programming can be a combination of linear and nonlinear functions; mixed integer linear programming is the combination of a continuous and an integer (discrete) variables.

Problem Class	**Description**	**Formulation**
1. Linear Programming (LP)	Linear objective, linear constrains:	$\min_x c^T x$
	Linear Optimization Problem (convex)	s.t. $Ax = b$, $x \geq 0$
2. Quadratic Programming (QP)	Quadratic objective and linear constrains:	$\min_x c^T x + \frac{1}{2} x^T Q x$
	Quadratic Optimization Problem (convex, if Q pos. def.)	s.t. $Ax = b$, $Cx \geq d$
3. Nonlinear Programming (NLP)	Nonlinear Optimization Problem	$\min_x f(x)$
	(in general non-convex)	s.t. $h(x) = 0$, $g(x) \geq 0$
4. Integer Programming (IP)	Some or all variables are integer (e.g. linear integer pb.)	$\min_x c^T x$
	Special case: combinatorial optimization problems	s.t. $Ax = b$, $x \in Z_+^n$
5. Non smooth optimization	objective function or constrains are	$\min_x f(x)$
	non differentiable or not continuous e.g.	s.t. $x \in G$.
6. Optimal Control	Optimization problems including dynamics in form of	$\min_{,x,u,p} \int_0^t \emptyset(t, x(t), u(t), p) dt$
	differential equations (infinite dimensional)	s.t. $\dot{x} = f(t, x(t), u(t), p)$, ...

Table 1.2.1: Classes of Optimization Problems

Table 1.2.1 summarizes the generic optimization classes, starting from deterministic constrained optimization with continuous and linearly constrained: LP and QP. A drawback exists when applying the pure hard OR paradigm to complex problem; Optimization yields the best solution to the modeled problem [6]. But, if we do not know much about our prob-

lems, or if it does not fit well within the class of problems that each of these algorithms can tackle, we are left with enumerating the possible solutions. This approach is almost always impractical because the number of alternative solutions can be prohibitively large. Optimization-type problem formulation has always been considered as purely a cognition-based human task, where reformulation of the optimization model is one such task [96]. OR analysts typically becomes better at formulating, solving, and making decision as their expertise in a model formulation domain increases with experience.

1.2.2 Model representation

According to Robert Fourer [37], the designer of the mathematical programming language (AMPL)[1]

> ". . . the optimization development cycle was found to take much more analyst time than expected. The culprit was the awkward and error-prone work of converting an optimization problem between the modeler's conception and the algorithm's representation. Indeed the natural way for a modeler to think about and express models is in direct conflict with the input requirements of solution algorithm's representation."

Various schemes for solving optimization problems have emerged over the evolution of the computer. Advancing from matrix generators, modeling languages are an abstract way to describe models and their data. There is a great variety of modeling languages, e.g., algebraic, behavioral, discipline-specific, domain-specific, framework-specific, object oriented, among others [37, 55, 66]. This section focuses on algebraic modeling as it is used most frequently in model representation of optimization problems.

[1]Together with David M. Gay and Brian Kernignhan, Robert Fourer was awarded 1993 ORSA/CSTS Prize by the Computer Science Technical Section of the Operations Research Society of America, for the design of AMPL modeling language.

Most optimization-oriented modeling languages are declarative languages. They define the problem, store the knowledge about a model, and usually do not specify how to solve the problem. Optimization problems are usually defined by algebraic modeling languages[2]. Declarative languages have three distinct characteristics [66]:

(i) Problems are represented in a declarative way.

(ii) There is a clear separation between problem definition and the solution process.

(iii) There is a clear separation between the problem structure and its data.

Essentially, a modeling language can be summarized by four features. First, a modeling language separates the programmer from the modeler. This is particularly essential since there are different skill sets involved in writing codes and modeling. Second, it makes modeling closer to a pencil-and-paper kind of experience for the modeler. Optimization context will be handled more flexible and is more maintainable. Third, a modeling language limits the amount of coding that the programmer has to write to model a problem (compactness and code-assisted). Lastly, a modeling language provides services associated with the coding where debugging, profiling, and conflict finders are accessible. Kuip [71] studied algebraic languages for mathematical programming. Using the classification on Table 1.2.2, a translation from the algorithms' form to modeler's form for linear programming classification that is analogous to programming languages is presented.

Algebraic modeling languages (AML) for optimization are associated with software packages that bridge the gap between an analyst's natural mathematical formulation of an opti-

[2]Algebraic modeling languages are a special class of declarative languages.

Language class	Language for algorithms	Basis of similarity
Algorithm's form	Machine code	Executable representation
MPS form	Assembly	The use of mnemonics
Matrix generator	Procedural	Grouping of mnemonics
Algebraic	Functional	Resemblance to mathematical notation

Table 1.2.2: Modeling and Language for Algorithms Classification (KUIP)

mization model and the complex algorithmic routines that drive the solver. AML is capable of describing problems of the form

$$min \quad z = f(x) \qquad s.t. \quad x \in X, \tag{1.2.1}$$

where f is an objective function defined over a domain X of feasible solutions. AML enables a modeler to state the optimization problem in an indexed-based mathematical form with abstract entities such as variables, parameters, indices, sets, and constraints. The basic idea of AML is to represent an algebraic expression in a compact way. AML can group conceptually similar entities into a set; then they can be referenced by indices to the elements of this set. For example, $\sum_{i \in I} X_i$ is represented by

$$\begin{aligned}
sum(X) \quad & \text{in Matlab,} \\
SUM(I, X[I]) \quad & \text{in GAMS,} \quad \text{and AIMMS (with i),} \\
sum\{i \text{ in } I\} X[i] \quad & \text{in AMPL,} \quad \text{GNU Mathprog, and OPL,} \\
lpSum(X[(i)] \;\; for \;\; i \;\; in \;\; I) \quad & \text{in Pulp,} \\
sum(I : X) \quad & \text{in MPL,} \quad \text{and LINGO,} \\
sum < i > \;\; in \;\; I : \;\; X[i] \quad & \text{in Zimpl.}
\end{aligned}$$

These problem formulations are very close to the original algebraic notations. The purpose

of AML is to expand this compact problem structure (and data) into the problem instantiation[3]. Developers can define generic expressions that are indexed over several sets. Optimization modeling languages have promoted the independence of model, data, and solvers. This idea has proved to be beneficial to developers.

General-purpose	Solver-specific
AIMMS (Paragon Decision Technology)	LINGO (LINDO Systems)
AMPL (AMPL Optimization)	Mosel (Fair Isaac)
GAMS (GAMS Development)	OPL (IBM Corporation)
MPL (Maximal Software)	
GNU MathProg (Free Software Foundation)	
Zimpl (Zuse Institute)	

Table 1.2.3: List of some main stream modeling languages

In terms of implementation, algebraic modeling languages are available to various extents in a number of software packages [37]. Most AML software follow certain design principles: (i) a balanced mix of declarative and procedural elements, (ii) open architecture and interfaces with other systems, and (iii) different layers of separation. Table 1.2.3 presents a list of the most widely used general-purpose versus solver-specific modeling languages. For a complete optimization modeling languages review, refer to Fragnier and Gondzio [38] and the references therein.

Owing to the fact that a model is rarely a perfect representation of the problem, an optimal solution to that model is rarely the best solution to the problem. Due to a complex systems nature, qualitative dimensions, the system's uniqueness, and the need to engage a management team in a decision-making process, the traditional hard OR method may not be able to cope effectively with these complex modeling issues.

[3]Problem instantiation is the one ready to be solved by an appropriate optimization code.

1.2.3 Soft OR

At variance with conventional Operations Research, "Soft OR" explores different views from contributors by facilitating participation and engagement. Soft methods stem from systems-thinking theory. Prominent traits of these methods include structuring messy, complex situations that lack an agreed upon objective and are full of uncertainties. The soft methods for problem structuring have been developed over the last four decades. Among other methods, the most well-known are: Creative Thinking [88], Soft Systems Methodology (SSM) [15], Strategic Choice Approach (SCA), Strategic Options Development and Analysis (SODA), and Problem Structuring Methods (PSM). Heyer [59] studies the methods, application, and future of soft operations research in the defense setting. We believe that the soft methods can help modelers achieve more precise modeling. The study indicates six categories of Soft OR problems and maps into particular Soft OR methods. The categories range from (i) better client's problem understanding, (ii) choosing among courses of action, (iii) problem with high level of uncertainty, (iv) strategic and future issue, (v) analysis of interactions, and (vi) identifying areas for change. Figure 1.2.2 shows the elements of the systems movement and their relationship to one another in working toward problem-solving of real-world problems using soft system methodology.

Soft methodologies typically include "soft variables," e.g. perceptions of quality, user satisfaction, morale, etc. To *better understand a client's problem,* cognitive mapping, influence diagrams, SSM, and total systems intervention [19, 63, 92, 107] are prime methodologies. The outcomes could range from a simple visual representation of the problem to rich pictures for promoting understanding. To identify the *set of options for choosing among courses of action*, decision trees and journey making are selected as key tools [59]. For

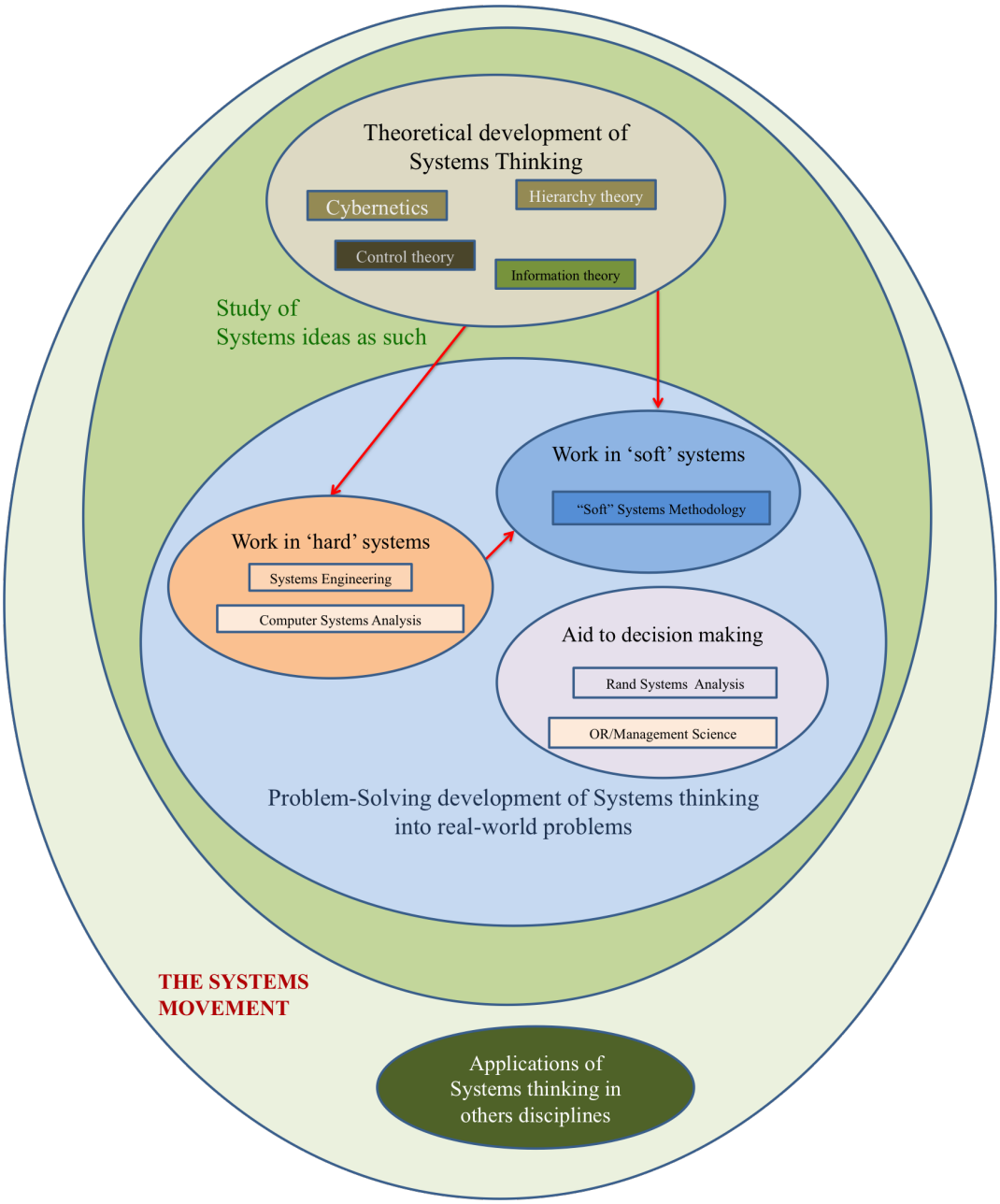

Figure 1.2.2: The systems movement [85]

problems with a high level of uncertainty, such as strategic-level decision making, robust analysis, SCA, and scenario planning are the most suitable methods to generate a list of possible decisions. For *strategic and future issues*, where group consensus on strategic issues are needed, the most relevant soft OR in this case are: SODA, scenario planning, SWOT, interactive planning, Viable System Methodology (VSM), journey making, and Delphi methods. Results are cognitive maps detailing emergent themes, scenario planning matrices, and a clear understanding of how to progress an organization from a current undesirable state. For *analysis of interactions, cooperation, and conflicts among multiple actors*, meta-games and hyper-games give an identification of decision options, as well as an understanding of differences in perceptions, maps for the best course of action, and constructs of possible future scenarios. Lastly, SSM is employed to *identify areas for change*, resulting in the development of action plans to effect changes.

Generic references include Checkland (1981), Ulrich (1994), Pidd (1996), Ormerod (1995-1999), Mingers and Gill (2000, 2006), Rosenhead and Mingers (2001), Keys (2006), and Paucar-Caceres (2009). More closely related references are, Bertalanffy (1950), Hall, Ackoff, and Churchman (1962), Beer (1966), Forester (1968), Beer, Espejo, and Harnden (1972, 1989), Ulrich and Mingers (1983-1985), Flood and Jackson (1991). Rosenhead (1996, 2004, 2009) also highlights a key concept of systems thinking that has contributed to Operations Research and Management Science.

Soft Systems Methodology (SSM)

SSM was developed by Checkland [14, 15, 16, 17, 18, 19] as an action-oriented process of inquiry into problematical situations where the 'hard' systems approaches have difficulties,

namely, problems that cannot easily be quantified; problems that can be quantified but have plenty of difficulties taking into account of the quantitative merits of such things as opinion, cultural, politics, viewpoints, or interaction. Figure 1.2.3 sums up the seven stages of SSM (more details in Chapter 3).

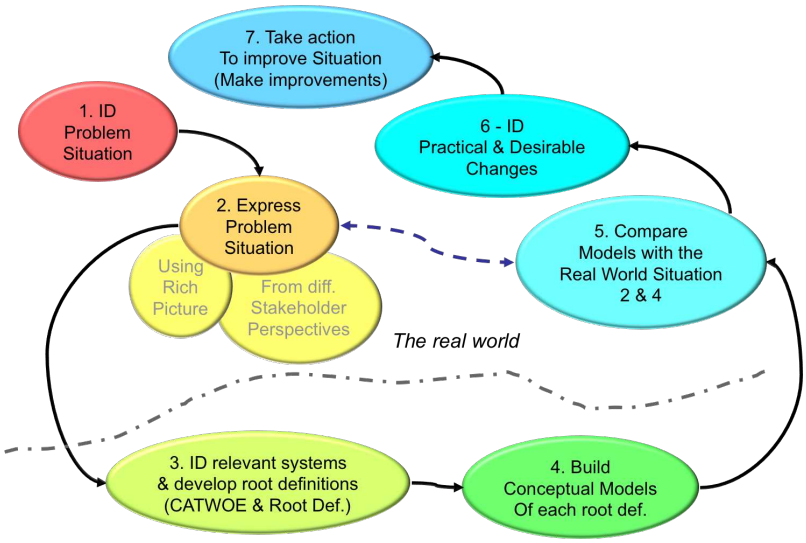

Figure 1.2.3: Seven steps of SSM

Users sort out an understanding of the situation to defining/taking action to improve it. The learning in SSM emerges via an organized process in which the real situation is explored, using as intellectual devices – which provide structure to the discussion – models of a purposeful activity built to encapsulate pure, stated worldviews.

Weltanschauung (Worldview): Weltanschauung is a fundamental concept of German philosophy that embraces a comprehensive worldview. This worldview may include natural philosophy; fundamental, existential, and normative postulates; or themes, values, emotions, and ethics.

Root definition: Root definition is precise textual statements that define the important elements of the relevant system being modeled, "a system to do X by means of Y in order to Z."

Customers	the beneficiaries of Z
Actors	those who do X
Transformation	input –> output
Worldview (Weltanschauung)	the world view that makes Y meaningful
Owners	those with the power to stop Z
Environment	elements outside the system

All the key concepts of a rich picturing of the situation, root definitions and conceptual models, CATWOE, and the vital concept of weltanschauung are in place in SSM.

1.3 Research Problem of Interest

Considering optimization problems that can be formulated in the most general form as follows:

$$\begin{aligned} minz &= f(x) \\ s.t.x &\in X, \\ F(x) &\in F, \\ x_i &\in Z \quad \text{for i} \in I, \end{aligned} \tag{1.3.1}$$

where $x = [\underline{x}, \bar{x}] = x \in \mathbb{R}^n | \underline{x} \leq x \leq \bar{x}$ is a box in $\mathbb{R}^n$, $f : x \to \mathbb{R}$ is an objective function defined over a domain X of feasible solutions, $F : x \to \mathbb{R}^n$ is a vector of constraint functions

$F_1(x),\ldots,F_m(x)$, F is a box in $\mathbb{R}^m$ specifying the constraints on $F(x)$ where inequalities between vectors are interpreted component wise, $I \subseteq 1,\ldots,n$ is the index set defining the integer components of x. The definition of a box includes two-sided bounds, one-sided bounds, and unbounded variables; since $\underline{x} \in (\mathbb{R}\bigcup -\infty)^n, \bar{x} \in (\mathbb{R}\bigcup \infty)^n$. An optimization problem is called bound constrained if $m = 0$ and is called unconstrained if in addition to $m = 0$, is $x = \mathbb{R}^n$. AML enables a modeler to state the optimization problem in an indexed-based mathematical form with abstract entities such as variables, parameters, indices, sets, and constraints.

application area / problem type	problem class
allocation problems (resources to orders, people to tasks)	MILP, CP
bending problems (production and logistics)	LP, MILP, NLP, MINLP
distribution and logistics (supply chain optimization)	MILP
engineering design	NLP, MINLP
financial problems (strategic planning)	MILP, MINLP
investment and de-investment problems (strategic planning)	MILP
market clearing problems	LP, MILP, NLP
network design (including planning and strategic planning)	MILP MINLP
portfolio optimization (production, finance, ...)	MILP, MINLP
process design	MINLP
production planning (production, logistic, ...)	MILP, MINLP
refinery planning and scheduling	NLP, MINLP
scheduling (planning subject to limited resources)	CP
selection and depot location problems (strategic planning)	MILP
sequencing problems	MILP

Table 1.3.1: Optimization problem class and its applications

Typically, an optimization problem can be classified by the nature of problem function $F(x)$, where significant algorithmic advantage can be taken of each characteristic [54]. Examples of such characteristics would include function of a single variable, linear function, sum of squares of linear functions, quadratic function, sum of squares of nonlinear

functions, smooth nonlinear function, sparse nonlinear function, and non-smooth nonlinear function. Additionally, a particular problem might be categorized as the minimization of a linear function subject to simple bounds.

Recall the standard form of Linear Programming (LP) problems, the minimization of a linear objective subject to linear equality and inequality constraints (all elements of must be non-negative). The standard-form LP problems can be stated as

$$\begin{aligned} &\min_{x \in \mathbb{R}^{\kappa}} c^T x \\ &s.t. Ax = b, \\ &x >= 0, \end{aligned} \tag{1.3.2}$$

the input data structures are the vectors and a matrix in sparse format, and the routines to generate these data structures are called matrix generators [36]. Optimization model formulation can be structured into a class such as LP problems, mixed integer linear programming (MILP) problems, nonlinear programming (NLP) problems, mixed integer nonlinear programming (MINLP) problems. In addition, constraint programming (CP) is counted as a common problem class as well. Table 1.3.1 shows the list of some possible applications of structured optimization problem classes.

It should be kept in mind that these optimization problem classes shown in Table 1.3.1, are neither exclusive nor exhaustive. Among these classes, there are several special cases of Eq. 1.3.1, differentiated by the properties of f, F, and I as follows:

- LP, if f and F are linear and $I \neq \varnothing$,
- NLP, if f or one component of F is nonlinear and $I = \varnothing$,

- MIP and MILP, and MINLP, if I is not empty and $I \neq 1, \ldots, n$.

In addition, Gill et. al. [53] summarized a typical classification scheme according to the general problem of interest form of the optimization problem (Eq. 1.3.1) as shown in Table 1.3.2, where significant advantage can be taken of each characteristic in choosing an algorithm.

Properties of objective function, $f(x)$	Properties of constraint, $F(x)$
Linear	None
Sums of squares of linear functions	Simple bounds
Quadratic	Linear
Sums of squares of nonlinear functions	Sparse linear
Nonlinear	Nonlinear

Table 1.3.2: Typical Classification Scheme

Certain problem characteristics have a much greater impact on the ease of optimization than others. The form of the problem constraints can have an enormous effect on the ease of solution. There is generally a very small increase in difficulty when moving from an unconstrained problem to one with simple bounds on the variables. General linearly constrained problems are noticeably more difficult to solve than those with bound constraints only, and the presence of nonlinear constraints introduces an even larger increase in difficulty. For these reasons, it is sometimes advisable to reformulate a model so as to eliminate nonlinear constraints; this topic will be discussed further.

1.4 Related Work

The following literature review focuses primarily on a survey of various intelligent modeling system attempts, where interdisciplinary theories are used from an integrated per-

spective for problem-solving and decision making in operations research. In 1962, Ackoff was chiefly the first to propose a systems approach to strategic planning in Operations Research applications [2, 3, 4]. His work indicates that one process cannot be optimal for all situations, his works was concerned with instability, non-linearity, discontinuity, and chaotic behavior which provided a foundation for Waldrop (1992) and Kauffman (1995) who studied "Chaos or Complexity Theory" [20, 5].

Simon [97] suggested key characteristics of well-structured domains in which (i) there is a definite criterion for testing a proposed solution, (ii) at least one problem space that can be represented for an initial problem state, (iii) a problem space has representable change, (iv) any knowledge that a problem can acquire can be represented in one or more problem space, and (v) a definition of state changes and of effects upon the state due to applying any operator in one or more problem spaces that govern the external world.

In late 1990, Checkland and Howell developed Soft Systems Methodology (SSM) in response to a perceived failure of traditional systems engineering. Perhaps, SSM has evolved to be the most popular Soft OR method. Rosenhead discovered systematic help in identifying an agreed framework for a problem, called Problem Structuring Methods (PSMs). PSMs were a family of interactive and participatory modeling approaches, requiring the implementation of group model building. Furthermore, PSMs provided support and stimulated group facilitation through modeling.

Ormerod published "Justifying the Methods of OR" using a philosophical terminology [84] in 2010. The paper attempted to find a balance between OR academic research and OR practice. Zhu studied why mixing-methodology theorizing fails and how to make it work again [112] in 2011. The research indicated that there are many opportunities for OR

mixing-methodology after the paradigm theorizing. The study explored a pragmatist alternative that is action-oriented, multiplicity-embracing, ethically concerned, and politically sensitive. The study asks whether there are particular types of problems more suited to mixing methods. Promoting ontological flexibility and methodology-in-use is a valuable starting point for after-paradigm theorizing that supports an innovative mixing-methodology practice. This raises an issue that it is not always clear if researchers are describing multi-paradigm multi-methodology.

In the 1980s and early 1990s, the model management research area was very active. Geoffrion developed the structured-modeling framework and implemented it using a structured modeling language (SML) [46, 47, 49, 50] to deal with the major problems[4] confronting the operations research and management science (ORMS) community. His approaches exploit the advantages of model typing in detecting numerous kinds of inconsistencies and errors in models [21].

Besides Geoffrion's attempts, several modeling languages were developed. These included a generalized algorithm for mathematical systems (GAMS) (Brooke et al.), a mathematical programming language (AMPL) (Fourer et al.), a linear, interactive, and general optimizer (LINGO) (Cunningham & Schrage), Advanced Integrated Multidimensional Modeling (AIMMS) , and Python Optimization Modeling Objects (Pyomo), to name a few. Recent modeling languages include Julia for Mathematical Optimization (JUMP), Gravity, and MINIZINC, and the list is still growing.

Fourer and Ma developed a unified framework for next generation distributed optimization systems called Optimization Services (OS) [76] in 2005. The goal was to integrate the

[4]Geoffrion noted that ORMS's activities at the time tended to be characterized as having low productivity. He also pointed out that managers and decision makers were reluctant to use a model-based approach [22].

algorithmic codes as services using XML. The purpose of this initiative was to address the issue of documentability, independence, modifiability, simplicity, and verifiability of optimization models and solvers. The ultimate goal of OS was optimization over the internet, where OS could provide a set of cooperative classes and interfaces to solve optimization problems.

In computer science, Bhrammanee and Wuwongse studied a formal framework that is model-based that was compatible with web-based technology [10], known as ODDM–OWL[5] Declarative Description Model-bases in 2008. They grouped the existing frameworks into two classes, as shown in Table 1.4.1: (i) the focus of content and (ii) the representation technique of decision models.

The focus of content	*data-centric*	hierarchical (AMPL)	Fourer, Gay, Kernighan (1990)
		entity-relationship	Blanning (1986)
		relational	Liang (1985)
	structure-centric	structured modeling	Geoffrion (1987)
		graphical modeling system	Sen, Chari (1997)
	abstract-centric	object-oriented	Lazimy (1993)
		RMT	Kwan, Park (1996)
	logic-centric	constraint logic programming (CLP)	Hiraishi (1995)
		logic modeling	Krishnan (1990)
	computation-centric	GAMS	Brooke, Kendrick (1998)
		Database structure	Fourer (1997)
		SML	Geoffrion (1987)
		LPL (Linear|Logical|Literate)	Hurlimann (2001)
The representation techniques	*graphic form*	E-R approach	Blanning (1986)
		Diagram technique for LP	Choobineh (1991)
		gLPS	Collaud, Pasquier-Boltuck (1994)
		SML	Geoffrion (1987)
		OOM approach	Lazimy (1993)
	text form	SML	Geoffrion (1987)
		GAMS	Brooke, Kendrick (1998)
		DB based	Fourer (1997)
		LPL	Hurlimann (2001)
		Logic modeling	Krishnan (1990)
	algebraic form	*generic representation technique in Optimization*	
	schematic form	OOF (Open Optimization Framework)	Ezechukwu, Maros (2003)
		XML-based	Kim (2001)
		Web service for Spreadsheet model	Lyer, Shankarakarayanan, Lenard (2005)

Table 1.4.1: Existing Model bases Work

[5]OWL stands for Web Ontology Language. It is built on top of RDF (Resource Description Framework) for processing information on the web. OWL was designed to be interpreted by computers. It is written in XML and has three sublanguages.

In 2016, Stapel's doctoral thesis on Ontology-Based Representation of Abstract Optimization Models for Model Formulation and System Generation [99] discussed an approach to the development of frameworks for the formulation and solution of optimization models, specifically the models representation using ontology, which is similar to the corpus representation and retrieval in my proposed framework. However, instead of aiming for a full automatization, my cognitive-driven framework extend the ideas of Geoffrion's Structured Modeling [41] to decompose and represent reusable model components, integrate models components into new components, as well as to perform semantic reasoning and present viable options to the modeler.

1.5 Thesis Outline

The dissertation has been divided into five chapters. First, Chapter 1 has introduced the motivations and research background. Chapter 2 presents a preliminary discussion, concerning a grand scheme of the research, common decision problems, cognitive computing system, and structured modeling. In Chapter 2, the modeling methodologies are discussed and integrated. In addition, Geoffrion's structured modeling and structured modeling language are extensively explored and reviewed. Afterward, Chapter 3 proposes an important extension for the structured modeling language. This will be the framework for corpus management and retrieval of models. A unified framework and problem-solving method for semi-structured problems are further explored in Chapter 4. Proof of concept, case study, and examples are provided at the end of Chapter 3, and 4. Finally, the conclusions and future research ideas are explored in Chapter 5.

Chapter 2

Preliminary Discussion

This chapter reviews relevant research to the present work and highlights key functional requirements for the study. The first part sets forth a grand scheme that is necessary for the later chapters. Next, a diverse problem classification is discussed. Common decision problems are further reviewed, followed by cognitive computing systems. Next, the focus shifts to Geoffrion's structured modeling and the structured modeling language (SML) that will be used as a foundation for the model corpus management. The last two sections review the fuzzification domain, and complex systems modeling.

Although not comprehensive, the material presented here is adequate to provide a substantial grounding in cognitive-powered modeling. The emphasis is on how these topics form more than just a miscellaneous collection but are part of an integrated approach for optimization modeling framework.

2.1 The Grand Scheme

Figure 2.1.1 presents a high-level view of the research with possible integration with a cognitive computation system like IBM $Watson^{tm}$, a supercomputer capable of answering questions posed in natural language. This grand scheme is composed of two layers: problem definition, and interface for decision makers. Element-wise, it can be divided into four parts: (i) real-world problems, (ii) primitive problem generation, (iii) scholarly and illiterate corpus, and (iv) crisp structured problem (refined hypothetical).

To begin, we start with a complex situation, where an individual or a group of decision makers (the strategic executives in a corporation) recognize the mess (problem(s)) and decide to tackle them with our framework.

2.1.1 Level I: Problem definition

This first level is the front end where a complex situation interacts with the systems. Decision makers feed the complex situation to the pipeline through a series of natural language dialogue (question-answering) by a guided discovery process, (possibly) utilizing IBM $Watson^{tm}$ cognitive computing power.

Problem primitive generation

Primitive problems are those that can be refined and can be efficiently implemented only if the problem-solver has access to the underlying representation of the abstraction. The primitive brainstorming provides a basis for thinking about problems. One way to quantify this resultant semi-structured data is to model it in terms of Object-oriented Petri Net

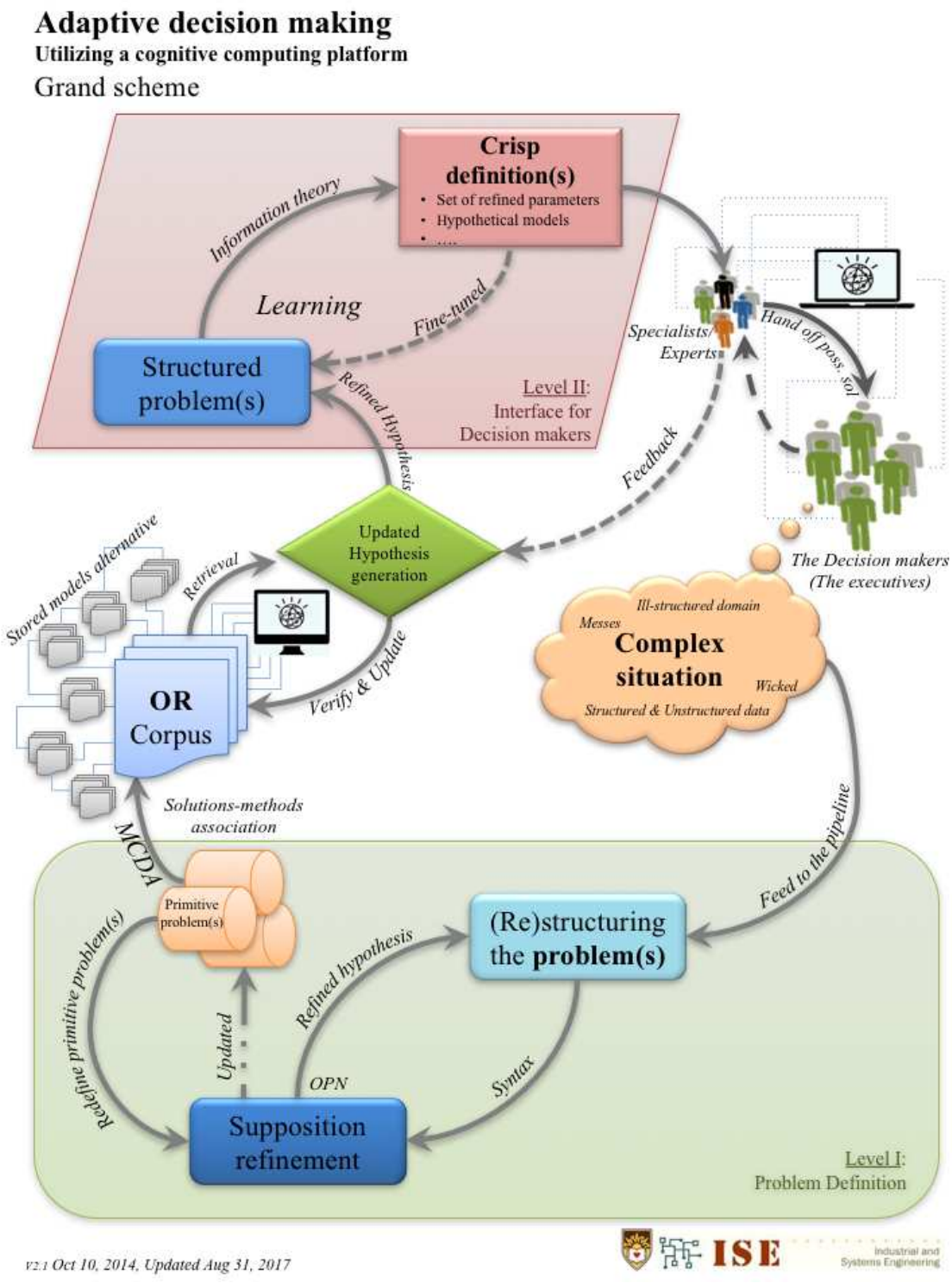

Figure 2.1.1: High level research thesis data and information flows

(OPN) representations. Hence, the mess can be clarified by rendering the structure of an OPN, if the Alternative World-view Representation (AWR) (or Alternative Weltanschauung Representation in German) is completed (we know the places, the transitions, and the ways in which the primitive problems are connected with each other).

The hypothesis refinement is achieved through the interpretation of Petri nets properties. Therefore, the syntax is exercised when we choose places and transition tags. Specifically, we look for names that bespeak what the transition or the place represents. Because places are the passive node of a Petri net since they cannot change the marking, we name places using nouns, adjectives, or adverbs. Transitions are the active nodes of a Petri net because they can change the marking via firing. We, therefore, name transitions with verbs to express action.

This process is reiterated until the complex situation is explained to the satisfaction of decision makers. Once completed, the information of primitive problems will feed forward to a solutions-methods association.

Scholarly and literate corpus

The middle interface is considered as a plethora of existing methods (OR corpus), combining structured and unstructured knowledge from an optimization perspective into the corpus. This division is vital to the research thesis due to the stipulation of forceful computing power for swiftly creating and updating hypothesis generation. It is very important to have well-defined boundaries that the system is expected to know and what its capabilities should be. For example, to master complex business problems, we might need to consider the input wisdom from related disciplines. The use of cognitive technologies and

IBM *Watson*^{tm} are viable as a driving force analogous with the functionality of a model management system (MMS) [29].

Geoffrion recognized the shortcomings of modeling systems in the 1980s [40] and, as a result, he developed "structured modeling" as a rigorous semantic framework that treats every OR model as a collection of distinct elements [41]. Therefore, we believe that structured modeling has numerous potential contributions to make to MMS. First, it provides a formal semantic ontology for models with its framework, based on graph theory. Hence, mathematical models benefit from this formalism because they can be represented as conceptual models. Second, structured modeling provides effective communication between analysts and decision makers. In this new era of cognitive computing, this modeling scheme also serves as a communication mechanism between IBM *Watson*^{tm} (as if it is an analyst) and decision makers. Our aim is to broaden this approach as a semantic instrument for this solutions-methods association state.

2.1.2 Level II: Interface for Decision Makers

The purpose of this layer is to develop crisp definitions that enable effective communication with experts and decision makers, i.e. At the beginning of this phase, the refined hypothesis (considered as a structured problem) is passed on as an improved structured problem-model interpretation, after specialists or experts have been allowed to provide input.

This is a repetitive learning process where the decision makers may interact with the IBM *Watson*^{tm} computer and/or specialists/experts, by providing feedback to refine the model definitions. In each iteration, our system fine-tunes these requirements and re-updates the hypothesis, resulting in handing off a possible satisfiable (in some case, op-

timal) solution to the executives/decision makers. A promising result of our research is the development of core theory, as well as the framework, that enables cognitive computing technologies for optimization-type problems in business analytics. In fact, utilizing unstructured data in the problem formulation, in our view, provides an answer to our original problem. (not just an answer to the optimization problems), hence, forming a cognitive-powered optimization adviser.

2.2 Diverse Problem Classification

Problems can be classified into three classes depending on the problem solver: well-structured, semi-structured, or ill-structured [34]. In addition, the problem classification depends on one's background and experience. An individual might view a problem as highly structured while another might have no idea what to do and view it as unstructured. Well Structured Problems (WSPs) represent a problem for which we have complete information and means of closing a modeling gap. Such problems are usually routine and repetitive and can be solved by ready-made solution techniques (algorithms). It is usually easy for a problem solver to recognize which algorithm to use. For instance, a variety of existing computer packages will find an optimal solution because very little creativity is needed to solve well-structured problems. However, Ill Structured Problems (ISPs) fall at the other extreme. They lack good information about the problem and are fuzzy about a present as well as a desired state of affairs. As a result, one often "muddles through" a solution. Such problems are so complex that complete information can never be obtained for them. In addition, human decision maker's behavior is erratic and unpredictable, and alternatives cannot be listed in a simple fashion. A high degree of creativity is needed to

address such problems.

Semi-Structured Problems (SSPs) fall between well-structured and ill-structured problems. There is some information available to define the problem partially, though enough uncertainty about actual or desired states exists to prevent routine procedures. We often use heuristics if our desired state of affairs is not sufficiently known. While hard data can be obtained and mathematical or simulation models can be developed, alternatives are often not easy to specify. They need to be discovered by trial and error through extensive experimentation or analysis. Furthermore, creative thinking can often aid a solution process and lead to solutions or solution methods that would not normally be considered.

Simon [97] extended the classification scheme from Ackoff and Sasieni for well-structured to ill-structured problems using five classification schemes as follows:

1. *Transparent problem*: a problem in which a logical structure is simple enough to be solved by inspection or discussion.

2. *Artificial problem*: a problem structure is apparent but the way to represent it symbolically is not clear.

3. *Exploration problem*: a problem structure is not apparent but there is the possibility of extracting structure by data analysis.

4. *Experimental problem*: it is not possible to isolate the effects of individual variables

5. *Insufficient problem*: a problem that has limited data, so an experiment cannot be conducted.

Eilon (as cited in Evans [34]) proposed another problem classification scheme; he classified decision-making problems into two dimensions, frequency, and replication. The frequency

of a given decision has to be made within a given period of time; it is a measure of a repetitiveness task. The replication refers to uniformity in a definition of the problem: the greater the variations, the lower the level of replication. Problems with a high degree of frequency and replication are generally well-structured and lend themselves to well-defined algorithms. To clarify, those with high replication but low frequency are still well-structured and suggest programmable decisions, but an economic issue exists as to whether cost savings or other benefits justify the effort and the expense involved in developing an automated procedure. Secondly, problems with low replication but high frequency are generally ill-structured, perhaps with many unpredictable and poorly defined characteristics and unclear objectives. Lastly, those problems with low replication and low frequency are the least amenable to pre-defined algorithms. The last type is our prime candidate for creative problem-solving approaches. Evans [34] presented classification schemes in more detail by function, structure, dimensionality, temporal reference, a degree of certainty, degree of generality, degree of closure, and the degree of quantification.

This study considers a problem with semi-structured or ill-structured characteristics. As shown in Figure 2.2.1, where a function may be descriptive or predictive, a structure of a problem is symbolic with multidimensionality under uncertainty. The temporal reference is dynamic, and degree of generality may be specialized (ad hoc) or general; a degree of closure may be closed or opened, and the degree of quantification is quantitative.

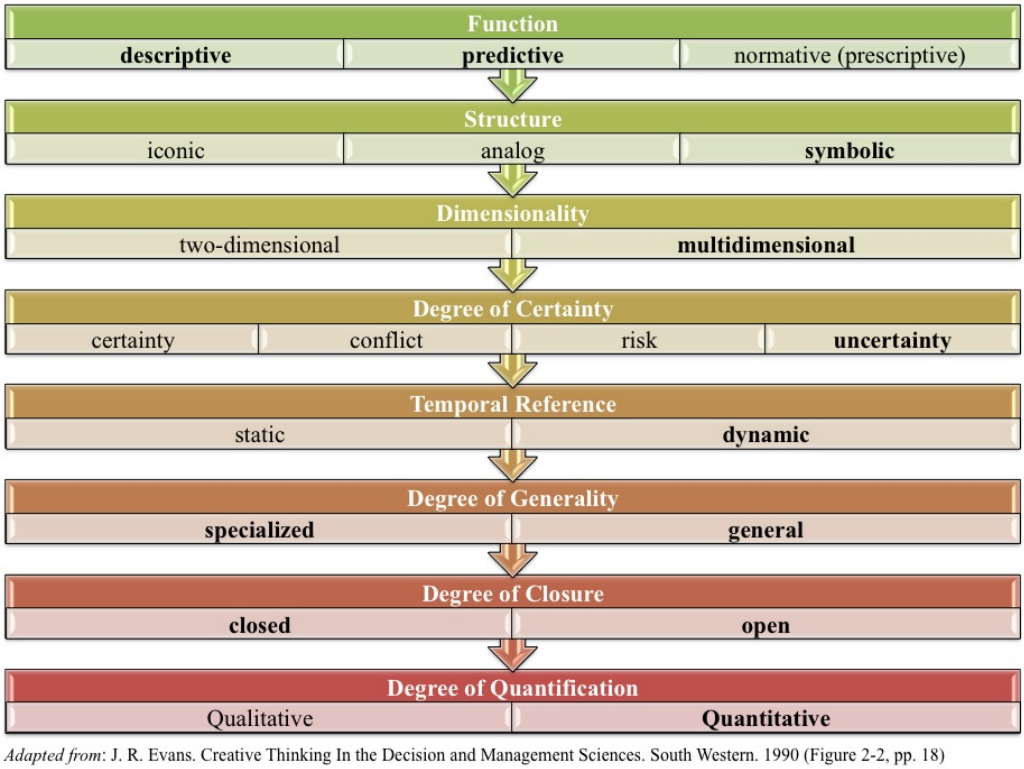

Figure 2.2.1: Problem scope, adapted from Simon [97]

Ackoff discusses a typical model of problem situations, considering a measure of a value of the decision that is made (action taken), V, as the functional relationship, f, of the independent variables, X_i, and the constant (or variable) parameters, Y_j. In this setting, the independent variables define alternative courses of action subject to control by the decision maker, while parameters that affect performance are not subjected to control by a decision maker. Such a model for problem situations can be represented as

$$V = f(X_i, Y_j) \tag{2.2.1}$$

A model of a problem situation has two essential characteristics. First, at least one of the "input" variables is subject to control by the person(s) confronted by the problem (i.e., it must model his or her possible choices of action). Second, the "output" variable must be a measure or index of the value of the alternative choices to the decision maker. Hence, models that satisfy these two conditions may be called decision or problem models. A dichotomy of problems has been depicted by RAND [2] as shown in Table 2.2.1, as follow.

Author	Domain name	Disciplinary affiliation
Ackoff	OR/systems	messes versus problems
Checkland	systems	soft versus hard systems thinking
Ravertz	history and philosophy of science	practical versus technical problems
Rittel	design	wicked versus tame problems
Schon	systems/management	swamp versus high ground

Table 2.2.1: A Dichotomy of Problems

Among others, Nadler[82] addressed the serious deficiencies of most problem classification schemes. These are: (1) deciding where a particular problem fits within a classification scheme; (2) classifying schemes that are descriptive at best; and (3) classifying schemes that lead to analytical techniques that limit the solution space and overlook critical thinking. In practicality, many problems cannot be pigeonholed into a fixed classification scheme. The classical classification will not work with them. This biased problem perception can cause an inappropriate problem formulation, and ultimately, a wrong solution to the intended problem.

2.3 Common Decision Problems

Decision making in an organization is referred to as the process of responding to a problem by searching for and selecting a solution or course of action that will create value for organizational stakeholders. Common decision problems deal with any arbitrary "yes-or-no" question on an infinite set of inputs.

Decision Problems

A decision problem has been called *the Entscheidungsproblem* (German for decision problems). The name was proposed by David Hilbert in the 1920s. In terms of a common decision problem, *the Entscheidungsproblem* poses the question: "does there exist an algorithm for deciding whether or not a specific mathematical assertion has a proof ? [106]" *The Entscheidungsproblem* is considered to be solved when a procedure that has a finite number of operations is valid or is satisfiable.

Hilbert's three questions are: (i) was mathematics complete, (ii) was mathematics consistent, and (iii) was mathematics decidable? To answer these questions, Alan Turing initiated the concept of algorithms and models of computation in his Turing machines. In the 1930s, Alonzo Church defined the notion of "algorithm" by proving that there is no computable function in a calculus problem. This initiative led to negative answers to *the Entscheidungsproblem*, where the first two questions were answered by Kurt Gödel's incompleteness theorem, and the last question was studied by the Church-Turing thesis on algorithms and decidability.

2.4 Cognitive Computing Systems

Cognitive[1] computing (CC) is a computerized simulation of human thought processes, benefiting from machine learning. Judith et.al. [33] suggest that cognitive computing technology facilitates humans to collaborate with machines. CC is a technology approach that enables humans to collaborate with machines. CC is a new type of computing with the goal of developing accurate models of how the human brain senses, reasons, and response to a stimulus. Cognitive systems differ from current computing as they shift beyond tabulating and calculating based on reconfigured programs and rules. Cognitive computing systems continually acquire knowledge (information) from the data fed into them. They address complex situations that are characterized by ambiguity and uncertainty. These systems refine this vagueness and look for patterns, as well as a way to process this data, becoming capable of anticipating new problems and modeling possible solutions. CC systems possess four cognitive properties: (i) adaptive, (ii) interactive, (iii) stateful, and (iv) contextual:

- *Adaptive* cognitive systems learn as information changes, and as goals evolve.
- The interaction between the system and users is the *interactive* feature.
- *Stateful* is a property whereby additional questions may be asked and additional input sources may be pinpointed to verify the lack of ambiguity of a problem statement.
- The *contextual* property is defined by drawing on multiple sources of information (or sensory input). This property is showing the comprehension and extraction of

[1] COGNITIVE |ˈkägnətiv| *adjective* of or relating to cognition. COGNITION |ˌkägˈniSHən| *noun* the mental action or process of acquiring knowledge and understanding through thought, experience, and the senses. a result of this; a perception, sensation, notion, or intuition.

contextual elements.

It is rare to find all four cognitive properties fully integrated and interactive. Cognitive computing systems redefine the nature of the relationship between people and their digital environments. Indeed, cognitive computing systems make context computable. The cognitive system requires integration of many elements. The model in a cognitive system refers to the corpus and the set of assumptions, algorithms that generate and score hypotheses. Figure 2.4.1 shows a continuous machine learning, which is a core element of a cognitive system.

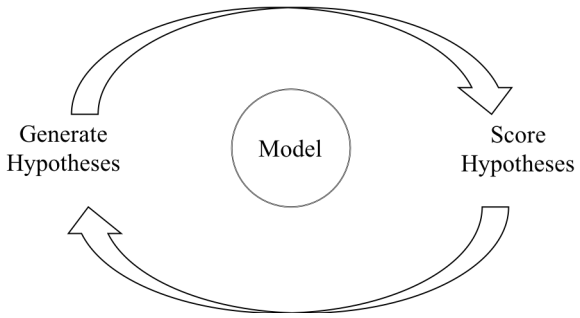

Figure 2.4.1: Continuous Machine Learning as part of a Cognitive System

It is crucial to have interaction between humans and machines as the systems learn through human interaction. In Figure 2.4.1, the continuous loop may be set to continue until a user is satisfied with the answer or until the system has evaluated all options. A cognitive system has three fundamental principles—(i) learn, (ii) model, and (iii) generate hypotheses—as described below [33]:

(i) A cognitive system learns and leverages data to make inferences about a domain of interest based on training and observation.

(ii) To understand the context, the cognitive systems need to create a model or representation of a domain and assumptions that dictate what learning algorithms are used.

(iii) A hypothesis is a testable assertion based on evidence that explains some observed phenomenon between elements within a domain. A cognitive system is probabilistic. It assumes that there is not a single correct answer. A cognitive system gains insight from data, it uses the data to train, test, or score a hypothesis.

CC are machines that operate at a different level than traditional computing systems because they learn and analyze from data input—text, images, voice, sensors, etc. For example, IBM $Watson^{tm}$ is a prime example of a cognitive computing system. It uses countless artificial intelligence applications (essentially natural language processing), information retrieval, and automated reasoning, yielding a powerful cognitive technology that can also be applied in advanced analytical situations. This research aims to utilize CC's capabilities to recommended adaptions in model formulations for the operations research analyst (OR analyst). Optimization-type problem formulations are considered as purely cognition-based human tasks, where reformulation of the optimization model is one such task [96]. An OR analyst typically becomes better at formulating and solving problems as their expertise in a model formulation domain increases with experiences. This mechanism, as shown in Figure 2.4.2 below, could be based upon a matching mechanism with model element retrieval from a library (model corpus). The modeling decisions might also be influenced by instance data. Purposefully, the suggested formulations could be used instead of the existing ones.

Learned model knowledge may be extracted for new, but similar problems, from a common optimization domain such as allocation, inventory, replacement, queuing, sequencing

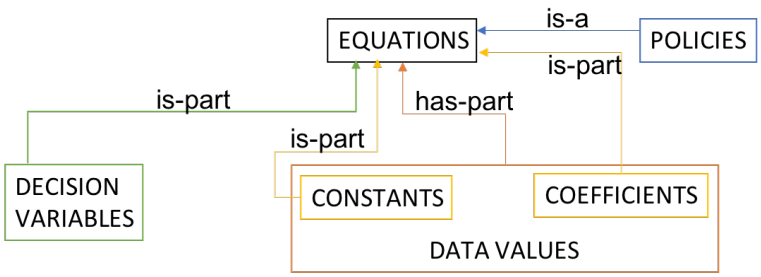

Figure 2.4.2: LP Semantics

and coordination, routing, competitive, and search [6], to name a few. Model reuse is part of an optimization model formulation assistant which is designed for supporting modeling tasks with the capacity of learning, generalizing, and updating modeling corpus using previous models as a basis. OR analysts can choose a class or family of optimization problems, define a new problem in this family and query the system for formulation choices on variables, parameters, objectives, and constraints. The research proposes a form of optimization model workbench where an OR modeler may query the system for advice and where a new experience becomes part of the existing training set. In the following session, simple optimization examples are considered to motivate the study. To respond to the fluid nature of an OR modeler's understanding of their problems, the cognitive computing system offers a synthesis of influences, contexts, and insights. To do this, systems often need to weigh conflicting evidence and suggest an answer that is "best" rather than "right."

Bluemix

IBM has made Watson technology available (as of this writing) as an open-standards, development platform in the cloud, called Bluemix. The purpose of IBM Bluemix is for building, running, and managing applications. Bluemix consists of applications, services,

build-packs, and other components. Bluemix is an IBM cloud platform environment that is built on the Cloud Foundry[2] open source technology. Bluemix offers more control to application developers by using its Platform as a Service (PaaS) offering. Bluemix provides mobile and web developers access to IBM software via pre-built Mobile Backend as a Service (MBaaS) capabilities. IBM's goal is to simplify the delivery of an application by providing services that are ready for immediate use and hosting capabilities. Thus, developers can focus on developing their application without having to manage the infrastructure that is required to host it.

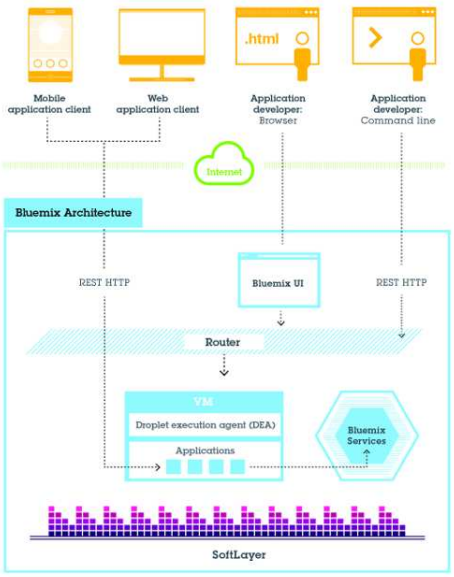

Figure 2.4.3: High-level Bluemix architecture (documentation Rev.: Feb 26, 2015)

Bluemix is an environment for building applications and user services for developers. As shown in Figure 2.4.3, Bluemix uses SoftLayer to deploy virtual containers that host each deployed application. Developers interact with the Bluemix infrastructure by using a web-

[2]Cloud Foundry is the industry's Open PaaS and provides a choice of clouds, frameworks and application services. (http://cloudfoundry.org/)

browser user interface or a Cloud Foundry command line interface, cf, to deploy applications. Clients use REST[3] or HTTP APIs[4] to route requests through Bluemix to an application instance or the composite services. The clients' interfaces can be mobile, web, or applications that are built on Bluemix.

Employing cognitive computing systems within the optimization domain is a journey that requires deeper understanding and insights beyond "question-and-answer". Therefore, our research attempts to capture natural mechanisms for modeling in optimization. Cognitive solutions are trained by subject matter experts (SMEs) (the training process is often referred to as supervised learning). Cognitive systems are probabilistic and not deterministic. As they learn over time, the accuracy rates of systems will improve accordingly. A system may not achieve 100 percent accuracy but the traceability of the machine recommendations will be important in cultivating confidence.

2.5 Geoffrion's Structured Modeling

Possibly the most important factor in modeling is the ability to grasp the essential structure of a problem or situation [94]. Structured Modeling (SM) is a conceptual model developed by Arthur M. Geoffrion in the late 1980s [29, 40, 41, 51] for this purpose. Geoffrion proposes SM to provide a practical and powerful framework for representing various kinds of models. It is a systematic way of thinking about models and their implementations, based on an idea that every model can be viewed as a collection of distinct elements. Each element

[3]REST (Representational State Transfer) is a service interface that allows applications to interact directly with Watson.

[4]API (Application Programming Interfaces) is a specified software component in terms of its operations, inputs, outputs, and underlying types.

has a definition which is either primitive or based on the definition of other elements in the model. Structured Modeling aims to provide a formal mathematical framework and computer-based environment for conceiving, representing, and manipulating a wide variety of models [41]. It is a general approach to modeling. SM is composed of six element types that are defined in the following:

(1) Primitive Entity elements, $/pe/$ have no related value and generally portray concepts hypothesized as primitives or concepts in the model.

(2) Compound Entity elements,$/ce/$ also have no related value and generally represent concepts that are defined in terms of other concepts.

(3) Attribute elements, $/a/$ have a constant value and generally represent properties of concepts.

(4) Variable Attribute elements, $/va/$ are similar to the attribute elements but their values are discrete and likely to change over time.

(5) The function elements, $/f/$ have a value that is dependent according to an explicit rule on the values of called equations; typically, the $/f/$ property can be calculated.

(6) The test elements, $/t/$ have a similar form to$/f/$. However, their values must be Boolean (i.e. either True or False).

The outcomes of SM are so called 'structured models' which are equivalent to semantic data models. SM's goal is to represent a new generation of modeling systems where the conceptual framework for a modeling life cycle can be represented in a format suitable for managerial communication while maintaining its mathematical complexity and being

executed on a computer. SM also uses organized hierarchy and partitioning to portray the semantic, along with the mathematical structure of a model. A structured model should invoke various types of solver supports. These can be thought of as residing in a "models library," where they are conveniently available for use whenever needed.

2.5.1 Structured Modeling Language (SML)

SML is a language for expressing a structured model that was developed by Geoffrion[49, 50]. It is executable and fully supports Structured Modeling's semantics framework. There exist other languages for SM (may have limited expressive capability), ones that are graph-based, logic-based, SQL-oriented, subscript-free, and object-oriented. SML can be represented by four levels of increasing expressive power that are upwardly compatible. Table 2.5.1 summarizes the current SML expressive power. Our work focuses on extending this power and introduces structured modeling with cognitive computing power (*level 5 addition to SML).

Geoffrion gives the conceptual framework of SM concepts in [41]. In SML, the *schema* is used to represent the *general structure*, while *Elemental Detail Tables* are used to representing *instantiated data*. Similar elements can be grouped and form a *genus* (plural *genera*), which may be organized hierarchically to manage model complexity.

In brief, Level 1 SML is simply a subset of the whole SML language. However, there must be only *one* element per genus, and there may be *no* value-bearing elements [49]. Next, Level 2 SML is adding values (real, integer, logical, or string) to structural modeling. Models in level 2 may have the following five extensions: definitional systems, graphs, spreadsheet models, formula-oriented models, and propositional calculus models. Level

1: Structural Modeling	- Definitional system; - Structural Models/Graphs; (labeled graph)
2: Structural Modeling with Value-Bearing elements	(all of level 1) and - added values/computation - Spreadsheets; - Numeric formulas; - Propositional calculus;
3: Structural Modeling with Simple Indexing	(the indexed version of the above) and - Mathematical programming; - Predicate calculus models;
4: Structural Modeling with Full Support for Sparsity	(the indexed version of the above) and - Relational Data Base models; - Semantic Data Base models;

Table 2.5.1: SML expressive power

3 SML adds index structure, sets, and Cartesian products, which is a key property for Structured Modeling possessing dimension independence. Lastly, Level 4 SML offers full support for sparse indexing structures. Level 4 enables all relational and most semantic databases. The next sections explore each level of SML, followed by an example-driven instantiation to illustrate the SML framework.

2.5.2 Level 1 Structural Modeling

Level 1 is the simplest representation. Starting with two types that are correlated, acyclic, and hierarchical, the outcome of this level can be labeled graphs and structural models. Appendix A summaries the definitions used at this level as core concepts in constructing SML. The general appearance of Level 1 SML includes: schema, genus paragraph, and module paragraph as shown in figure 2.5.1

SML is relatively close to natural language in that it is nearly free-form except for defined

```
Schema (given a modular outline)
       Genus Paragraph (/pe/, /ce/)
              Genus Name
              Genus Type Declaration
              Calling Sequence (simple)
              Interpretation
                     Defined Key Phrase
                     Referenced Key Phrase
       Module Paragraph
              Module Name
              Interpretation
                     Defined Key Phrase
                     Referenced Key Phrase
```

Figure 2.5.1: Level 1 SML features

key phrases and reserved words. Level 1 SML captures the problem representation by Schema, a collection of genus paragraphs and module paragraphs. Designed by Geoffrion, the genus name is indicated by the "$" symbol with uppercase letters, followed by the first two genus type declarations (primitive $/pe/$, compound $/ce/$) and calling sequence for a compound entity. The module paragraph name is globally unique in a schema, starting with an ampersand symbol. Moreover, each collection in a schema has free-form comments as the interpretation.

Structural modeling illustration using the transportation problem

Referencing Section 2.6.1, for the transportation problem discussed in [68], their problem is to minimize the transportation cost by finding the set of shipments x from plants $\{Dallas, Chicago\}$ to customers $\{Pittsburgh, Atlanta, Cleveland\}$. We show a Level 1 SML schema for this transportation model in Figure 2.5.2, below. Formally, Level 1 SML can represent any structural model that can be represented by a graph [49].

Level 1 SML serves the simplest representation of definitional systems. Figure 2.5.2 shows

```
&SDATA                SOURCE DATA
   PLANT /pe/         There is a list of PLANTS.

&CDATA                CUSTOMER DATA
   CUST /pe/          There is a list of CUSTOMERS.

&TDATA                TRANSPORTATION DATA
   LINK(PLANT,CUST) /ce/      {PLANT} x {CUST}
                      There are some transportation LINKS from
                      PLANTS to CUSTOMERS. There must be at least
                      one LINK incident to each PLANT, and at
                      least one LINK incident to each CUSTOMER.
```

Figure 2.5.2: Level 1 SML Schema for transportation model

a simple schema for our 2×3 transportation problem. The schema encompasses named genus and module paragraphs, which are organized into an indented outline [43]. The module paragraphs are using "&" as a prefix, noting that at Level 1:

- each genus paragraph hosts *one* definition of the definitional system;
- the genus name is indented under a module paragraph, informal English is used for description and key phrases are shown as being underlined;
- the SM notational system is declared after the genus name using '/ /', i.e. $/pe/$ for primitive entity, $/ce/$ for compound entity.

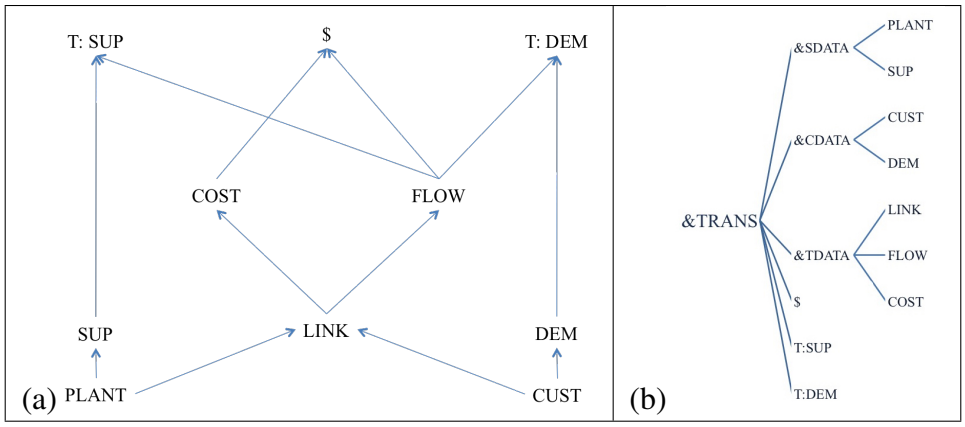

Figure 2.5.3: Genus graph (a) and modular structure (b) of transportation model

Insightful graphs are always available and easy to generate in the SM framework. Figure 2.5.3 (a) shows the aligned genus graph, which could be transformed into the element graph (if we annotated with nodes and arc attributes). Figure 2.5.3 (b) shows the modular structure tree, which is defined as a generic structure. The logic of this modular structure becomes part of our model.

2.5.3 Level 2 Structural Modeling with Value-Bearing Elements

Level 2 enhances the first level (permits level 1 to have values). Now, three more types of definitions are defined, numerical formulas, propositional calculus, and spreadsheets, that are in addition to the structural models and labeled graphs that already exist in the Level 1 SML. There is still one element per genus; however, Level 2 has a true logical capability. Appendix A (2) summaries the definitions used in level 2. This concludes the foundation structure of SML. The appearance of Level 2 SML is as shown in Figure 2.5.4.

```
Genus Paragraph (/a/, /va/)
   Range Statement
      Subrange
Genus Paragraph (/f/, /t/)
   Generic Rule
          Expression
                   Primitives
                     Nonnegative Integer
                     Nonnegative Real
                     Truth Value
                     Quoted String
                     Simple Variable
                   Functions
                     Arithmetic Operators
                     Standard Functions
                            Numeric-Valued Standard Functions
                            Logical-Valued Standard Functions (simple)
                     @IF Function
                     Relational Operators
Elemental Detail Table (stubless)
```

Figure 2.5.4: Level 2 SML features

The notational characteristics of SML permit the ability to separate the model from the problem statement. In Level 2, four additional genus type decorations are declared–attribute $/a/$, variable $/va/$, function $/f/$, and test $/t/$. The range statement uses four data types (Integer, Real, String, Logical) to represent attribute and variable genus type. The lower and upper limit of the range statement is delineated by sub-range. A stub-less elemental detail table must be given for $/a/, /va/, /f/$, or $/t/$ to hold values for them. Adjoining subsets of elemental detail tables may be joined. As level 2 has only one element per genus, there will be only one row at this level.

Structural modeling with value-bearing elements illustrative example

Level 2 SML serves as an additional information layer to the model. Three more entities are now supported in Level 2: attribute, function, and test. As shown in Figure 2.5.5, an example of Level 2 shows column-wise elemental details (each column represents $/a/$ and

$/f/$). Notice that all attributes in this example are the reals (Geoffrion [49] noted as a default in the absence of schema). The interpretation of Figure 2.6.2 in Sub section 2.6.2 is self-explained; the cell name applies to module paragraph and genus name. The cell formulas are expressed next to the function $/f/$ element.

```
SHIPMENT_SAVING /pe/          SHIPPING WORKSHEET
&FREIGHT_LOG                  FREIGHT data
    &VAR_PARAM                List of parameters (percentage of purchasing price)
        CARRYING_CHG /a/      Cost of holding inventory
        FREIGHT_RED /a/       Percentage of freight reduction
        COST_RED /a/          Negotiation buying price from supplier(reduction)

    &PROD_ITM                 Original condition PRICE
        PRICE_P /a/           Purchasing price (USD)
        FREIGHT_COST /a/      Freight cost (estimate per unit)

    &MONTHLY                  MONTH TO MONTH PURCHASING
        SUB (PRICE_P, FREIGHT_COST) /f/ ; PRICE_P+FREIGHT_COST
        OWN_C (SUB, CARRYING_COST) /f/ ;  SUB+(SUB x CARRYING_CHG)

    &PRICE_AND_FREIGHT_RED    REDUCTION PRICING
        PRICE_R (PRICE_P,COST_RED) /f/ ;  PRICE_P-(PRICE_P x COST_RED)
        NEW_FREIGHT (FREIGHT_COST,FREIGHT_RED) /f/;
                              FREIGHT_COST-(FREIGHT_COSTxFREIGHT_RED)
        SUB_SV (PRICE_R,NEW_FREIGHT) /f/ ;PRICE_R+NEW_FREIGHT

        OWNERSHIP_COST (SUB_SV,CARRYING_CHG) /f/;
                              (SUB_SVxCARRYING_CHG)+SUB_SV
```

Figure 2.5.5: Level 2 SML Schema for shipment and freight logbook

2.5.4 Level 3 Structural Modeling with Simple Indexing

While Level 2 SML provides the unique characteristic of separation of models from problem statements and solvers, Level 3 SML provides a separation of general structure and instantiating data. This unique property allows exploitation of parallel structure and predetermined relational data table structure. Level 3 encompasses simple indexed versions of all previous model types. It also adds a simple indexed version of mathematical program-

ming and predicate calculus. The general appearance of Level 3 SML is as shown in Figure 2.5.6.

```
Unindexed and Indexed genera
        Self-Indexed
                Index, Alias Index
                Identifier
        Externally Indexed
Generic Calling Sequence
        Calling Sequence Component
        Generic Name
        Generic Index Tuple
Index Replacement Options
        Simple Index Replacement Options
        Functional Dependency
                Dependent Index, Independent Index or Indices
                Functional Dependency Name
                Functional Dependency Table
        Multi-Valued Dependency
                Dependent Index, Independent Index or Indices
                Multi-Valued Dependency Name
                Multi-Valued Dependency Table
Range Statement
        Uniqueness Declaration
Index Set
        Specific Index Tuple
        Dense Index Set
        Index Set Statement (ISS) for Self-Indexed Genera
Expression (new features)
        Primitives
                Simple Variable (indexed)
        Functions
                Index-Supporting Functions
                        Local Index
                        Local Index Renaming
                Ordinate Functions
                        Index Ordinate Function
                        Functional Dependency Ordinate Function
Elemental Detail Table (general case)
        Stub
        Interpretation Column
        Joining
```

Figure 2.5.6: Level 3 SML features

For an *Unindexed and Indexed genera* paragraph, the lower-case index and alias index are separated by commas. An external index relies on a non-empty generic index tuple. All types, except $/pe/$, are applied for a *Generic Calling Sequence* paragraph. All types, except primitive, are indexed by a period and replace an index by $< integer >$. The generic name is a name with an index tuple. The generic index tuple is the genus's index for a self-indexed genus; it is empty for an unindexed genus. Next, in the *Index Replacement*

Option paragraph, the "simple index replacement options" replace indexes by $< integer1 : integer2 >$. The dependent index with positive integers appends in a functional dependency name. In *Range Statement*, a uniqueness declaration terminates a range statement in a keyword. A *specific index tuple* is a generic index tuple with a particular identifier in place of each index.

Only Level 3-SML has a dense index and a table containing an index set; furthermore, the index set statement (ISS) limits the size of the index set i.e. $P \leq Size\{GenusName\} \leq Q$. The new feature here is the expression, including primitives and functions, where primitives represent a general kind of simple indexed variable. *Functions* embody index-supporting functions and index ordinate functions where local index renaming is permitted for an index via one or two primes. Lastly, the *Elemental Detail Table* uses stub as an independent column where one table per genus is needed. This table has to be in the same order as the represented genus paragraph throughout the SML.

Structural modeling with simple indexing illustration example

Level 3 SML satisfies most mathematical programming models with an indexing formulation. Now, we revisit the previous classic transportation example (section 2.6.1) where we assume all conceivable plant-to-customer transportation links exist[5] (refer to Figure 2.6.2 in Section 2.6.1). Note, that we need a level 4 SML as a model for which some links may not exist.

Notice that the genera *PLANT* and *CUST* now add indexing (simply after its name). This indexing is also applied for other model classes at level 2 such as spreadsheets and formu-

[5]Dense indexing: we call the indexing 'dense' if an index record appears for every search key value in the file. The record contains search key value and a pointer to the actual record.

las. Geoffrion demonstrates this example in [41, 43].

We continue with the original example and follow the work done by Kendrick and Krishnan [68] as shown in Section 2.6.2. Since we omit the display of elemental detail tables, we shall use labels in the spreadsheet of Figure 2.6.3 as elemental names shown in Figure 2.5.7.

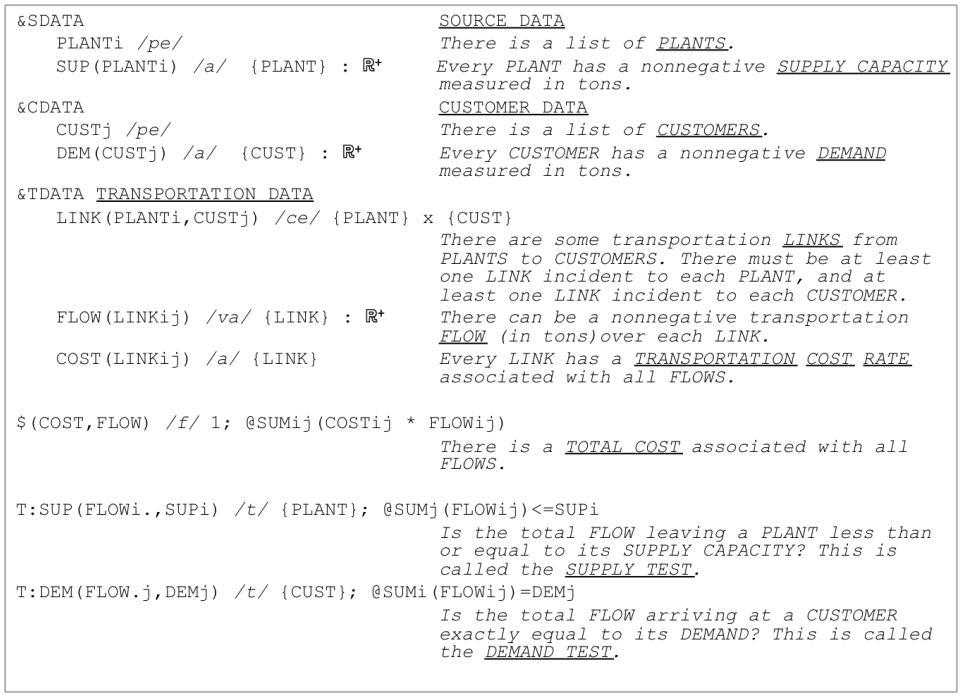

Figure 2.5.7: Level 3 SML Schema for transportation model

SML's elemental detail table has a genus name (on the left of || in Figure 2.5.8) which serves as a key, resulting in a set of relational algebraic equations.

Table Name	Column Names
PLANT	PLANT ‖ INTERP SUP
CUST	CUST ‖ INTERP DEM
LINK	PLANT CUST ‖ FLOW COST
$	‖$
T:SUP	PLANT ‖ T:SUP
T:DEM	CUST ‖ T:DEM

Figure 2.5.8: Elemental detail table for Dense Transportation

Figure 2.5.8 above shows the elemental detail table for our classic transportation model. Geoffrion dedicates the vertical double bar as an indicator for a key. As a result, models can refer to their instantiating data. Furthermore, SML is highly compatible with any relational database system.

2.5.5 Level 4 Structural Modeling with Full Support for Sparsity

In addition to Level 3, Geoffrion [50] discusses five additional notable characteristics apparent at Level 4 SML: *(1)* sparsity support, (2) explicit semantic framework, (3) exhaustive context-sensitive semantic restrictions, and (4) standardization through generality, and (5) executability. He also highlighted three important characteristics of SML in [50] as follows.

(i) *Separation of General Structure and Instantiating Data:* To allow the flexibility of modeling complex situations, SML uses *schema* to represent *General Structure* and put *Instantiating Data* in the *Elemental Detail Table*. Hence, we are able to differentiate the purposes of the general structure and the elemental detail.

(ii) *Exploitation of parallel structure:* It is obvious, as Geoffrion[43] highlights, that grouping as one of the most appealing properties for definitional systems. The group-

ing property extends to modeling languages. Furthermore, it enables us to utilize existing models that have common parts along with an extension from any of the parts.

(iii) *Predetermined Relational Data Table Structure:* Benefiting from a unique schema elemental detail tables rules set, modelers do not have to worry about data structures or the design table.

```
Feasible Specific Index Tuple
   Default Index Set
Generic Calling Sequence (new feature)
   Floating Index Replacement Option
Expression (new feature)
   Primitives
           Simple Variable (indexed)
              Offset Indexing
   Functions
           Functional Dependency Ordinate Functions
              Offset Indexing
           @EXIST Function
           Index-Supporting Functions
                  Floating Local Index Range
                  Default Operand Value
Index Set Statement (ISS) for Externally Indexed Genera
   Select
   Relational Expression
           Generalized Index Set
                  Symbolic Index Tuple
           Set Operators
                  Cartesian Product
                  Difference
                  Intersection
                  Natural Join
                  Projection
                  Selection
                      Filter Formula
                  Union
   Qualifier Sentence
           Qualifier Phrases
                  Extensive vs. Restrictive
                  Binary Relations
                  Covering
   ISS Rules
```

Figure 2.5.9: Level 4 SML features

Level 4 encompasses richly indexed versions of all previous three levels. It supports rela-

tional and semantic databases. This is an important property for many applications, especially, mathematical programming. The general appearance of Level 4 SML is as shown in Figure 2.5.9. Level 4 SML is composed of four main paragraphs. The first one is the feasible specific index tuple where any specific index tuple can be substituted into the generic calling sequence. The set of all feasible index tuples is represented by the default index set. Next, the floating index replacement option identifies a particular identifier or interval of identifiers. In the expression paragraph, the simple variable is now indexed using offset indexing, i.e. $< index + integer >$ or $< index - integer >$. In the functions subparagraph, offset indexing is used for simple variables and the functional dependency ordinate function. Level 4 uses $@EXIST$ function to resolve non-existent references in a generic rule. The floating local index range allows greater control of the local indices. Lastly, the Index Set Statement (ISS) for externally indexed genera confers responsibility on the user for specifying the desired tuples in the elemental detail tables.

Structural modeling with full support for sparsity illustration

Figure 2.5.10 presents an SML schema for an inventory-routing example referred to in the IRP literature (presented in Section 2.6.3 and model formulation in Appendix C). The mathematical program for this problem is now rendered by SML schema Level 4. There are several language features worth noting here. First, for the index set statement indicated after some genus type, we can easily associate and impose constraints on the entity by declaring the index set after the genus name. In the B_t in the SUPPLIER module, there are three associated index sets, for example. Next, the calling sequence can incorporate finer control over calling elements, i.e. $x_{s<t-1>}$ specifies the particular shipping quantity to customer s at the previous time period $(t-1)$ or $x_{s,t-1}$. Due to the case sensitivity in SML

schema, we use a subscript to depict some index to preserve the example originality (i.e. small capital attribute and variable value).

Geoffrion gave an informal explanation of the increasing expressivity power of SML in [47]. Analogously, consider a transportation problem example in Section 2.6.1, where an informal explanation of SML may be expressed as follows:

- Level 1 as shown in the illustrative example. Figure 2.5.2, can be viewed as a text-based graphic representation. Nodes and arcs are named, then, represented by a primitive entity, and compound entity respectively. These elements may be organized hierarchically, distinct from the graph.
- Level 2 lets those nodes and arcs have values (prespecified or calculated by formula).
- Level 3 introduces data tables to assemble node and arc information by the defined groups. Similar nodes and arcs are indexed and organized by dense group (not involving index-based sub-setting).
- Level 4 allows the use of index-based subletting for defining the sparse composition of nodes and arcs into groups.

&IRP_DAT — GENERAL PARAM

h_0 /a/ — A unit inventory cost at supplier

h_s /a/ — A unit inventory cost at customer (retailer) *s*

u_s /a/ — Consumer rate

U_s /a/ — Maximum inventory level of customer (retailer) *s*

x_{st} /v/ — SHIPPING QUANTITY to customer *s* at time *t*

y_{st} /v/ :*Binary (1,0)* — Variable for determining a transportation cost, being equal to 1 if the customer *j* is followed by customer *i* at time *t*

z_{st} /v/ : *Binary (1,0)* — Value is equal to 1 if the customer *s* is served at time *t* otherwise 0.

&PLANNING_PERIOD — PLANNING HORIZON

τ/pe/ — Planning horizon, $t \in \tau = \{1, \ldots, H\}$

$t(\tau)$/a/ — Planning period within a planning horizon.

&SUPPLIER — SUPPLIER DATA

SUPPLYi/pe/ — There is a list of SUPPLIERS.

Bt(SUPPLYi, $U_{<0t-1>}$, x_{st}) /a/ — Every supplier has a nonnegative starting inventory at period *t*.

&CUSTOMER — CUSTOMER DATA

CUSTs /pe/ — There is a list of CUSTOMERS (RETAILERS).

$I_{st}(x_{s<t-1>}, u_{s<t-1>})$ /a/ — Inventory level of customers *s* at period *t*.

&VEHICLE — VEHICLE DATA

TRk/pe/ — There is a list of TRUCKS *k*.

Q(TRk) /a/ — Every TRUCK has a nonnegative VEHICLE CAPACITY measured in units (i.e. tons)

c_{ij}/a/ — The delivery COST to ship inventory from *i* to *j*.

&TSTOCK_DATA — STOCKING DATA

T:B_t(SUPPLYi, $U_{<0t-1>}$,x_{st}) /f/ {SUPPLY}; Bt=$B_{<t-1>}$+$U_{<0t-1>}$-@SUMs($x_{s<t-1>}$)

T:B_t(SUPPLYi, $U_{<0t-1>}$,x_{st}) /t/ {SUPPLY}; $Bt \geq @SUMs(x_{st})$

Is the supplier stockout?

&TINVENTORY_DATA — INVENTORY DATA

T:I_{st}($x_{s<t-1>}$,$u_{s<t-1>}$) /f/ {CUST}; $I_{st} = I_{s<t-1>} + x_{s<t-1>} + u_{s<t-1>}$

T:I_{st}($x_{s<t-1>}$,$u_{s<t-1>}$) /t/ {CUST}; $I_{st} \geq 0$ Making sure that the customer's inventory is not stockout.

T:OU(Us,Zst,Ist) /t/ {CUST};

$x_{st} \geq Us \times Zst - Ist$,
$x_{st} \leq Us - Ist$,
$x_{st} \leq Us \times Zst$.

Is the order up to - replenishment level satisfied?

&TCAPACITY_DATA

T:VEH.CAP(x_{st},Qk) /t/ {TRk}; @SUMs(x_{st})≤Qk(TRk)

T:ROUTING(x_{st},Qk,Z0t) /t/ {CUST,TRk}; @SUMs(x_{st})≤Qk(TRk)×Z0t

&T:ADDT_VALID_INEQ — ADDITIONAL VALID INEQUALITY

T:SERV_RETAILER($I_{s<t-k>}$,$Z_{s<t-j>}$) /t/ {CUST,TRk};

$I_{s<t-k>} \geq (1 - @SUMj(I_{s<t-j>})) \times @SUMj(U_{s<t-j>})$

Does the retailer get a service in time? (or, Is the retailer served in time?) *t-k, t-k+1,…,t*?

T:OU(Ist, Us, $Z_{s<t-k>}$, u_s) /t/ {CUST, IRP_DAT};

$Ist \geq Us \times Z_{s<t-k>} - @SUMj(u_{sj})$

At time *t-k*, Is it the last time customer *s* was visited before time *t* ?

T:ROUTE_ADDT(Zst,Z0t) /t/ {CUST,TRk};

Zst≤Z0t

Is the supplier included in the route when a customer *s* visited at time *t*?

T:PATH(y_{ij},Zit) /t/ {CUST,IRP_DAT};

$y^t_{i0}(\tau) \leq 2 \times Zit$,
$y_{ij}(\tau) \leq Zit$,

Is the supplier *i* that is the successor of customer *j* in the traveled route visited at time *t*?

$(Bt,Ist,TRk) /f/ @SUMt($h_0$×Bt)+@SUMtSUMs($h_s$×Ist)+@SUMijSUMt($c_{ij}$×$y_{ij}(\tau)$)

There is a TOTAL COST associated with retrieving and maintaining inventory, and shipping from a supplier(s) to customers.

Figure 2.5.10: Level 4 SML Schema for IRP example

In contrast with algebraic modeling languages that are commonly used for mathematical programs, the Structured Modeling approach is developed through the cognitive process of modeling, founded on an explicit semantic framework. SML is a highly expressive language. It has been shown that four divisions of levels give a useful classification and the capability of various implementations [45, 48, 52, 70, 67, 77, 101, 62]. Four additional characteristics pertaining to SML as a whole are:

(i) founded on an Explicit Semantic Framework,

(ii) exhaustive Context-Sensitive Semantic Restrictions,

(iii) Standardization through generality, and

(iv) executability.

In this digital age, where big data comes into play, SML requires additional expressivity to keep up with advances in both computation and information complexities. In the following Chapters, an extension of Structured Modeling that will benefit the cutting-edge modeling process that may draw on unstructured data is developed and described, an outcome of the dissertation research effort.

2.6 Selected Simple Examples for SML Illustration

This section presents selected toy examples for demonstrating Geoffrion's SML representation, starting with a similar Transportation problem to the one used as an illustration for the Level 1, 2, and 3 SML in the literature [49, 50]. We add the spreadsheet example for

shipments and freight log as a good demonstration for the Level 2 SML. We also add the study of the inventory-routing problem for illustrating the Level 4 SML.

2.6.1 The Transportation Problem Example

The classic transportation problem is a model that finds the minimal cost of product shipment through a complete bipartite graph to satisfy the demand of customers and the supply from the plants is limited.

$$
\begin{array}{lll}
\textit{minimize} & \sum_{i\in U}\sum_{j\in V} c_{ij}x_{ij} & \\
\text{subject to} & \sum_{j\in V} x_{ij} = a_i & \forall_i \in U \\
& \sum_{i\in U} x_{ij} = b_j & \forall_j \in V \\
& x_{ij} \geq 0 & \forall_{(i,j)} \in U \times V
\end{array}
$$

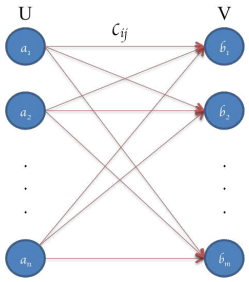

Figure 2.6.1: Hitchcock-Koopmans transportation model

Each element in U is called a supply point, and each element in V is called a demand point, where $U = \{1,2,...,n\}$ and $V = \{1,2,...,m\}$, as shown in Figure 2.6.1 and Figure 2.6.2. For example, the classic distribution of products from warehouses to resellers is as follows: the node set U is the warehouses, the node set V represents the resellers, and the edge $(i,j) \in U \times V$, represents a distribution link from warehouse i to the reseller j.

We exploit this model and demonstrate it as a detailed example of each SML's level. The parameters (for SML Level 2) are shown in Table 2.6.1.

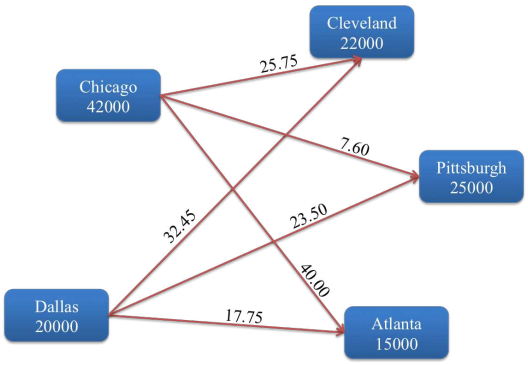

Figure 2.6.2: A simple 2x3 transportation model

	Pittsburgh	Atlanta	Cleveland	*Capacity*
Dallas	23.50	17.75	32.45	*20000*
Chicago	7.60	40.00	25.75	*42000*
Demand	*25000*	*15000*	*22000*	

Table 2.6.1: Classic transportation problem parameters

2.6.2 Spreadsheet Example: Simple Shipments and Freight Log

A shipment and freight spreadsheet example is a non-circular worksheet that can be used as a demonstration for Level 2 SML, even though Level 3 will be much more capable to handle this type of input. For the input (Figure 2.6.3 (a)), there are holding charges, unit-freight reduction, and a vendor's reduction, while the purchase price and the freight costs per unit are shown in Figure 2.6.3(b).

a

Monthly Carrying Charges	Freight reduction	Cost Reduction
3.50%	5.50%	12.00%

b

Original Conditions: Purchasing month to month			
Purchase Price	Freight Cost	sub-total	cost of ownership
$50.00	$4.50	$54.50	$56.41

c

Price & Freight Reduced 12.00% reduction				
Purchase Price	New Freight Cost	sub-total	month 1 ownership cost	Net Savings
$44.00	$4.25	$48.25	$49.94	$6.47

Figure 2.6.3: Simple shipments and freight log spreadsheet

2.6.3 The Inventory-Routing Problem (IRP)

The inventory-routing problem is an interesting problem because it is rather difficult to solve. It was initiated by Bell et.al.[9] from a practical standpoint at Air Products and Chemicals, Inc., in the early 80s. IRP is a combination of two problems: inventory management and vehicle routing. The aim is to meet the customers demand at the lowest possible cost, plus the additional concerns of related logistics costs. The IRP is to find the supply and distribution policies answering the following five basic questions:

(i) Which customers must be supplied that day?,

(ii) What vehicle will be used to supply each customer?,

(iii) What volume of the product is to be delivered?,

(iv) When should the supplier visit the customers?, and

(v) What delivery route will each vehicle take?

There are two types of contexts in IRP, customer managed inventory (CMI) and vendor managed inventory (VMI). To elaborate, CMI is a classic-type problem where customers have a fixed and known demand, while the distributors have no concern for inventory. And VMI is the context with a more modern focus: customers present random demand, and the distributor's stock is managed at the customer location. For inventory, there are two common predefined rules to replenish customers: the Maximum Level (ML) policy and the Order-up-to Level (OL) policy. In ML policy, the replenish level is flexible but bounded by the customer's available capacity. In OL policy, the delivered quantity will fill up the customer's inventory capacity. The "standard" version of IRP does not really exist [73], although, we can classify the IRP by seven criteria, namely: fleet composition, fleet size, inventory decisions, inventory policies, routing, structure, and time horizon.

The IRP is considered an important problem in the modern logistics value chain. To formulate the problem, the objective function is to minimize the sum of the inventory and transportation cost and the sum of two penalty terms related to stock-out and the vehicle capacity constraint. It is also an NP-hard problem; thus, heuristic methods are typically needed. Since searching for a solution can take a very long time, the search space can be defined by the following criteria: (a) no stock out at either suppliers or retailers; (b) the level of retailer's inventory is never greater than its maximum level; and (c) no violation of the vehicle capacity constraint. Appendix C presents the single customer IRP model.

2.7 Ontologies

An ontology is a specification of a conceptualization. The word "ontology" comes from the Greek *ontos*, for "being," and *logos*, for "word." Ontological engineering denotes a set of design principles, development processes and activities, supporting technologies, and systematic methodologies that facilitate ontology development and use throughout its life cycle: design, implementation, evaluation, validation, maintenance, deployment, mapping, integration, sharing, and reuse [32]. An ontology consists of a hierarchical description of important concepts (or classes) in a particular domain, along with the description of the properties of each concept. Sowa [98] defines the subject of ontology as the study of the categories of all the kinds of entities—abstract and concrete— that exist or may exist in some domain. An ontology should comprehensively capture the common understanding—a foundation in knowledge representation, storing and sharing—of a community as it applies to a specific domain.

Ontological engineering can be used to define a set of representational terms found in mathematical modeling. As the knowledge of a modeling domain is represented in a declarative formalism, the set of objects is called the "universe of discourse". Ontologies are often equated with taxonomic hierarchies of classes, class definitions; and the subsumption relation, but ontology need not be limited to these forms. Ontologies are also not limited to conservative definitions, that is, definitions in the traditional logic sense, that only introduce terminology and do not add any knowledge about the world [31]. To specify a conceptualization, one needs to state axioms that do constrain the possible interpretations for the defined terms.

The *types* in the ontology represent the predicates, word senses, or concept and relation

types by the perspective of a person who uses a language L for the purpose of discussing topics in the domain of interest D. An ontology embodies a formal naming and definition of the types, properties, and interrelationships between concepts in a particular domain. The process of developing an ontology often reveals inconsistent assumptions, beliefs, and practices within the community; including [33]:

- defining classes in ontology,
- arranging the classes in a taxonomic (subclass–superclass) hierarchy,
- defining slots and describing allowed values for these slots,
- filling in the values for slots for instances.

Ontologies provide an important role in the description of semantic of models organized in a corpus. An important aspect of ontologies is that the ontology languages such as the Web Ontology Language (OWL)[6] are related to a subset of First-Order Logic (FOL). The OWL 2 Direct Semantics[7] specify the semantics for the respective OWL profile in a model-theoretic way. New inferences can be gained based on the specified semantics by logic reasoning. The close connection to description logics provides a basis for reasoner implementations. Complex reasoning tasks on optimization models can be achieved with the aid of additional rules providing the possibility to retrieve important information for automation tasks. The reasoning procedures can also validate ontology definitions to avoid contradictions.

[6]OWL is a family of knowledge representation languages for authoring ontologies and is endorsed by the World Wide Web Consortium.

[7]Based on two semantics: OWL DL and OWL Lite semantics that is based on Description Logics.

Figure 2.7.1: Class Hierarchy of Top Optimization Modeling

"Meta" means one level of description higher. *Metadata* are data about data, and a *meta-model* is a model used to describe other models. We can infer that metadata is descriptions of data, and meta-models are descriptions used to characterize models. A metamodel is an explicit model of the constructs and rules needed to build specific models within a domain of interest. For example, Figure 2.7.1 characterizes a valid metamodel as an *ontology.*

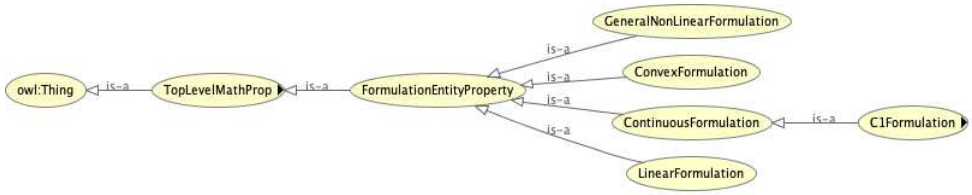

Figure 2.7.2: OWLViz of Optimization Problem Representation

Ontologies have been widely used in applications regarding knowledge representation and reasoning. Remarkable example applications exist in mathematical model management [81, 83, 10, 26, 32, 104, 7, 27, 28, 99], with a number of applications using ontologies to deal with the semantics of content within the model management system. Throughout this document, the OWLViz[8] of the asserted class hierarchy will be used to present our mathematical modeling ontology concept, such as the class hierarchy for model formulation shown in Figure 2.7.2. In the software engineering community, model-driven engineering (MDE) is being developed in parallel with the Semantic Web. The most significant research initiative in this area is the model-driven-architecture (MDA), which is being developed un-

[8]OWLViz enables class hierarchies in an OWL ontology to be viewed and incrementally navigated.

der the Object Management Group (OMG)[9]. MDE evolved as a paradigm shift from the object-oriented technology, in which the main principle is that "everything is an object"; into the model engineering paradigm, based on the principle that "everything is a model."

Mathematical Modeling Representation with Ontologies

In the context of mathematical modeling, an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. Its representational primitives are typically concepts (this research applied a structured modeling concept of classes), attributes (or properties), and relationships (or relation among class members). Ontologies have been applied to improve semantic information processing among OR analysts and computational systems.

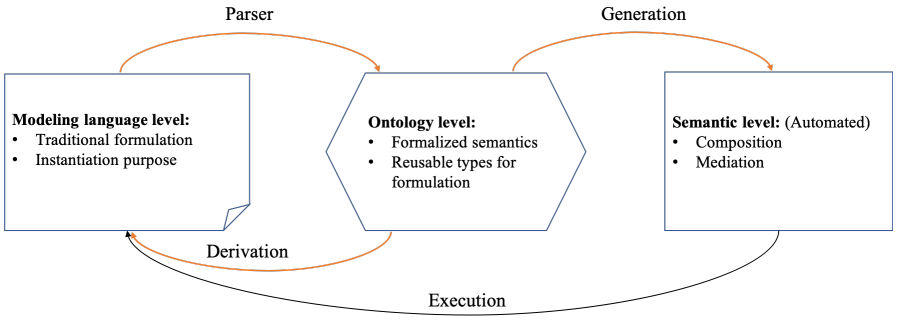

Figure 2.7.3: Applying Ontology Representation (Adapted from [99])

The model formulation process in the ontology representation will be part of the demonstration in Chapter 3. With a formulation of an abstract optimization model (including imprecise problem description) and further input describing the instance data, it might be

[9]The Object Management Group (OMG) is a computer industry standards consortium. OMG Task Forces develop enterprise integration standards for a range of technologies. Refer to www.omg.org

possible to retrieve a focused mathematical programming model. Figure 2.7.3 presents ontology representation as a layer in-between the algebraic modeling language level and the service level [99]. Thus, when composing models within this approach OR analysts may start with a preferred, supported AML and edit or reuse a model. With minimal effort, the AML parser allows bringing the model into an ontology representation.

2.8 Fuzzy Domains

Web Ontology Language Description Logic (OWL DL) becomes less suitable in domains in which the concepts to be represented do not have a precise definition. For instance, consider the case where we would like to build an ontology about flowers[10]. Then we may encounter the problem of representing concepts like "Candia is a creamy white rose with dark pink edges to the petals", "Jacaranda is a hot pink rose", "Calla is a very large, long white flower on thick stalks". As it becomes apparent such concepts hardly can be encoded into OWL as they involve fuzzy or vague concepts, like "creamy", "dark", "hot", "large" and "thick", for which a clear and precise definition is impossible. Figure 2.8.1 portrays the architecture of feedback fuzzy framework. Zadeh [110] introduced fuzzy theories in 1962 and cognitive uncertainties can be represented by his fuzzy set theory. Some key concepts are summarized in the following sections.

[10]Lecture note of Robert Fullér, What is fuzzy logic and fuzzy ontology?, KnowMobile National Workshop, Oct 30, 2008, Helsinki. Retrieved on March 4, 2010.

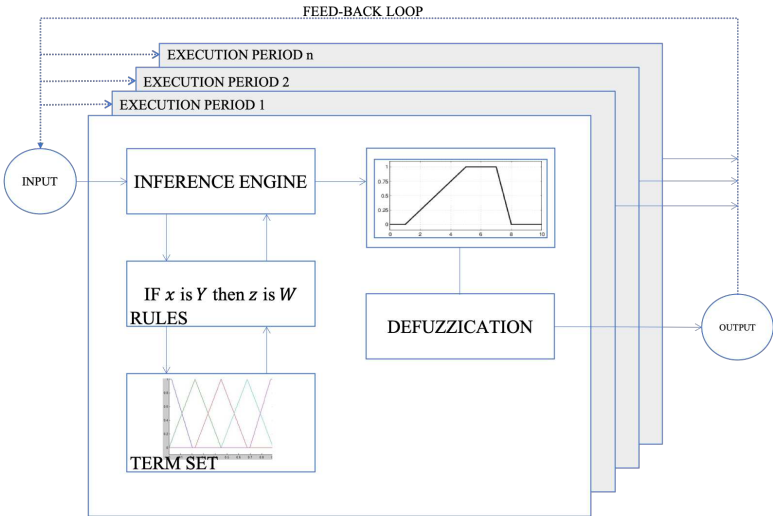

Figure 2.8.1: Fuzzy Framework [24]

2.8.1 Fuzzy Sets

Fuzzy logic starts with the concept of a fuzzy set. A fuzzy set is a set without a crisp, clearly defined boundary (see the flower discussion example above). It can contain elements with only a partial degree of membership. Zadeh [110] defined a fuzzy set as an extension of crisp sets, defined by elementary set theory. Any statement can be fuzzy. The major advantage that fuzzy reasoning offers is the ability to reply to a yes-no question with a not-quite-yes-or-no answer. As humans do this kind of thing all the time, as do OR analysts when they are working on problems, it is a rather new trick for computers in general. As shown in equation 2.8.1, a fuzzy set A associates a degree of membership for each object belonging to the universal set X. The membership function (MF), A, maps each $x \in X$ to its degree of membership in the fuzzy set A :

$$A : X \rightarrow [0,1] \tag{2.8.1}$$

A fuzzy set admits the possibility of partial membership in it, where the degree an object belongs to a fuzzy set is denoted by a membership value between 0 and 1. Here membership of degree 0 is interpreted as x not belonging to A and membership of degree 1 is interpreted as x fully belonging to A. A membership function associated with a given fuzzy set maps an input value to its appropriate membership value, MF, is usually represented by a curve that defines how each point in the input space is mapped by some degree of membership value. Intermediate values are interpreted as x having, in turn, intermediate membership in A. A fuzzy set A satisfying $sup_{x \in X} A(x) = 1$ is called normal. Intersections and unions on fuzzy sets are defined as:

$$\begin{aligned} (A \cap B)(x) &= min\{A(x), B(x)\} \\ (A \cup B)(x) &= max\{A(x), B(x)\} \end{aligned} \tag{2.8.2}$$

It is useful to define extensions to both intersections and unions on fuzzy sets. These operations are called the standard intersection, and standard union as defined in equations 2.8.2, noting that the Lukasiewicz's[11] operators use the sum for the union and the product for the intersection. As in Zadeh [110, 111], they corresponding to the most optimistic/most pessimistic of two given membership degrees. The cardinality measure of a fuzzy set A is defined by $M(A) = \sum_{u \in U} \mu_A(u)$, where U is a universe of discourse—a collection of objects denoted generically by u, and $M(A)$ is the measure of the size of A. The complement of a given degree x is defined as $1 - x$. Note that the intersection, union, and complement defined in the classic (crisp) set theory are sometimes referred to by their equivalent logical operators: AND, OR, and NOT, respectively.

[11] Łukasiewicz logic is a many-valued logic. Later generalized to $n - valued$ (for all finite n) as well as infinitely-many-valued ($\aleph 0 - valued$) variants, both propositional and first-order.

2.8.2 Fuzzy Ontologies

A fuzzy ontology is a quintuple $F =< I, C, T, N, X >$ where I is the set of individuals (objects), also called instances of the concepts. C is a set of fuzzy concepts (or classes – $cf.$ in OWL – of individuals, or categories, or types). Each concept is a fuzzy set on the domain of instances. The set of entities of the fuzzy ontology is defined by $E = C \cup I$. T denotes the fuzzy taxonomy relations among the set of concepts C. It organizes concepts into sup-(super-)concept tree structures. The taxonomic relationship $T\ (i, j)$ indicates that the child j is a conceptual specification of the parent i with a certain degree. N denotes the set of non-taxonomic fuzzy associative relationships that relate entities across tree structures, such as: *Naming relationships*, describing the names of concepts; *Functional relationships*, describing the functions (or properties) of concepts. Lastly, X is *the set of axioms* expressed in a proper logical language; i.e., predicates that constrain the meaning of concepts, individuals, relationships and functions.

2.8.3 Fuzzy Decision Tree

Decision trees are one of the most useful methods for learning and reasoning from instances. Although, due to the imprecision (to deal with language uncertainties) in decision-making processes, Watson et al. [105] proposed a fuzzy decision analysis concept that considered parameters that are only known approximately or which reflect subjective knowledge of the decision maker, by applying fuzzy-set theory. Their work addressed imprecision in decision analysis, especially when changes in multiple parameters might alter the recommended decision, while sensitivity analysis on individual parameters might not indicate variation. The Fuzzy Decision Tree (FDT) aims to combine the ability of a deci-

sion tree[12] with a fuzzy representation that dealing with inexact and uncertain information. There are several heuristic algorithms available in the literature for generating FDT.

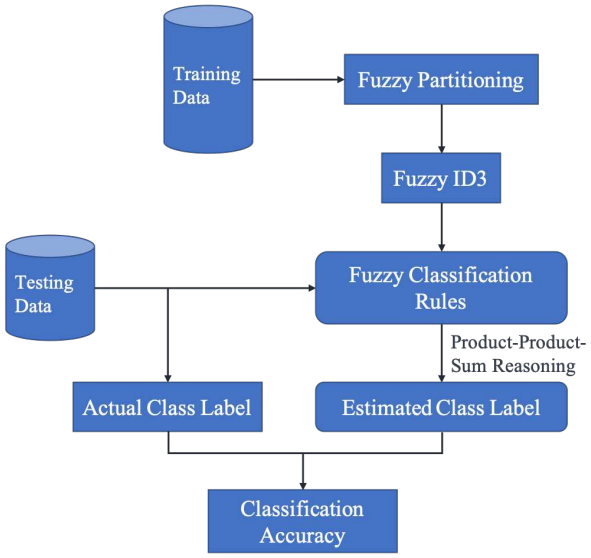

Figure 2.8.2: Architecture of Fuzzy Decision Tree Induction

The generation of a fuzzy decision tree consists of three major steps namely (i) fuzzy partitioning (clustering), (ii) induction of FDT, and (iii) fuzzy rule inference, as shown in Figure 2.8.2. FDT is constructed using any standard algorithm like Fuzzy ID3 (discussed below), where we follow a top-down approach, making locally optimal decisions at each node in a recursive manner with a divide and conquer approach.

Decision Trees

A decision tree is a simple, flowchart-like graph used to analyze decision making as well as to determine optimal decision paths for a given scenario. One such scenario occurs during model formulation, often a decision maker (referred to as OR analyst from this

[12]Decision trees can learn from examples, present knowledge in a comprehensible form.

point on) faces a scenario, where the OR analyst needs to make a series of decisions during a period of time, and where these decisions affect some final outcome. Outcomes and options for future decisions might also be affected by changes in the state of the world during this period. The OR analyst should then adjust her strategy based on information that is available at hand (at each time she makes a decision). A scenario like this could be considered a stochastic discrete-time dynamic optimization problem.

A natural representation for such a scenario is a tree with leaf nodes representing outcomes reached by a sequence of decisions and stochastic events. The tree represents a multi-step decision-making process where a chain of discrete decisions has discrete outcomes yielding some stochastic utility. The graph contains at least three kinds of nodes to represent these properties: decision nodes, chance nodes, and outcome nodes (leaves). Traditionally, decision trees have two major components: a procedure to build the symbolic tree and a method for decision making. In most cases, an OR analyst's decisions over choices in the tree is directed to maximizing expected utility[13] (information gain). One way to interpret the decision tree is as a dynamic optimization problem where a natural approach to solving for an optimal decision strategy is by dynamic programming.

In decision tree learning, one of the most used algorithms for constructing a decision trees has long been the ID3 (Iterative Dichotomizer 3) method introduced by Quinlan [91] in 1986. This algorithm tries to generate a decision tree from a dataset. The ID3 algorithm is described below. Later on, as an extension to Quinlan's ID3 algorithm, a statistical classifier algorithm C4.5 [91] used to generate a decision tree for classification was developed. Another method, CART [12] tries to split a compact interval into two subintervals and choosing the optimal split-point based on the training data.

[13] the Occam's razor principle

Algorithm 2.1 ID3

$(Examples, TargetAttribute, Attributes)$

create a root node for the tree, *Root*
$A \leftarrow$ the attribute with highest information gain
set decision tree attribute for *Root* $\leftarrow A$
for all $p_i \in A$ **do**
    add a new tree branch below *Root*, labeled with the value p_i
    let $Examples(p_i) \leftarrow$ the subset of examples that have the value p_i for A
    if $Examples(p_i) = \phi$ **then**
        add leaf node for this branch with label = the most common value in the examples
    else
        add as subtree for this branch the tree provided by ID3$(Examples(p_i), TargetAttribute, Attributes - \{A\})$
    end if
end for

Fuzzy expectation

For the simple case of a discrete random variable X with possible outcomes $x_1, x_2, ..., x_n$ and with respective utilities $u_i \in \mathbb{R}, \forall i$ and probabilities $p_i \in [0,1], \forall i$ so that $\sum_i p_i = 1$, the expected value is

$$\mathbb{E}[X] = \sum_{i=1}^{n} p_i u_i \tag{2.8.3}$$

For such a discrete random variable a crisp expected value is a mapping of $\mathbb{E}[\cdot] : \mathbb{R}^n \times [0,1]^n \rightarrow \mathbb{R}$.

Extending this definition to a fuzzy domain and range, let's consider a discrete random variable X whose possible outcomes yield fuzzy utilities corresponding to membership functions U_i and take place according to fuzzy probabilities corresponding to membership functions P_i. These fuzzy numbers are used to compute the expectation value $\mathbb{E}[X]$, due to U_i and P_i being fuzzy, $\mathbb{E}[X]$ is also fuzzy. Thus, let's denote this fuzzy expectation as

$\mathbb{E}[X](u), u \in \mathbb{R}$. We now have

$$\begin{aligned} \mathbb{E}[X](u) &= \sup_{u=\sum_{i=1}^{n} p_i u_i} \{U_1 \cap P_1 \cap ... U_n \cap P_n\} \\ &= \sup_{u=\sum_{i=1}^{n} p_i u_i} min\{U_1(u_1), P_1(p_1), ..., U_n(u_n), P_n(p_n)\} \end{aligned} \tag{2.8.4}$$

For simplicity, referring to the membership degree of an object to a certain set of attributes as the membership of the object to the corresponding tree node. The membership can be computed gradually and needs to use an intersection operator for fuzzy memberships. Considering that the attributes $A_1, A_2, ..., A_k$ were encountered from the root of the tree to a certain node (on level k), the membership degree of our object[14] is:

$$\bigcap_{i=1}^{k} \mu_{A_i}(v_i) \tag{2.8.5}$$

Depending on the semantics of the attribute, for testing symbolic value, we can either test it against numerical values or symbolic values (i.e., a 10 minute time period might be considered 40% long and 60% short). We can either assign a disjoint or a certain degree of similarity, as in the case of traditional decision trees. If we are testing a numerical value, we always have to deal with a condition against a cut point α. Yuan and Shaw [**?**] defined the $\alpha - cut$ of a fuzzy set A as

$$\mu_{A_\alpha}(a) = \begin{cases} \mu_A(\alpha), & \mu_A(a) \geq \alpha; \\ 0, & \mu_A(a) < \alpha. \end{cases} \tag{2.8.6}$$

[14] considering that the values for the corresponding attributes are $v_1, v_2, ..., v_k$

In this case, for each such attribute, we can introduce an extra parameter $\beta > 0$. Around each cut point, we say that there is an uncertainty interval of length β. Inside this interval, the condition $x < \alpha$ has a certain membership degree between 0 and 1. For each class, we compute the membership of our object. Since we have the corresponding value, as well as, a membership degree associated with it, we can compute the disjunction between the leaves that have a particular value. If the classes are not independent, we can compute the weighted average of the leaf values as

$$\frac{\sum_{x \in LeafNodes} value(x) \times \mu_x}{\sum_{x \in LeafNodes} \mu_x} \tag{2.8.7}$$

where $LeafNodes$ is the set of all leaves in the tree, $value(x)$ is the value corresponding to the leaf node x and μ_x is the membership degree of the given leaf.

Algorithm 2.2 Fuzzy ID3

$(FuzzyPartitionSpace, LeafSelectionThreshold(\beta_n), BestNodeSelection)$

While there exist candidate nodes
DO
Select one of them using a search strategy,
Generate its child-nodes according to an expanded attribute obtained
by the given heuristic.
Check child nodes for the leaf selection threshold.
Child-nodes meeting the leaf threshold has to be terminated as leaf-nodes.
The remaining child-nodes are regarded as new candidate nodes.
end

2.8.4 Fuzzy Inference Process

Fuzzy inference is the process of formulating the mapping from a given input to output using fuzzy logic. In general, the fuzzy inference process is composed of five parts: (i) fuzzification of the input variables, (ii) application of the fuzzy operator in the antecedent, (iii) implication from the antecedent to the consequent, (iv) aggregation of the consequents across the rules, and (v) defuzzification. The process of fuzzy inference involves all the pieces that are described in membership functions, logical operations, and if-then rules. Figure 2.8.3 shows the information flow of the fuzzy inference diagram that is composed of all the smaller diagrams.

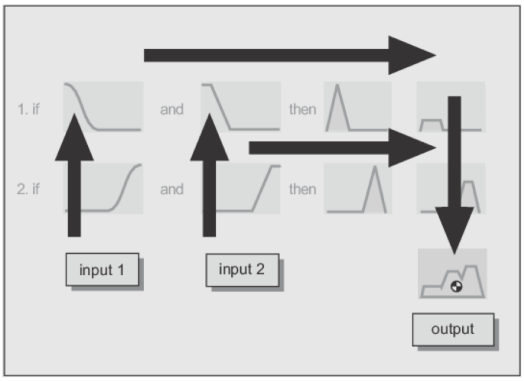

Figure 2.8.3: Fuzzy Inference Diagram (MathWorks Documentation)

(i) *Fuzzification of the input variables* amounts to either a table lookup or a function evaluation. The input is always a crisp numerical value limited to the universe of discourse of the input variable. The output is a fuzzy degree of membership in the qualifying linguistic set (in the interval from 0 through 1).

(ii) *Application of the fuzzy operator* If the antecedent of a rule has more than one part, the fuzzy operator is then applied to the output function, yielding a single truth value

as an output (format at least two membership values input of fuzzified variables).

(iii) *Implication method* requires the rule weight determination. Every rule has a weight that applies to the number given by the antecedent. The input for the implication step is a single number given by the antecedent, and the output is a fuzzy set.

(iv) *Aggregation of the consequents across the rules* must be combined in some manner. Aggregation is the process by which the fuzzy sets that represent the outputs of each rule are combined into a single fuzzy set. Aggregation only occurs once before the final defuzzification step.

(v) *Defuzzification* step is the final step of the fuzzy inference process. The input for defuzzification is an aggregate output fuzzy set and the output is a single number. There are five built-in defuzzification methods in MATLAB: centroid, bisector, middle of maximum, largest of maximum, and smallest of maximum. The most popular defuzzification method is the centroid calculation, returning the center of an area under the curve as shown in Figure 2.8.4 below.

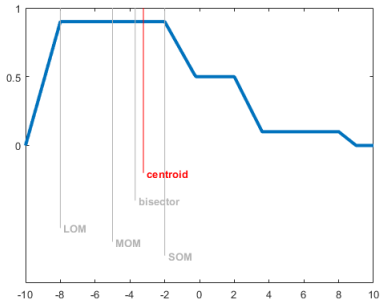

Figure 2.8.4: Defuzzify (based on MathWorks Documentation)

2.9 Complex Systems Modeling

One way to approach a complex problem is to understand the art of complex problems solving. It is crucial to recognize the big picture when considering complex systems, typically featuring Ill-Structured Problems (ISPs). Because tools and methods appropriate for such problems are different from those for well-defined problems. Dealing with ISPs requires systematically thinking about and foreseeing narrow technical details. Successful complex systems modeling requires different people with different perspectives relating to different parts of the systems.

2.9.1 The Art of Complex Problem Solving

Decision-makers often deal with problems or "manage some system" that is inherently complex, intractable, and uncertain, referred to as ISPs. To do this they must struggle to build a "mental model" of a system using various sources, e.g., casual conversations, conferences or meetings information, fragmentary data, news articles, written reports, personal observation, etc. It is expected that somehow, in their heads, individual managers will combine this information into a coherent and accurate picture of real-world problems and make wise decisions based on this information.

2.9.2 Visual Modeling

Visual models (VM) are uniquely effective tool for working with complex problems. They can explicitly show, in parallel, all major elements of a problem. Visual models provide a complementary representation to linear verbal language and represent visual-metaphors

one thinks about. VM provides a tangible representation that explicitly captures the complexity of a problem, yielding a concrete representation to work with. The key benefit of visual modeling is that it enables us to build an integrated and tangible model of a problem, greatly enriching a decision-making process while keeping that process grounded in a realistic representation of the current situation.

2.9.3 A Holistic Approach

To work with all aspects of the problem, integration of many options and coordination of all possible actions should be viewed as "wholes", not as a collection of parts. Consequently, treating the problem-solving process as a system and working with key elements leads to a holistic approach. The ultimate goal of this study is to promote coherent action on ISPs. In doing this, the related tasks are tied together by a number of related tasks. This will help us think more clearly, make better decisions, and implement more effectively by exploiting the unique capabilities of visual modeling, visual facilitation, and illustration to organize and advance this process.

Chapter 3

Model Supposition and Structured Corpus

> "Mathematical creativity consists in either recognizing that an existing formalism is applicable to the problem at hand or inventing a new one." – Kac and Ulam [65].

This chapter proposes two main elements for the optimization modeling corpus that constitutes a middle interface for the cognitive computing platform. First, the qualitative modeling techniques are discussed, along with a cross-paradigm technique that enhances soft system methodology for the problem described in the structured corpus. Second, the extension of structured modeling language (SML) as an additional level of SML, Level 5, which offers structured modeling with *contextual support*, involving an exhaustive data analysis and selecting the most important pieces of information for model building. Henceforth, OR modelers will no longer need to start from scratch or reinvent the 'formulation wheel' constantly, or read every possible journal articles in the field when formulating a problem.

3.1 Qualitative modeling techniques

Modeling typically means the construction of models which can be used for detection of insights or for the presentation of perceptions of systems. The "act" of modeling consists of selection and construction; workmanship; an analogy conclusion or other derivations on the model and its relationship to the real world; and preparation of the model for its use in systems (accounting future evolution and change). Wilson [107] presented a view of models and the distinction between models and modeling that:

> "Models (of any kind) are *not* descriptions of the real world they are descriptions of *ways of thinking* about the real world"

The theoretical framework must be based on a mathematical framework that allows proving properties and must be flexible for coping with various modeling methodologies. Definition 3.1 set forth eight conceptual modeling capacities. Among those capacities, some are used in parallel, and some may conflict with the others during implementation. Based on a skeleton account of SSM, one interacts with real-world situations by making judgments about them [93]: are they 'good' or 'bad', 'acceptable' or 'unacceptable', 'permanent' or 'transient'?

Definition 3.1 Eight conceptual modeling capacities

Every complex problem is ill defined in its own way. Conceptual modeling technique includes the following eight capacities [100]:

1. The model provides some understanding of the original;
2. The model provides an explanation of demonstration through auxiliary information and thus makes original subject easier or better to understand;
3. The model provides an indication and facilities for making properties viewable;
4. The model allows to provide variations and supports optimization;
5. The model supports verification of hypotheses within a limited scope;
6. The model supports construction of technical artifacts;
7. The model supports control of things in reality;
8. The model allows a replacement of things of reality and acts as a mediating means.

3.1.1 Ill-structured problems

Ill-structured (or wicked) problems demand problem-solving skills and abilities that differ from well-structured problems. These skills include critical thinking, incorporating multiple perspectives, and reasoning. Soft Systems Methodology is a constructive approach that centers on eliciting peoples' views of a process; capturing these views in conceptual models; using these models in a debate to clarify the problem situation [19]. It has been proven to be a suitable match to the nature of this type of problem and incorporate all those skills, as indicated by Checkland [14, 15, 16, 17, 18, 19].

Such problems tend to be vaguely defined, involving the modeling of human activities within a system [95], and resulting from the informal diagrams that are augmented with a linguistic component. We are interested in how to efficiently characterize the problem for diagnostic purposes. The possible information acquisition is: what questions should be asked?; what tests should be run?; what data can safely be ignored?; and at what point are

algorithms absolutely necessary to obtain desired solutions?

Examples of ill-structured problems can be found in complex adaptive systems. For instance, collaborative tagging in large-scale online systems, the ecosystem, human social group-based endeavor (i.e. political parties or communities), manufacturing businesses.

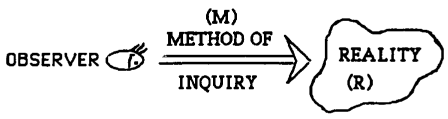

Figure 3.1.1: The big picture lenses

Figure 3.1.1 is adopted from Khisty [69], where the observer perceives a reality (R), using his methodology (M), through two main intellectual stances (hard and soft perspectives). The goals are to (1) formulate a model based on a prescriptive stance, and; (2) emphasize participation and learning. Since the problem structuring is based on the skeleton of Checkland's SSM, this chapter concerns the learning mechanism. It focuses on examining a current modeling technology and improving current methodology using a cross-paradigm approach and continues by providing illustrating examples and classification of ill-structured domains.

3.1.2 Classification of ill-structured domains

Example of ill-structured problems are problems that have more than one solution path and contain elements of uncertainty such as case-based problems, design problems, dilemmas, system analysis problems, and troubleshooting.

Data Retrieval	Document Retrieval	Model Retrieval
1. Direct (“I want to know X”)	Indirect (“I want to know about X”)	Investigate (“I want to find a model that explain X”)
2. Necessary relation between a formal query and the representation of a satisfactory answer.	Probabilistic relation between a formal query and the representation of a satisfactory answer	Satisfying relation between a formal query and the representation of a useful model that recognizes tradeoff between accuracy and complexity.
3. Criterion of success = correctness	Criterion of success = utility	Criterion of success = improved ability to predict, manipulate or understand X
4. Speed dependent on the time of physical access.	Speed dependent on the number of logical decisions	Speed dependent on the number of modifications required to obtain a model
5. Finite representation of information.	Unlimited ways to represent information	Unlimited but classifiable ways
6. Finite search space	Combinatorial large search space	Exhibits large search space

Table 3.1.1: Comparison of data, document, and model retrieval [11]

Case-based problems have been a prime interest in computer and artificial intelligence community for decades. Case-Based Reasoning (CBR) enacts a *problem-solving paradigm* where previous experiences are used to guide problem-solving. CBR mechanisms are Retrieve, Reuse, Revise, and Retain cases.

Design problems typically deal with an increasing amount and complexity of project information. Information-processing theorists are concerned with explicating the internal structures and procedures that cognitive systems use during the design activity. Eastman [30] called this 'intuitive design'.

Dilemmas refer to a situation where a difficult choice has to be made between alternatives. A dilemma problem may offer two possibilities, neither of which is practically acceptable. *System analysis* defines the problems to be solved and provides the architecture for the proposed system.

Troubleshooting is a form of problem-solving that involves a logical process of elimination to identify the true source of a problem. Troubleshooting is used in complex systems where the symptoms of a problem can have many possible causes.

3.2 Cross-Paradigm Technique

The concept of optimization that predominated in OR studies is difficult to achieve in messy situations as it is unlikely that there would be agreement as to what constitutes an optimum.

General Systems Theory (GST approach) provides a way of looking at problems with insights gleaned from knowledge about systems in general. GST would help us think more clearly about what our goals happen to be, and about what methods we should use to achieve them. The design process is conceived through the following sequence of steps:

(1) an exploration and decomposition of the problem;

(2) an identification of the interconnections between the components;

(3) the solution of the subproblems in isolation; and

(4) the combination of the partial solutions into a complete problem solution.

Cross-paradigm technique is concerned with the effectiveness (is what we are doing right?,), efficiency (is it using minimum resources?) and efficacy (does it work?). The results of these questions will be in the form of recommendations or policy.

Alternative Weltanschauung Representation (AWR)

Enhanced Soft Systems Methodology represents an extension of the formal nature of SSM models. There were three early enhancement attempts in the 1990s by: Gregory in Modal logic [57, 56, 58]; Minkowitz in Formal Process Modeling [80]; and Probert [89] in Logic and Conceptual Modeling. In 1998, Petri Nets (PN) were used to model weltanschauung alternative by Lamp [74] in the conceptual stage of SSM but no further analysis was discussed. In addition, Sagoo and Boardman [95] combined PN with behavioral analysis to the system of soft systems methodology.

To strengthen the analysis of SSM, I employ the *alternative weltanschauung representation (AWR)* using the structural analysis of a Petri Net (PN). Given that PNs are a tool for the study of systems, PN theory allows a system to be modeled by an AWR and use a mathematical representation of the system. Analysis of the PN can then, hopefully, reveal important information about the structure and dynamic behavior of ill-structured problems. From a multifaceted set of issues, I interpret the information contained in the SSM model, identify the agents, activities, and artifacts. I, then, apply the following proposition:

Proposition 3.1 Alternative Weltanschauung Representation (AWR)

A Petri Nets based worldview (weltanschauung) is called AWR, if and only if the following structure holds:

1. the AWR has two special structures:
   places $P = p_1, ..., p_n$ that represent the agents or artifacts, and
   transitions $T = t_1, ..., t_m$ that represent activities.
2. Considering the set of component worldviews, it is possible to combine the PNs, merging places with identical labels.

For each worldview in an SSM model, one can easily construct a special PN according to the proposition. Each AWR is composed of a separate PN. By merging places with identical label, agents (or artifacts) are represented by places, $P = \{p_1, ..., p_n\}$, and activities are represented by transitions, $T = t_1, ..., t_m$. This equivalent root definition in traditional SSM can be further analyzed by transforming AWR to an incident matrix C with n rows (places), and m columns (transitions). The column vectors $\underline{t}_i$ form the matrix C of AWR as

$$C = def(\underline{t}_1, ..., \underline{t}_l) = \begin{pmatrix} z_{11} & \cdots & z_{l1} \\ \vdots & & \vdots \\ z_{1k} & \cdots & z_{lk} \end{pmatrix}$$

Unlike the PN's behavioral analysis, the structural analysis does not depend on the initial marking of a PN. I am interested in the set of *minimal invariant*s. This information can then be used to evaluate the modeled problematic systems and suggests improvement or direction of changes.

Definition 3.2 Invariants

Given a graph (net) G with n places and m transitions, let C be its incidence matrix. A P-vector[a] $x \in \mathbb{N}^n$ with $x \neq 0$ is called:

1. Place invariants (P-Invariants)
   if $x^T.C = 0^T$;
2. Transition invariants (T-Invariants)
   if $C.y = 0$ where a T-vector[b] $y \in \mathbb{N}^m$ with $y \neq 0$.

[a] A P-vector can also be represented by a function $x : P \to \mathbb{N}$
[b] A T-vector can also be represented by a function $y : T \to \mathbb{N}$

To calculate invariants, I use the Farkas algorithm. The input to the Farkas algorithm is the juxtaposition $(C|E_n)$: the incident matrix C with n rows that denote places and m columns that denote transition, and the identity matrix E_n with $n \times n$ dimension. Martinez and Silva [72] have given a Gauss elimination like algorithm to calculate invariants:

Step1: Convert AWR to an incident matrix, and prepare the juxtaposition $D_0 := (C|E_n)$.

Step2: Nullify the i^{th} column of D_0 by adding any rows of D_0.

Step3: Iterate for $j = 1, 2, ..., m$ (stop when first m columns are nullified).

Step4: Take out the un-nullified columns (starting at the $(m+1)^{th}$ column, leaving first m nullified columns).

Step5: Rank of this resultant matrix is $(n-r)$. Thus $(n-r)$ linearly independent rows are minimal P-invariants.

This algorithm aims to find a minimal set of P-invariants (it can be applied to T-invariants by simply transposing the construction of the identity matrix). There is no guarantee that this algorithm will find only the minimal invariants. However, I am interested in the set of minimal invariants (W), whereby, its support does not contain the support of any other invariant Z, and the greatest common divisor of all non-zero entries of W is 1, ($supp(Z) \subset supp(W)$).

Symbols	Properties	Necessary & Sufficient Conditions
SB	Structurally Bounded	$\exists y > 0, Ay \leq 0$ (or $\nexists x > 0, A^T x \gneqq 0$)
CN	CoNservative	$\exists y > 0, Ay = 0$ (or $\nexists x, A^T x \gneqq 0$)
PCN	Partially CoNservative	$\nexists y \gneqq 0, Ay = 0$
RP	RePetitive	$\exists x > 0, A^T x \geq 0$
PRP	Partially RePetitive	$\exists x \gneqq 0, A^T x \geq 0$
CS	ConSistent	$\exists x > 0, A^T x = 0$ (or $\nexists y, Ay \gneqq 0$)
PCS	Partially ConSistent	$\exists x \gneqq 0, A^T x = 0$

Table 3.2.1: Necessary and sufficient conditions for some structural properties

The complete characterization of PN structural properties is summarized in Table 3.2.1. I interpret the expression $A^T x$ as the difference of markings in each place, and the expression Ay as the change in a weighted sum of tokens for each transition firing. Algorithm worst-case complexity is exponential due to a possible exponential number of matrix rows generated in the Farkas's solution process. To deal with this issue, I consider three robust implementations: (i) using a greedy strategy to pick columns; (ii) imposing a threshold on the number of rows to produce; and (iii) delivering an intermediate output of valid invariants.

3.3 Illustrative Examples

The aim of this section is to demonstrate problematic situations that use soft systems methodology. The example illustrates how to tackle the original *weltanschauung* and apply the ESSM to such situations.

3.3.1 Car rental example

Since the first car sharing (the Selbstfahrergenossenschaft (or self-drive cooperative) car-share program was initiated in Zürich in 1948) there has been no known formal development of such a system until the 1960s.

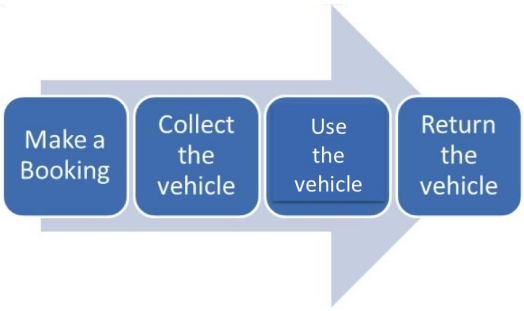

Figure 3.3.1: Basic car rental process

The basic questions that the car rental company asks their clients are: What time will the car be needed?, How long will the car be in use?, What type of car is preferred?, and Where would you like to pick up the car? ... to name a few.

Problem statements

A car rental is a vehicle that can be used temporarily for a specified rental fee. As shown in Figure 3.3.1, a customer (renter) who needs a car must contact a car rental company, then contract out for a vehicle. Most car rental companies offer optional insurance that can be arranged during a rental contract. Customers generally have options to choose amongst various car rental companies but limited options for the insurance companies.

Original 'Soft' approach

I consider this car rental example as a problematic situation in the car rental business. Referring to the §3.1, rich pictures are commonly used in problem expression. A rich picture is a non-judgmental approach for reviewing a situation or examining a system. To illustrate the worldviews, I have adopted the rich picture illustration from Edwards and Humphries as shown in Figure 3.3.2.[1]

As shown in the rich picture, the free form diagrams consist of the structure, process, issues, concerns, and development involved in systems of interest. I start my simplified example by observing three worldviews: (A) customer, (B) car rental company, and (C) insurance company as initiated in Lamp's work [74].

[1] Source: Lecture note of Prof. H. M. Edwards & Dr L. Humphries, University of Sunderland

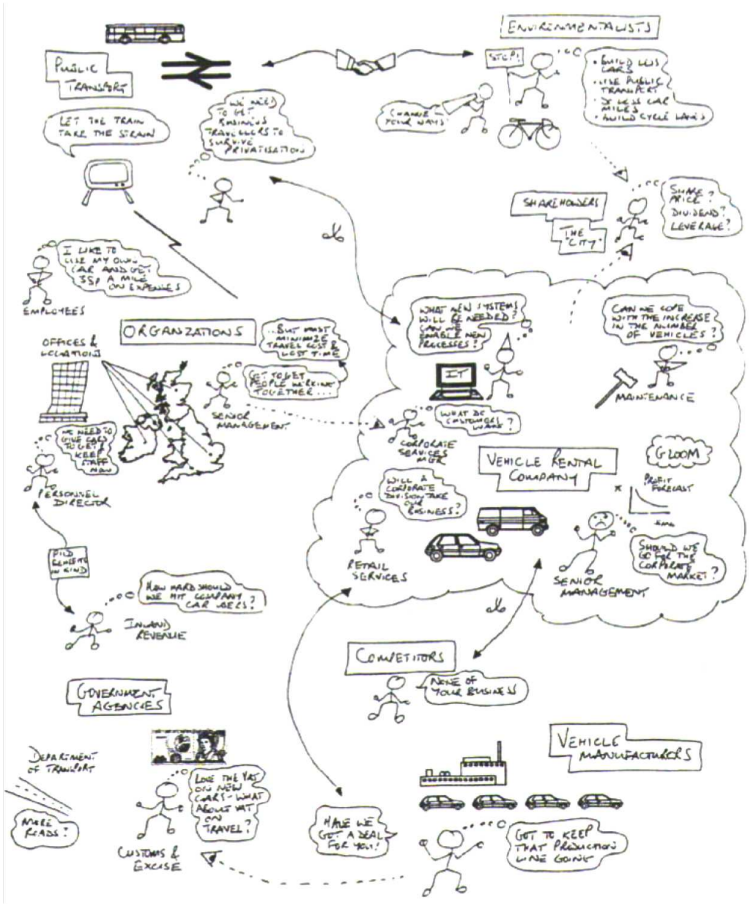

Figure 3.3.2: Rich picture of car rental example

	(A) Customer's	(B) Rental company's	(C) Insurance Company
Customers	Customers (casual and regular)		The car rental company
Actors	Renter company's employees		Insurance company's employees
Transformation	Satisfying customer needs for temporary use of a car		Providing choices of coverage
Worldview	Existing rental firms for a renter	Existing rentals firm	Identifying driver and risks
Owners	Manager of the rental station		The underwriters of the insurance company
Environment	Regulatory concerning registration and licensing		Motor vehicle and personal injury regulations

Table 3.3.1: Simplified car rental weltanschauung (worldview)

From CATWOE, it is easy to construct a Petri net in accordance with each worldview,

where the systems of interest can be split into three major categories: customer, car rental company, and insurance company.

Applied problem reframing

For illustration, we introduce Petri nets worldviews for car-rental workflow as:

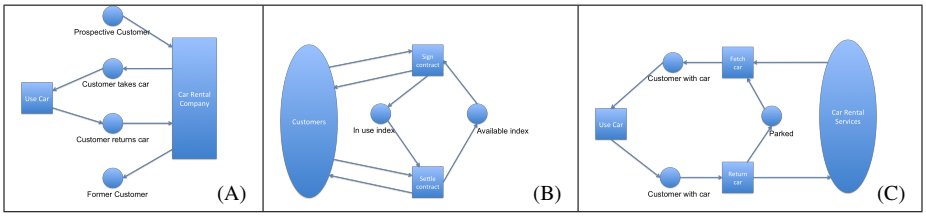

Figure 3.3.3: Using Petri Nets to model worldview alternatives [74]

An elementary system net for the car rental (or combined) worldview is constructed according to AWR in proposition 3.2. Figure 3.3.3 presents a separate Petri nets without subnet, according to three worldviews, shown in Table 3.3.1 where Places are *Prospective Customer, Cust. with a car, Cust. return a car, Former cust., Parked, Cust. with contract, Cust. to settle, In-use index, Available index*; and Transitions are *Use car, Fetch car, Return car, Sign contract, Settle Account*.

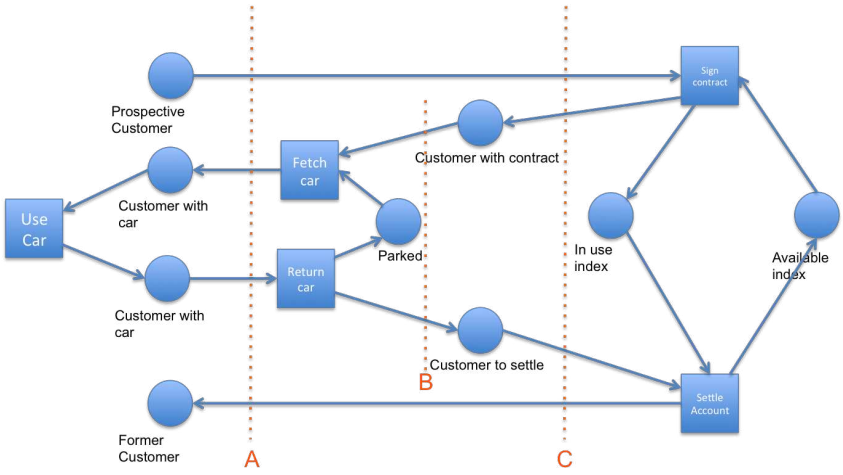

Figure 3.3.4: Graph N: for car rental example

In figure 3.3.4, the circles represent places and the squares represent transitions. Next, it is easy to impose an order of the form $t_1, ..., t_l$ on the transition of N, the column vectors $\underline{t_i}$ to form the matrix $\underline{N}$ of N.

$\underline{N}$	Use car	Fetch car	Return car	Sign contract	Settle Account
Prospective customer	0	0	0	1	0
Customer with car	-1	1	0	0	0
Customer return car	1	-1	0	0	0
Former customer	0	0	0	0	1
Parked	0	-1	1	0	0
Customer with contract	0	-1	0	1	0
Customer to settle	0	0	1	0	-1
In use index	0	0	0	1	-1
Available index	0	0	0	-1	1

Table 3.3.2: Transition matrix

Applied analysis of invariants

Finding place invariants by using the Farkas algorithm, I need the input from Table 3.3.2 as the identity matrix C juxtaposition with the identity matrix, $E_9 = I_{9\times 9}$; resulting in $D_{P_0} = [C|E_9]$, as follow:

$$D_{p_0} = \left[\begin{array}{ccccc|ccccccccc} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Note that place invariant matrix C consists of P rows where $P = (p_1, ..., p_9)^T$ and T columns where $T = (t_1, ..., t_5)$ of the matrix $\underline{N}$. Implementing five steps of the Farkas algorithm, I obtain the place invariants after three replications; however, after four replications, there is no valid T-invariant.

(1) **Place invariants** (**if** $x^T.C = 0^T$)

For $j = 3$ the Farkas algorithm stops, the results show two possible P-invariants,

$$x^T = \left[\begin{array}{ccccccccc} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] \left\{\begin{array}{l} Customer(\text{with car, return car}) \\ Index(\text{in use, available}) \end{array}\right. .$$

(2) **Transition invariants (T-Invariants)**

$$y = \left[\begin{array}{ccccccccc|ccccc} 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}\right]$$

There is no feasible T-invariant for this example. This indicates there exists no valid set of transition firings that can return to the original net N to the same marking. Translating to the nature of the car rental operations, where each customer requires unique services.

A P-invariant can be regarded as a token conservation component. The interpretation is ad-hoc to SSM models. In this particular example, I have two p-invariants. They point out that the car rental company must pay extra attention to car handling (both rent out and return). The example also indicates the importance of an internal index for keeping the business tracked. On the other hand, a T-invariant identifies a set of transition firings, which can return the net N to the same marking.

3.3.2 Problematic inventory situations

A small example in *real-life inventory* control, where inventory can be either finished products waiting to be sold or the raw materials used to produce the finished goods. Inventory is listed as a current asset on a company's balance sheet. In addition, inventory can be used as collateral to obtain financing at some point as shown in Figure 3.3.5[†].

At least five challenges have to be satisfied: (i) an anticipated increase in demand (meet demand and protect against unanticipated change); (ii) bulk ordering (to take advantage of

[†]From [64] Operations and supply chain management, the core, 3 Ed.

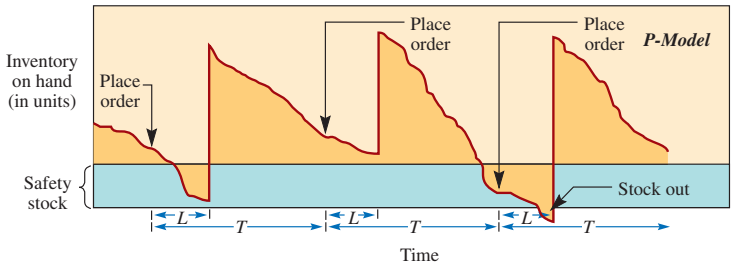

Figure 3.3.5: Typical Problematic Inventory Situations

price breaks); (iii) Prevent the production idling (in case of one part of the process breaks down); (iv) To keep a steady stream of material flowing to retailers (rather than making a single shipment of work in process (WIP) or goods); and (v) Time-based constraints (where geographical location may affect the transit time).

To satisfy the challenges, one may consider this inventory problem in many ways: (1) Decision Problem, (2) Inventory and Production Problem, and (3) Cost versus Revenue Considerations. Using a system-thinking viewpoint in decision analysis, with a Hard OR viewpoint, will break this problem into four parts:

- a *model*, a set of assumed empirical relations among a set of variables.
- a subset of *decision variables*, which are chosen by the firm (decision-maker).
- an *objective function*, such that a higher value (more profit) represents a more desirable state of affairs from the viewpoint of the firm. In some cases, a firm may shift its focuses to be more concerned for the viewpoint of its customers, where this may conflict with the initial profit-oriented view.
- *computing methods*, that examine, quantitatively, the effects of alternative values of the decision variables on the objective function. Ideally, such methods aim for an

optimal solution. Realistically, there may be a case where no optimal solution can be found. An analyst may have to turn toward a more descriptive solution.

The General Structure of Inventory Models[‡]

Inventory is one of the key components that drive the firm. Cost of goods sold (COGS) is a key driver of profit, total assets, and tax liability. A company incorporates inventory values to measure a certain aspect of its financial health using financial ratios such as inventory turnover. Major cost elements are the cost of ordering or producing, storage costs, discount rate, penalty costs, cost of change in the rate of production, and salvage costs.

In any inventory cost study, one may view this problem as one in 'production,' or as one in 'ordering.' In production, besides the inventory cost (as shown in Figure 3.3.6[§] below) accrued since the setup stage, there can be additional costs due to the preliminary labor and miscellaneous expenses of starting a production run. In ordering, the setup is due to the administrative expenses of processing the order.

[‡]Follow closely from Arrow Et. Al.[8]
[§]From [64]

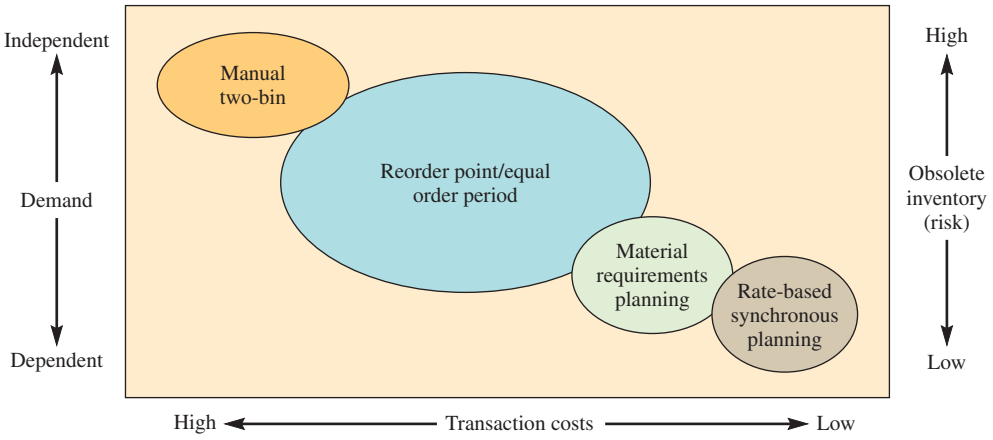

Figure 3.3.6: General Structure of Inventory Models

In stocking an inventory for any commodity, there will be a cost $c(z)$ to ordering or producing a given amount z of the commodity. A hard perspective relies on various assumptions about this cost function in different circumstances. For example, the most basic assumption is the cost of ordering is directly proportional to the amount ordered. A more complex cost function would suggest that a cost $c(z)$ is a concave function of z, meaning each additional unit will cost less. In some special cases, this cost $c(z)$ is composed of a cost proportional to the amount ordered plus some setup cost which may be a constant for z positive and zero for $z = 0$. Concave cost function also arises whenever there exist economies of scales (large scale production).

In contrast, a cost $c(z)$ can be a convex function of z. This will happen when additional output requires hiring additional staffs and purchasing additional equipment without increasing the size of the plant so that production becomes less efficient. Inventory cost functions are usually construct in ad-hoc fashioned, where a realistic cost function may be

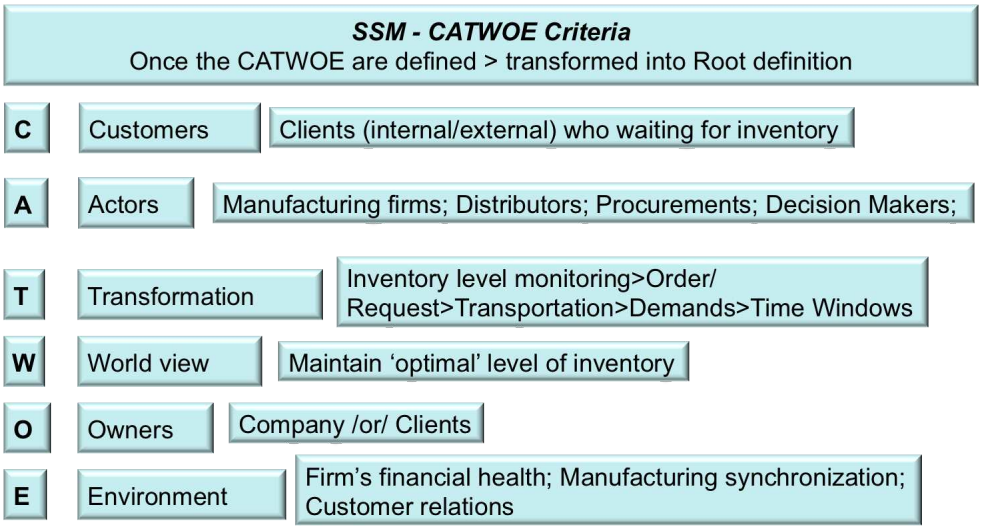

Figure 3.3.7: Applying SSM to a Problematical Inventory Control

made of pieces, each of which is either linear, concave, or convex. Because any changes in commodity and other material prices affect the value of a firm's inventory, it is important to understand how a firm accounts for its inventory. Common inventory accounting methods include first in, first out (FIFO), last in, first out (LIFO), and lower of cost or market (LCM). Some industries, such as the retail industry, tailor these methods to fit their specific circumstances.

In a broad sense, the time structure of inventory models can be either discrete or continuous. One treats the following two situations as a continuous variable [8]: (i) when it is assumed that demands occur as a continuous, or at least piecewise continuous function of time; (ii) when the demands come discontinuously and randomly at non-equal time points.

The basic requirement for counting an item in inventory is an economic control rather than physical possession. Therefore, when a company purchases inventory, the item is included in the purchaser's inventory even if the purchaser does not have physical possession of

inventories. Given the significant costs and benefits associated with it, Appendix D explains how a firm defines stock-flow identities and their assumptions used to calculate the optimal level of inventory should be at any given time. Changes in inventory levels can send mixed messages to investors.

3.4 Extension from Geoffrion's Structured Modeling

Due to the lack of methods for presenting the qualitative operations research knowledge that was mentioned earlier, an SML extension will be pursued as a vehicle summarizing and accumulating application experiences, and gaining knowledge on the model formulation. This section proposes a framework employing structured modeling archetype in a cognitive computing environment. We expect this extension to be groundbreaking as a decision analysis framework for the IBM $Watson^{tm}$ cognitive computing environment, or similar cognitive computing systems.

Fundamental framework of structured modeling

The Structured Modeling framework consists of three levels: (i) *elemental structure*, (ii) *generic structure*, and (iii) *modular structure*. An *elemental structure* is the most basic level of the model. It is defined as a non-empty, finite, closed[¶], and acyclic[‖] collection of elements to avoid the circulation of denotations. SM uses elemental structure to depict the real world. A collection of inter-related genera represents the second SM's level. A

[¶]A closed element: "A collection of elements is closed if, for every element in the collection, all elements in the calling sequence of that element are also in the collection."

[‖]Acyclic: "A closed collection of elements is acyclic if there is no sequence $E_1, ..., E_n$ such that E_1 calls $, ..., E_{n-1}$ calls E_n, where $n > 1$ and $E_n = E_1$".

generic structure is a generalization of the elemental structure. Genera may be organized conceptual units and be referred to as '*modules*'. These modules enabled the modeler to view the model at a different level of complexity; hence, this is known as the *modular structure.* In addition, a strict partial order called '*monotone-order*' is also introduced in this structure [90]. *A rooted tree* is a prime example of the graphical representation of the modular structure, where the root embodies the entire model and each terminal node is a genus. Recalled, the typical appearance of genus paragraph in SML is as shown in Figure 3.4.1 [46].

```
GNAME  [new index][(generic calling sequence)] /type/
       [index set statement][domain statement]
       [range statement][generic rule statement]
       ~| [interpretation].
```

Figure 3.4.1: Typical Genus Paragraph

- GNAME is a genus name beginning with *an upper case letter* or *dollar sign*.
- */type/* is a genus declaration, indicated by six reserved words according to six SM types.
- a new index is an option only used by self-indexed genera. It specifies a single lower case letter to serve as a symbolic index, together with any desired alias indices.
- a domain statement is an optional specification of the data type for the identifiers (names) used for the individual elements of a self-indexed genus.
- a range statement for an attribute genus specifies a set of permissible values for its elements (begin with a colon). Range type options are reals, integers, logical, strings and an enumerated list of strings.

- a generic rule statement for a function (or test) genus specifies all the value-determination rules for the elements of the genus (begin with a semicolon).
- interpretation is an optional plain natural language explanation of a typical element of the genus (or entire collection of elements) (it follow the format $\sim$ |).

The grammar of SML is clearly defined and flexible for mapping algebraic declarative modeling languages that are often used in the optimization paradigm. Indices are the basis for a generic name used to denote a typical element. A specific index tuple is obtained by giving specific identifier-values to all indices of a generic index tuple. For indexed genera, generic names generate element names when a particular index tuple is used as a suffix instead of a generic index tuple.

Example Consider transportation data (Section 2.6.1). The following examples show how the Level 4 SML grammar outlines the problem.

Dense TRANSPORTATION link:

```
LINK (PLANTi, CUSTj) /ce/        There is a transportation
LINK from each PLANT to each CUSTOMER.
```

Sparse TRANSPORTATION link

```
LINK (PLANTi, CUSTj) /ce/ Select  There is a transportation LINK
from some PLANTS to some CUSTOMERS.
```

Geoffrion foresaw three main SM design challenges: (i) a general conceptual framework for thinking about models; (ii) an executable language based on this framework, and (iii) software integration. Geoffrion [41] highlights desirable features for a new generation of modeling systems that can be summarized as:

(a) a rigorous conceptual framework for modeling based on a single model representation that provides sufficient generality to encompass most of the great modeling paradigms;

(b) independence of model representation and model solution;

(c) representational independence of general model structure and the detailed data, and

(d) a framework that will be useful for the entire model life-cycle.

A completely satisfied structured model requires explicit enumeration and specification of the following [43]: (i) all elements in detail; (ii) including all calling sequences; (iii) attribute values, function, and test element rules (but not values); (iv) generic structure satisfying similarity; and (v) a monotone-ordered modular structure. Otherwise, a structured model is said to be incompletely specified.

Benefiting from this rigorous definition of the semantics of the Structured Modeling's framework and representation, it is an ambitious undertaking to create an additional level to original SML – Level 5 Contextual and Adaptive Support. In the following sections, among the key points, we answer the following questions:

(i) What are the features of Level 5?

(ii) In what ways can cognitive computing handle the analytics (optimization tasks)?

(iii) What are the difficulties in implementing a Level 5?

To answer the first question, our proposed Level 5 SML incorporates model adequacy for model reuse and diagnosis purposes (search, discovery, and navigation through the opti-

mization model corpus**). We design Level 5 SML as an extension to the original four-levels of SML such that it can communicate with a cognitive computing type system such as IBM $Watson^{tm}$. Level 5 SML also utilizes ontology engineering concept.

3.5 Structured Modeling with Contextual and Adaptive

Our approach extends the work of Geoffrion [42, 44, 45, 51] to an adaptive exposition. We adopt the fundamentals of Structured Modeling concepts and Structured Modeling Language's notation as a paradigm-neutral model representation language. Level 5 SML is a stepping stone toward a cognitive analytic. We first focus on optimization models as a corpus for an optimization advisor.

The unique properties of our Level 5 SML is matched with Watson Explorer architecture in three ways: (i) Position-based index ("schema-less" properties), (ii) Hashing for recurrent genus type, and (iii) Self-synchronizing replication. Level 5 SML is designed to expand and broaden this framework. It will also be enabling a cognitive ability to manage and utilize the analytic corpus. The objectives of Level 5 SML are as follow:

1 to have a scheme that is well suited to the structured model instance;

2 to have a scheme that allows an easy transition to and from conventional models;

3 to have a scheme that a cognitive computer can process without undue difficulty for handling most models and their instance detail.

**In IBM $Watson^{tm}$, the corpus is a centralize content that store in a central repository. It uses to respond to user questions.

Level 5 SML: The Structural Modeling with Contextual and Adaptive Support

The structured modeling framework uses a hierarchically organized, partitioned, and attributed acyclic graph to represent the semantic as well as the mathematical structure of a model [45]. The general appearance of Level 5 SML is as shown in Figure 3.5.1.

```
Field-level updates
        Default Index Set
        Consistent hashing (new feature)
Authentication
        Asynchronous parsing agents
Generic Calling Sequence
        Floating Index Replacement Option
Expression
        Primitives
                Key-value stores
                Simple indexed variable
                        Offset Indexing
                @EXISTING expression (new feature)
                Matched description
        Functions
                Functional Dependency Ordinate Functions
                        Offset Indexing
                @EXISTING Index-supporting Functions (new feature)
                        Floating Local Index Range
                        Default Operand Value
Index Set Statement (ISS) for Externally Indexed Genera
        Relational Expression
                Generalized Index Set
                        Symbolic Index Tuple
                Set Operators
        Qualifier Sentence (supporting unstructured commentator)
```

Figure 3.5.1: Level 5 SML Features

Level 5 SML allows the use of position-based index as done in previously defined SML models. Recurrent genus types are imprinted with a hashtag. Level 5 is in compliance with Structured Modeling Markup Language (SMML), which is based on XML schema and an adaptive version of semantic (SA-SMML) The key requirement is asynchronous parsing support.

Consider the second question, “in what ways can cognitive computing be handled the analytic (optimization tasks)?” There are three ways to handle the analysis: formative, expan-

sive, and cognitive. For example, IBM uses Watson analytics as an automated service for data preparation, refinement, and analysis. We illustrate by example, the guided problem discovery wizard, in Chapter 4, where the complex situation feeds into the pipeline of our optimization problem reformulation using a cognitive technological framework.

3.6 Structure of the Ontology Representation

Adapted from [99] and references therein, the basic structure of the ontology representation is presented in Figure 3.6.1 as a simplified aggregated view of the approach. Our main differences from [99] are associated with the use of Geoffrion's structured modeling scheme, applied to the interface of meta-domain ontologies and concrete-domain ontologies. Moreover, there exists a modification due to the introduction of fuzzy representation in both quantities and fuzzy relation of the guided problems-discovery-wizard.

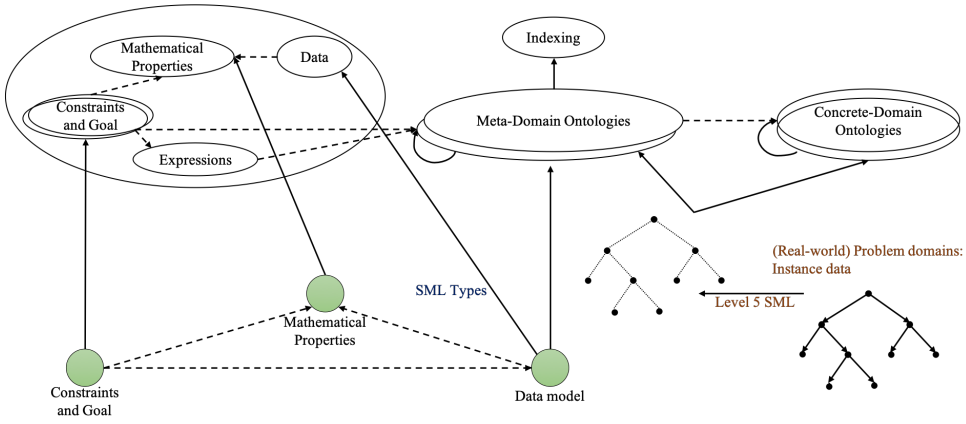

Figure 3.6.1: The Ontology Representation Structure

The ontology representation is more abstract than AML. The idea behind the approach aims to reuse the formulation types with different data conceptualizations and slightly varying

expression parts. Abstract optimization models are to be represented in an ontology graph structure. Thus, any manipulations on a model specification will be manipulations of the knowledge graphs given by the model as well. To summarize this representation structure:

Optimization modeling Mimicking an OR analyst's thought process in building an AML, the conceptualization of data and variables used in a constraint are specified before the actual formulation of an abstract AML. Therefore, in ontologies, goals, and constraints are grouped for a certain "very abstract domain" such as (in this example), "network", hence, the data ontologies conceptualization now only focuses on subdomain "network" too.

Meta Domain Ontologies (MDO) As we can see, meta-domain ontology types are also sub-types of indexing (general concepts for sets, parameters, and variables). From application domains, types for formulation entities, the data conceptualization for sets, parameters, and variables are to be defined as semantics types and referred to as MDO.

Concrete Domain Ontologies (CDO) We need concrete model instantiation services, which are derived from the structured modeling domain. Since data-related ontology classes are to be used to support the generation of data transformations that are referred to as CDO.

In the following section, a visualization of the constraint-type usage and the graph structure on the ontology assertion level for this example is presented in Figure 3.7.1.

3.7 Illustrations

This section presents toy examples of model (re)formulation using my framework. Specifically, the formalization and model formulation that is associated with the network flow domain.

Optimization Models with Ontology-based Representations

The critique system for model formulation needs an interaction between modelers (OR analyst) and the critique system or the guided problems-discovery wizard (GPW). Ontology is utilized to represent models semantics for data transformation generation and service composition in the GPW. A process, described here, leads to a life coach for an OR analyst to sort through an abstract optimization model to discover core components. GPW works through an integrated and automated treatment of changes, leading to a reformulation. First (Refer to Figure 2.7.3 in Section 2.7), an initial abstract optimization model is formulated in a preferred format of the OR analyst in AML format. Then, the system works through the AML Parser to obtain an ontology-based formulation. The ontology representation supports different tasks such as specifying model formulation corresponding to an AML representation as well as all kind of semantics manipulation.

3.7.1 A Common Min-Cost-Flow Example

The first example model is a common min-cost-flow problem (MCFP) (with integrality requirement). The reason we choose it as the toy example is because MCFP is a general structure in network flow models that provides a unified approach to many applications,

such as assignment, maximum flow, transportation, and transshipment problems. Furthermore, the possibility now represents itself for benchmarking our modeling with a similar research scheme by Stapel [99] on the last phase of my research. MCFP is also a reasonable generalization of the application of a fuzzy domain within my framework. Here are two facets associated with fuzziness of the problem characteristics.

Impreciseness and Uncertainty. Real-world problems include uncertain (and imprecise) data, the cost and the capacities of the network can be vague or uncertain. These uncertainties can be (ideally) captured by applying fuzzy set theory.

Multiple objectives. Multi-objective (and multiple hierarchy levels) models have a crucial role in mathematical programming and applications. A fuzzy decision tree representation, together with the use of linguistic statements as target values appearing as membership functions of fuzzy sets in the problem formulation, can reduce the complexity of such a problem.

There is a different (and equivalent) formulation for a minimum cost flow, such as find the maximum flow at minimum cost, or, send x units of flow from s to t as cheaply as possible. Consider, a graph $G = (V,E), V = \{1,2,...,n\}$ and let there exists non-negative edge capacities u, edge costs c, supply/demand b on each vertex, a flow of $f(v,w)$ on each edge (v,w). This problem has the following model representation:

$$
\begin{aligned}
&\min \sum_{(v,w)\in E} c(v,w) f(v,w) \\
&\text{subject to} \\
&f(v,w) \leq u(v,w) \quad \forall (v,w) \in E \\
&\sum_{w\in V} f(v,w) - \sum_{w\in V} f(w,v) = b(v) \quad \forall v \in V \\
&f(v,w) \geq 0 \quad \forall (v,w) \in E
\end{aligned}
\tag{3.7.1}
$$

Underlying assumptions are that cost and capacities are integral. Additional assumptions are: (i) graph is directed, (ii) if $(v,w) \in E$, then $(w,v) \notin E$, (iii) $\sum_b b(v) = 0$, and there exists a directed path of infinite capacity between each pair of nodes.

3.7.2 Input Examples of MCFP formulation in various AML

Table 3.7.1 shows different mathematical programming languages that an OR analyst might use for model input, where the general version of the min-cost-flow formulation is shown in Equation 3.7.1. Note that this demonstration is intended to use an AMPL formulation and the whole model is abstracted from concrete instance data. The goal is to minimize the overall linear flow cost. Parameters are given as follows:

- *Supply/Demand*: Observing supply parameters at nodes which are defined with positive values for inflows/supplies, negative values for demands and a zero value in sum (if a node is a transshipment node).

- *Cost*: There are linear flow cost parameter on the arcs plus and an upper bound on the capacity for the throughput of a single commodity (for simplicity of the example, at the moment).

- *ArcFlow*: This is the flow variables, enforcing zero as a lower bound. The arc-flows are to be determined in the solutions.

In our first example, the network balance forces the inflows and outflows at nodes to be equal. The typical network problem *MassBalance* constraints and the *UpperCapacityConstraints* are shown in Table 3.7.1. I believe that this simple MCFP suffices to allow a discussion of the basic ideas of the ontology representation approach and fuzzy domain ideas.

3.7.3 Ontology Specification for the MCFP

Example of a Constraint-Type Usage In a typical balance constraint of a network problem as shown in a target AMPL statement in Equation 3.7.2 can be represented by the ontology excerpts as shown in Figure 3.7.1.

$$\begin{aligned} &S.t.\ MassBalance\{i\ in\ Nodes\}: \\ sum\{(i,j)\ in\ Arcs\}(ArcFlow[i,j]) - sum\{(i,j)\ in\ Arcs\}(ArcFlow[j,i]) &\qquad (3.7.2) \\ &= SupplyDemand[i];. \end{aligned}$$

The identifier of parameters, sets, constraint, and variables in the AMPL model (Equation 3.7.2) are different to the suffixes of individuals in the ontology due to the Optimization Modeling (`OM`) prefix (`OM#EName`), used to call and annotate string names. The respective ontology classes are based on a meta-domain ontology for networks that carries the prefix `Net`. The `SingleCommodityBalance` type represents flow conservation equations for a single commodity with a single flow variable. Next, we shift to the summaries of the

AMPL	GAMS	Pyomo
set Nodes;	Set nodes /.../;	edges=[...]
set Arcs within {Nodes, Nodes}	Set midnodes(nodes) / /;	cost={.... }
	Set lastnodes(nodes) / /;	capacity={....}
param Cost{Arcs};	Alias(nodes, nodefrom, nodeto, n)	
param UpperCapacity{Arcs};	Set edges(nodes,nodes)	
param SupplyDemand{Nodes} integer;		
#Follow the convention >0 for supply and <0 for a demand	Parameter cost(nodes,nodes) /..../;	
var ArcFlow{Arcs}>=0, integer;	Parameter capacity(nodes,nodes) /..../;	
#In standard formulation, Zero lower bounds of flows		
#Integrality requirement for the demonstration!		
minimize FlowCost: sum{(i,j) in Arcs}(Cost[i,j]*ArcFlow[i,j]);	Model mincostflow / all /;	mcf=ConcreteModel()
s.t. MassBalance{i in Nodes}:sum{(i,j) in Arcs}(ArcFlow[i,j]-sim{(j,i) in Arcs}	...	mcf.uf=Constraint(
(ArcFlow[j,i]=SupplyDemand[i];		expr=sum(mcf.flow[e]
s.t. UpperCapacityConstraints{(i,j) in Arcs}: ArcFlow[i,j]<=UpperCapacity[i,j];		for e in edges if e.....)

Table 3.7.1: Various Inputs Example for MCFP

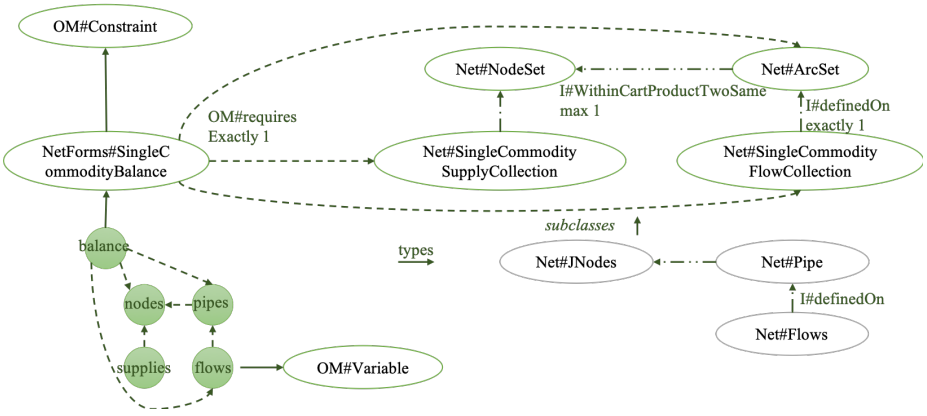

Figure 3.7.1: Ontology Excerpts of a Standard Balance Constraint Type

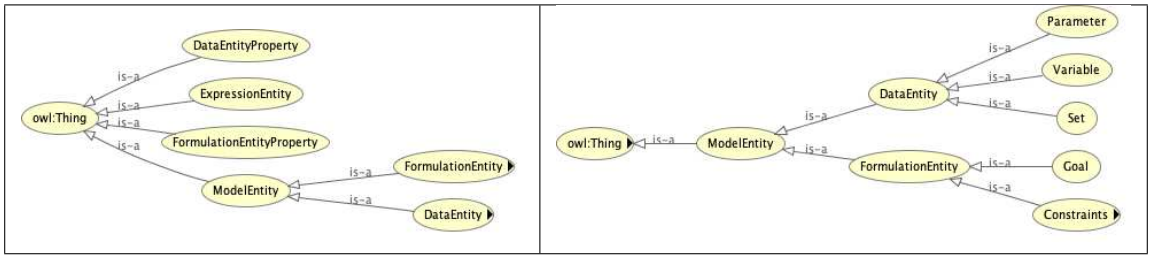

Figure 3.7.2: Top Level Optimization Modeling Definitions

top level definitions in `OM`, including `FormulationEntity` and `DataEntity` subclasses that use fundamental vocabularies for optimization model specifications. The top-level definitions in optimization modeling are the backbone of the ontology.

`ModelEntity` provides a class for all entities occurring in an abstract optimization model for instance (on the right-hand side of Figure 3.7.2) parameters, variables, sets, goals, and constraints. A complete description of the `OM`-ontology can be found in Appendix E. The mathematical properties of a general formulation such as linearity, positivity, integrality, and so on, are to be specified when a specific optimization model is defined in the ontological representation. Figure 3.7.3 provides a visualization of mathematical parameters.

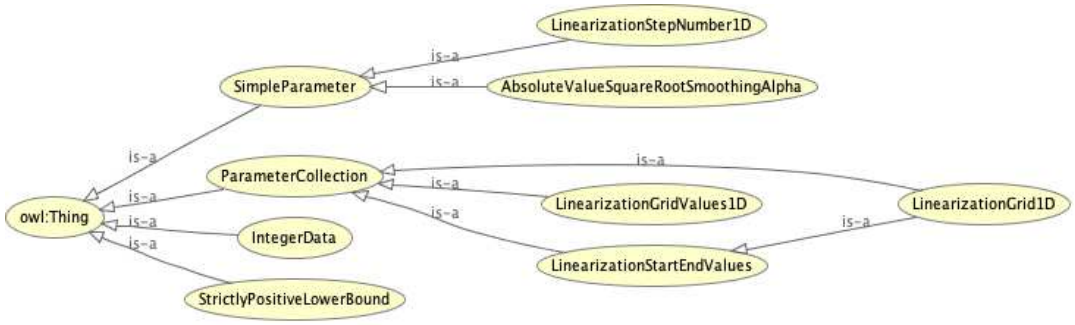

Figure 3.7.3: OWLViz of Mathematical Parameters Collections

Ontology Specification for the MCFP There is an interplay of different vocabularies when we observe Figure 3.6.1 in Section 3.6. Until now, we haven't specified the complete MCFP ontology representation, as yet. The model will now be specified using the ontology representation. We will further investigate the AMPL excerpts of the respective vocabularies in use, as well as, the explicit type definitions for the model. Figure 3.7.4 portrays the min-cost-flow instance structure in the ontology representation. The model consists of three formulation entities, six data entities, and some mathematical properties that are imported from ontology modeling properties (`OM.MProp`). Here, the AMPL statement to be derived from the usage of `NetForms#SingleCommodityFlowBoundsUp` as specified in the ontology graph is shown in Equation 3.7.3 below.

$$\begin{aligned} &S.t.\ UpperCapacityConstraints\{(i,j)\ in\ Arcs\}: \\ &\qquad ArcFlow[i,j] \leq UpperCapacity[j,i]; \end{aligned} \tag{3.7.3}$$

The ontology structure for MCFP above shows some mathematical property individuals. These properties are specified for the model entities by axioms. AML statement (excerpt ≥ 0 and integer, Equation 3.7.4) below reflects the two mathematical properties namely,

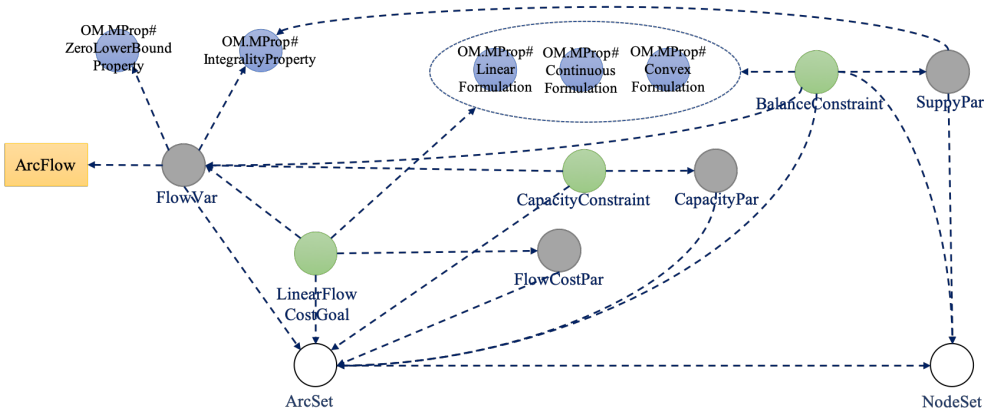

Figure 3.7.4: MCFP Instance Structure (Adapted from [99]) in the Ontology Representation

`OM.MProps#ZeroLowerBound` and `OM.MProps#IntegralityProperty` from the ontology modeling property.

$$var\ ArcFlow\{Arcs\} \geq 0,\ integer; \tag{3.7.4}$$

Notice that this integrality property (`IntegralityProperty`) of type `OM.MProps#IntegerData` is also used by `SupplyPar` which also reflects an integrality requirement for the input supply parameters. The underlying explanation here is that mathematical property individuals can be shared by different model entities.

This research considers a fuzzy logic system where data model and real-world problem domain instance data can be defined as the nonlinear mapping of an input data set to a scalar output data. OR analysts often use the words contain expressions of frequency, such as "always," "very often," "sometimes," "seldom," "never" during model formulation. In the following section, the discussion shifted toward the process of quantifying various form of uncertainty, such as ambiguity, imprecision, or vagueness occur naturally in reusing

model components, considering the representation of these imprecise data as fuzzy sets by making a crisp quantity fuzzy called fuzzification process.

3.8 Fuzzification

For fuzzification of crisp arithmetics and ordering methods, the computation of crisp expected value was fuzzified by fuzzifying the underlying arithmetic operations. In a case of the fuzzy decision trees, this is adequate since the probability distribution for each event in the scenario is discrete and infinite. Although, there exists no consensus as to how fuzzy quantities should be ordered, leaving it up to the OR analyst to decide how to discern order between fuzzy utilities. The following section presents an example depicting the fuzzification process.

Strength of Confirmation

Table 3.8.1 presents an example of the linguistic values, intervals and representative values expressing frequency and strength of confirmation for an ambiguity domain. Note that Φ means undetermined.

One way to handle the modeling diagnosis is to analyze the ambiguity that is left on. As discussed in Sub section 3.7.3 (Figure 3.6.1), an important point is how to handle the ambiguity that arises within the proposed SML Level 5 interpretation. Normally, an OR analyst conducts a through research in the domain of modeling interest. However, research into the acceptability of the information ambiguity is the key to robust modeling guidance. Let us consider the following four items as a basis for discussion example: *S***:** symptoms, test

Linguistic Value (Strength of Confirmation $m()$)	Interval	Representative Value
always	[1,1]	1
almost always	[0.98,0.99]	0.99
very often	[0.83,0.97]	0.9
often	[0.68,0.82]	0.75
sometimes	[0.33,0.67]	0.5
seldom	[0.18,0.32]	0.25
very seldom	[0.03,0.17]	0.1
almost never	[0.01,0.02]	0.01
never	[0,0]	0
unknown	Φ	Φ

Table 3.8.1: Linguistic Values, Intervals, and Representative Values Examples

results, (or indicators); *D***:** name of the diagnosis; *SC***:** a combination of symptoms; *IC***:** any other internal combinations; and T**:** fuzzy score, determined by $T = [0,1] \cup \Phi$.

Each symptom S_i is characterized by μ_s which have value in the fuzzy scores T. However unlike standard membership functions (discussed earlier in Section 2.7), our score also may takes the value Φ. The intersection, union, and complement of the fuzzy set are defined below:

$$
\begin{aligned}
X_1 \cup X_2 &= \begin{cases} min\{x_1, x_2\} & if X_1 \in [0,1] \quad and \quad X_2 \in [0,1] \\ \Phi & if X_1 = \Phi \quad and \quad X_2 = \Phi \end{cases} \\
X_1 \cap X_2 &= \begin{cases} max\{x_1, x_2\} & if X_1 \in [0,1] \quad and \quad X_2 \in [0,1] \\ X_1 & if X_1 \in [0,1] \quad and \quad X_2 = \Phi \\ X_2 & if X_1 = \Phi \quad and \quad X_2 \in [0,1] \\ \Phi & if X_1 = \Phi \quad and \quad X_2 = \Phi \end{cases} \\
\bar{X_1} &= \begin{cases} 1 - X_1 & if X_1 \in [0,1] \\ \Phi & if X_1 = \Phi \end{cases}
\end{aligned}
\tag{3.8.1}
$$

The fuzzy relations between items are also considered, for instance, symptom-diagnosis (SD), symptom combination-diagnosis ((SC)D), symptom-symptom (SS), and diagnosis-diagnosis (DD). There are two dimensions to the depth of the relationship between each element, namely the frequency and strength of confirmation $m()$. Table 3.8.1 presents some examples of the strength value. In summary, we can write the knowledge that relates to the symptom-diagnosis as "*IF A THEN B WITH (O,C)*" where O and C can be found as linguistic expressions in Table 3.8.1.

Chapter 4

Unified Framework for the Guided Model Discovery

> "The models should shed light on the limit of what can be learned, just as computability does on what can be computed." – Valiant [103]

This chapter is intended as a demonstrator for the proof of concept to aspects of modeling optimization problems using the proposed framework. Although no guaranteed strategy will resolve every modeling scenario, the structure being discussed allows for the application of some conventional wisdom about optimization to be applied to the corpus, continuously. The materials presented in this chapter plays a role similar to that of a consultant that an OR analyst might allow to internally work inside his or her head before, during, or after the models have been formulated. We wish to emphasize again that the advice we shall offer is not foolproof, but based on the corpus knowledge at hand, and many of the statements to be made should (and will often) be qualified.

4.1 Updated Representation Framework

Figure 4.1.1 illustrates an update to the data and information flows that presented earlier in Chapter 2 (Figure 2.1.1) using the Unified Modeling Language (UML) behavioral diagram to visually represent the dynamic nature of the proposed framework.

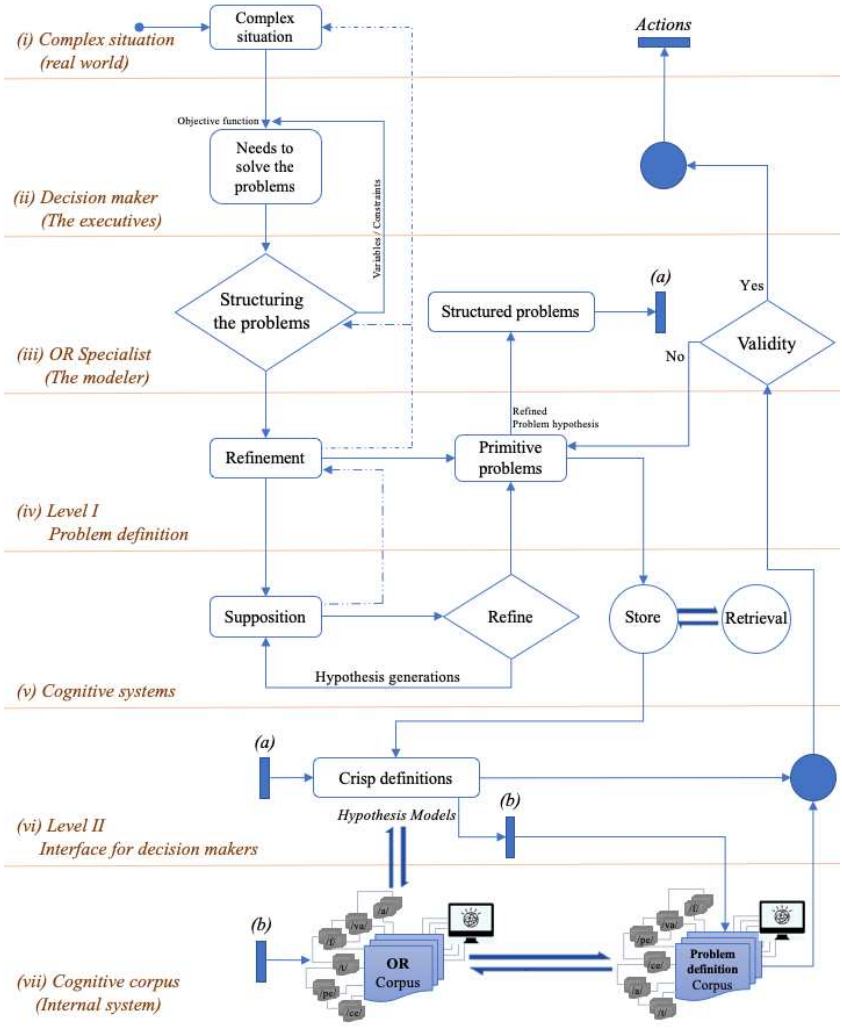

Figure 4.1.1: UML Representation for framework

The efficient optimization of the model can be critically dependent on the specific properties of the formulation. It is often the case that the formulation of the model must make many subjective decisions that do not affect the accuracy of the model, yet are crucial to whether the model is compliant to solutions by optimization algorithms. The UML representation framework is composed of seven layers: (i) complex situation, (ii) decision maker, (iii) the modeler, (iv) problem definition, (v) cognitive systems, (vi) interface for decision makers, and (vii) internal system corpus.

The solid arrows in Figure 4.1.1 indicate the rational thought process of information flow for the (optimization) modeling cycle. Moving from the complex situation (real world) to the need to solve problems by structuring them utilizing the guided model discovery, as shown in layers (iv) to (vii), allows a modeler to cope with the fact that the nature of a possible model may vary so much. The dashed arrows indicate feedback iterations that an OR analyst should consider in the initial stages based on professional intuition before allowing the proposed model reformulation process to refine the model further in a more automated fashion.

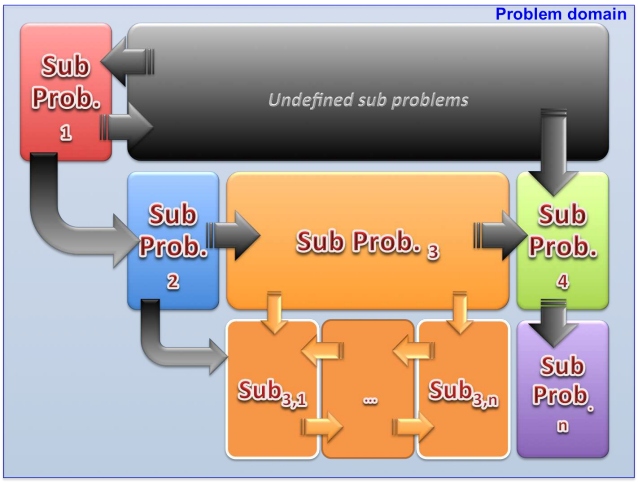

Figure 4.1.2: Segregate problem solving

It is here that abstracting away irrelevant details and focusing on the crucial elements takes place. Figure 4.1.2 depicts a somewhat different approach that starts from a natural way to solve large combinatorial problems consisting of decomposing them into independent sub-problems. However, due to the nature of an ill-structured problem domain, there exist undefined sub-problems. These problems are solved with an appropriate procedure. This segregated approach may lead to solutions of moderate quality, since sub-problems might have been matched to a well-defined domain. Even so, it is not easy to find appropriate ways to decompose a problem apriori.

Newell and Simon proposed a general method for problem-solving in the 1950s. According to their method, there are four stages for problem-solving: (i) problem identification, (ii) strategy planning, (iii) execution, and (iv) the solution and evaluation. A mean-ends analysis is a technique used in Artificial Intelligence for controlling search in problem-solving computer programs, and it consists of the four-steps.

Definition 4.1 Newell's mean-ends analysis

1. Compare the current state, the goal state, then identify differences.
2. Select an operator relevant to reducing the difference.
3. Apply the operator if possible.
4. If it cannot be applied, establish a sub goal of transforming the current state into one in which the operator can be applied.
5. Iterate the procedure until all differences have been eliminated or until some failure criterion is exceeded.

Due to the complex and changing nature of a real-world problem domain, the analysis most likely involves categorical, combinatorics, or symbolic variables rather than real analysis, and they are more likely to be discrete rather than continuous. For example, a large number of human-designed system problems involve combinatorics, symbolic or categorical variables rather than real analysis, and synthesizing a configuration rather than proportioning

the design parameters. The solution to these problems will involve a compromise for "good enough" and rarely achieve optimality.

4.2 Mathematical Relationships Expression

Mathematical expressions play a dominant role in structuring and investigating the relationships among the parameters in a model. An OR analyst often uses equations as the means of evaluating and experimenting that cannot be done in the real world due to dangerous, impossible, or prohibitively expensive circumstances. Equations usually involve two types of quantities: parameters and variables. The parameters are known or controlled numerical values which enter into the formulation. The variables are unknown and are allowed to assume any value in a specified range [94].

The inference mechanism uses these as training experiences to make predictions on which sets of variables, parameters and function have semantic correlations. This information helps formulate new problems. The new problems when formulated join the existing database, thereby enlarging the experienced corpus with each iteration. A wide variety of different functions are used in business applications, but the most frequently is listed as primary functions are as follow:

1 The polynomial of degree n: $f(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n, n$ a positive integer, $a_n \neq 0$

2 Relational functions: $f(x) = P_m(x)/P_n(x)$ where $P(x, f) = 0$ where P_i is a polynomial of degree i

3 The algebraic function: $f(x) =$ the solution of $P(x, f) = 0$ where $P(x, f)$ is a polynomial in two variables x and f.

4 The exponential function: $f(x) = e^x$

5 The logarithmic function: $f(x) = lnx$

The problem is to determine these unknowns. A system of equations may be undetermined if it has more variables than the number of equations and overdetermined in the opposite case. In the undetermined case, some of the variables are arbitrary, and the remaining variables may be expressed as functions of the arbitrary variables. Usually, if the system of equations is not redundant and the number of variables is equal to the number of equations, it is plausible to obtain specific values for the variables as solutions. The values of the variables depend on those of the coefficients, and solutions of equations are thus functions of their parameters. Table 4.2.1 presents three more simple mathematical expression examples. Note that the mathematical expression parameters are chosen to depict the nature of the description statement.

An OR analyst aims toward constructing a model that reflects the real world as closely as possible. However, one does not advocate over-simplification or distortion in model formulation simply to be able to solve the problem more easily. A model to be optimized should be developed by striking a reasonable balance between the aims of improved accuracy in the model and increased ease of optimization [54]. For example, the problem formulation, borrowed from D. Teichroew, "the contribution of return on inventory investment equals the contribution divided by the average inventory investment," the whole expression

Description	Mathematical Expression
Average investment equals one-half of the purchase price plus one-half the cost of getting the merchandise into selling position.	$I = \frac{1}{2}W + \frac{1}{2}D \quad or$ $2I = W + D$
Cost of getting merchandise into selling position equals direct cost in dollars plus a variable cost which depends directly on the selling price.	$D = D_1 + D_2S$
The contribution is the selling price less the purchase price and the cost of getting the merchandise into selling position.	$C' = S - W - D = S - 2I$

Table 4.2.1: Formalizing problems example[106]

is multiplied by the expected turnover rate and by 100 can be represented by

$$C = (\frac{C'}{I})(T)(100) \tag{4.2.1}$$

From an internal system perspective, any expression belongs to ontology class ExpressionEntity. Besides, Figure 4.2.1 portrays how the framework applies Structured Modeling elements for breaking down the input expression.

The reason many real-world optimization problems remain unsolved is partly due to the changing nature of the problem domain, which makes calculus or real variable based methods less applicable. A model to be optimized should be developed by striking a reasonable balance between the aims of improved accuracy in the model and increased ease of optimization. An optimization problem has a structure involving a set of constraints given in the form of equations or inequalities and a function, frequently called an objective function

Expression	/pe/	/ce/	/a/	/va/	/f/	/t/
the contribution of return on inventory investment equals the contribution divided by the average inventory investment," the whole expression is multiplied by the expected turnover rate and by 100	contribution, return, inventory, investment, equals, expression, expected turnover	the contribution of return	100			
		C	100	C', I, T	C=(C'/I)xTX100	

Figure 4.2.1: Example of SM concept for simple expression

to be maximized or minimized subject to the constraints.

Optimization Modeling Ontology (OM)

Let us consider the ontology of model corpus as we have investigated in Section 3.6 (Figure 3.6.1). Now, Equation 4.2.1 above will be recognized by our framework optimization problem ontology with a class *ExpressionEntity* displayed at the bottom of the second row in Figure 4.2.2. The top-level optimization ontology presents here is the most elementary vocabulary of optimization modeling. There are four classes, *FormulationEntityProperty*, *DataEntityProperty*, *ModelEntity*, and *ExpressionEntity* respectively. ModelEntity, and ExpressionEntity respectively. ModelEntity has two subclasses, namely, *DataEntry* and *FormulationEntry*.

The top-level optimization ontology structure can be visualized by OntoGraf features as shown in Figure 4.2.3 below. Note that OntoGraf is a built-in tool available in Protégé software for organizing our created OWL ontology. For simplification and intuition, the prefix *OM* will be used to denote our top-level Optimization Modeling Ontology. `OM#ModelEntity`

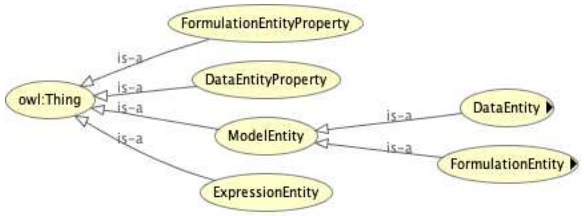

Figure 4.2.2: OWLViz of the top-level optimization modeling ontology

is the class for all entities happening in an abstract optimization model. There are two classes, namely, `OM#DataEntityProperty` and `OM#FormulationEntityProperty` that respective conceptualizations for the model entities are classified. These two classes are also managed under the `OM#ModelEntity` class.

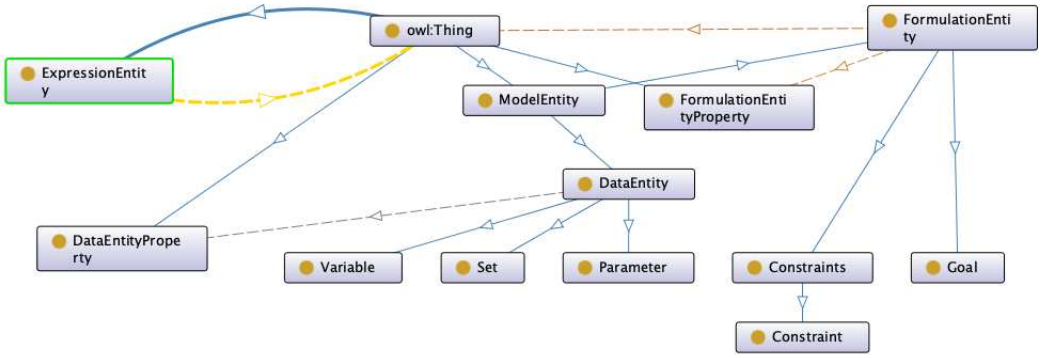

Figure 4.2.3: OntoGraf of the class of optimization modeling ontology (`OM`)

For the integration with established AML representations, every model entity carries a unique string-identifier as a name of an abstract optimization model. Reusable formulation conceptualizations will form object property restrictions that reflect the range to own subclass in the corresponding domain. (Re)Formulation of these reusable conceptualizations will be discussed in the following section. A complete definition of the ontology in Manchester Syntax can be found in Appendix E of this dissertation.

4.3 (Re)formulation

Optimization is a normative or prescriptive field in which among all possible solutions which satisfy a system of constraints, that which yields an optimum (maximum or minimum) to the objective function value is sought. Often the process of solving an optimization problem can be accomplished by the process of solving a set of equations. Unfortunately, optimization problems in their full generality do not readily lend themselves to allowing the user to abandon the need to think since certain critical aspects of the problem can in many instances be defined only by the user [54]. Optimization techniques include many decisions that are inherently problem-dependent, which are often reflected in specific parameters associated with the implementation of the algorithm. A set of mathematical relationships appear in models in various forms, such as equations, inequalities, and logical dependencies. Ultimately, variables may be omitted; constraints may be added, transformed, or removed; conditions and relationships may also be modified to make the model easier to solve.

4.3.1 Bertrand's Decision Function Concept for Reformulation

J. W. M. Bertrand studies the design and monitoring of a master production scheduling in a manufacturing resource planning environment in Efficiency of Manufacturing Systems [108]. He illustrates a conceptual framework of a decision function with seven keys assumptions. He defines the decision function component in Definition 4.3.1. The conceptual decision function as shown in Figure 4.3.1, includes: $I'(t)$ representing possible decisions that are investigated with respect to their expected result $D'(t)$, given the state of the process, $S(t)$, and given the expected values of the environmental variables, $E'(t)$. The

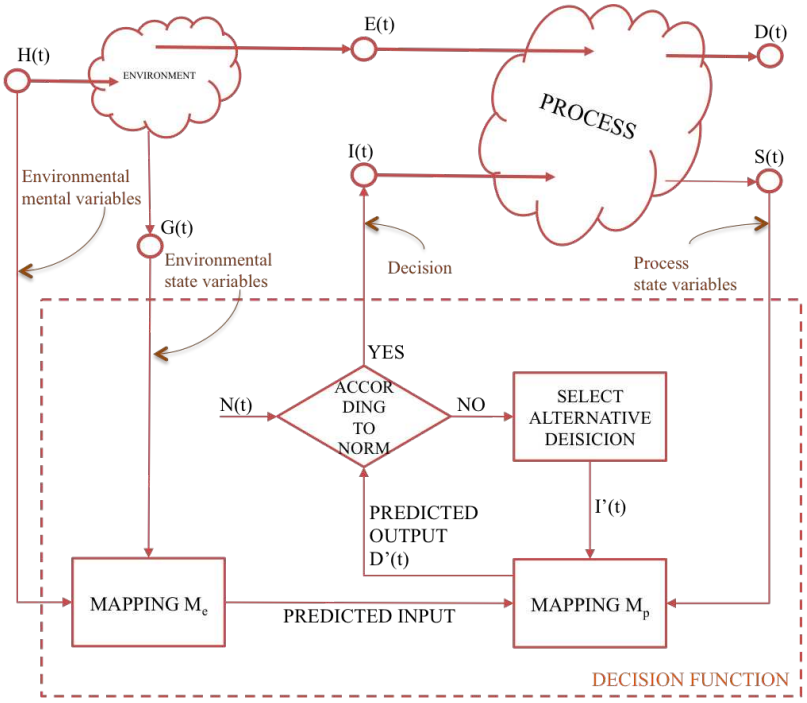

Figure 4.3.1: Bertrand's decision function [108]

models M_e and M_p will not be perfect: Not all variables influencing $D(t)$ or $E(t)$ in reality will be incorporated; Their relationships may really be much more complex. $H(t)$, $S(t)$ and $G(t)$ may contain errors.

Definition 4.2 Bertrand's decision function

There exists a decision function on a control problem. The decision function has:

$N(t)$ norms for a set of goal variables,

$D(t)$ the decision function can manipulate a set of decision variables,

$I(t)$ the goal variables are influenced by the controllable variables,

$E(t)$ a set of environmental variables, which influence the goal variables,

$S(t)$. a set of process state variables for the dynamic process.

'PROCESS' is the relationship between the controllable variables and the environmental variables, on one hand and the goal variables, on the other hand.

Optimization methods involve numerous decisions that are inherently problem-dependent, which are often reflected in certain parameters associated with the implementation of the algorithm. There are no a priori optimal choices of these parameters for all problems. However, in order to simplify life for the inexperienced or uninterested user, optimization software tends to have two forms of user interface that remove the need for the user to specify these parameters. The formal decision procedure may not always be completely implemented for various reasons: each of these different sources of uncertainty constitutes a disturbance.

Let us apply Bertrand's logic by focusing on an optimization system with an abstract optimization model as a core component. With the ontological concept in mind, mathematical properties of the formulation and the data individuals such as convexity, integrality, linearity, and others are to be specified when a specific optimization model is defined in the ontological representation. Reusable conceptualizations are to be provided in other ontologies that use the (top-level) Optimization Modeling Ontology (`OM`) and the Optimization Modeling Mathematical entity Properties Ontology (`OM.MProp`) (§4.3.2) respectively.

4.3.2 Mathematical Entity Properties Ontology (`MProp`)

Figure 4.3.2 below visualizes the `OM.MProp` ontology for defining (or generating) mathematical properties of abstract optimization models entities. As discussed in the previous subsection, the optimization modeling ontology (`OM`) (Figure 4.2.3) import this `OM.MProp` ontology then using the class `OM#FormulationEntityProperty` and `OM#DataEntity-Property` as object properties.

For references, a complete specification in Manchester Syntax can be found in Appendix

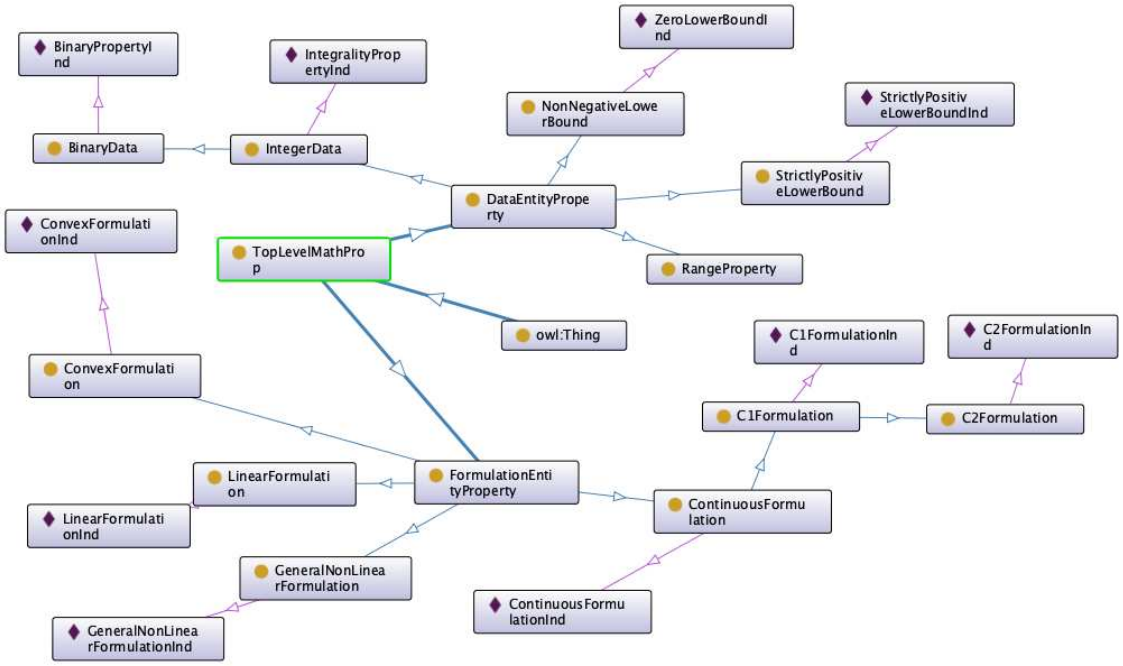

Figure 4.3.2: OntoGraf of the class for mathematical properties ontology (`MProp`)

E of this dissertation. In the same manners as in the previous section, ontology name and classes are presented in a `Typewriter` font and the prefix `OM.MProp` is used to identify ontologies' classes. Hence, the discussion will simply use the class name and omitted the prefix. The arcs in Figure 4.3.2 represent that a class has subclass (inner arcs) or has individual (outer arcs).

Starting from the middle of the figure, `owl:Thing`, `OM.MProp` has a top-level concept (highlighted in the figure), `TopLevelMathProp` where two main ontology classes are set up, namely, `FormulationEntityProperty` and `DataEntityProperty` respectively.

To enable the SWRL-rules* of the knowledge for abstract optimization models specified in

*SWRL is an acronym for Semantic Web Rule. Language. • SWRL is intended to be the rule language of the. Semantic Web. can be used to express rules as well as logic, combining OWL DL or OWL Lite with a

the ontology formalism, the individual types (display with a diamond symbol in the figure) are defined, e.g., `BinaryPropertyInd`, `ContinuousFormulationInd`, `StrictlyPositive-LowerBoundInd`, ..., `ZeroLowerBoundInd`, to name a few.

By designing ontology for mathematical properties this way, the respective individual types function as a collection of a separate data property; for instance, for instance, in the `DataEntityProperty` class: the `NonNegativeLowerBound` class has the `ZeroLowerBoundInd` which provides a concept to indicate a lower bound of zero; while the `IntegerData` class has the `IntegralityPropertyInd` for specifying integer, and the `BinaryData` class has `BinaryPropertyInd` for 0-1 ranges of parameters and variables.

4.3.3 (Re)Formulation with Soft Systems Methodology

Proposed by Peter Checkland [14, 15, 16, 17, 18, 19]as a framework for implementing soft systems thinking, Soft Systems Methodology (SSM) was developed as part of "Soft" operational research. SSM is a seven-step action-oriented process of inquiry into problematical situations in the real-world. Users learn along the way from finding out more about the situation to allow defining/taking action to improve it. The most developed soft OR methodologies are: (a) Soft Systems Methodology (Checkland's) [17], (b) Interactive Planning (Ackoff's) [3, 4], (c) Strategic Assumption Surfacing and Testing, SAST (Mason and Mitroff's) [16], (d) Systems Intervention Strategy (Mayon-White) [79], (e) Social System Design (Churchman's) [14], (f) Cognitive Mapping [109, 25, 86], SODA (Eden's) [78], and (g) Viable Systems Diagnosis (Beer's) [92].

subset of the Rule Markup Language.

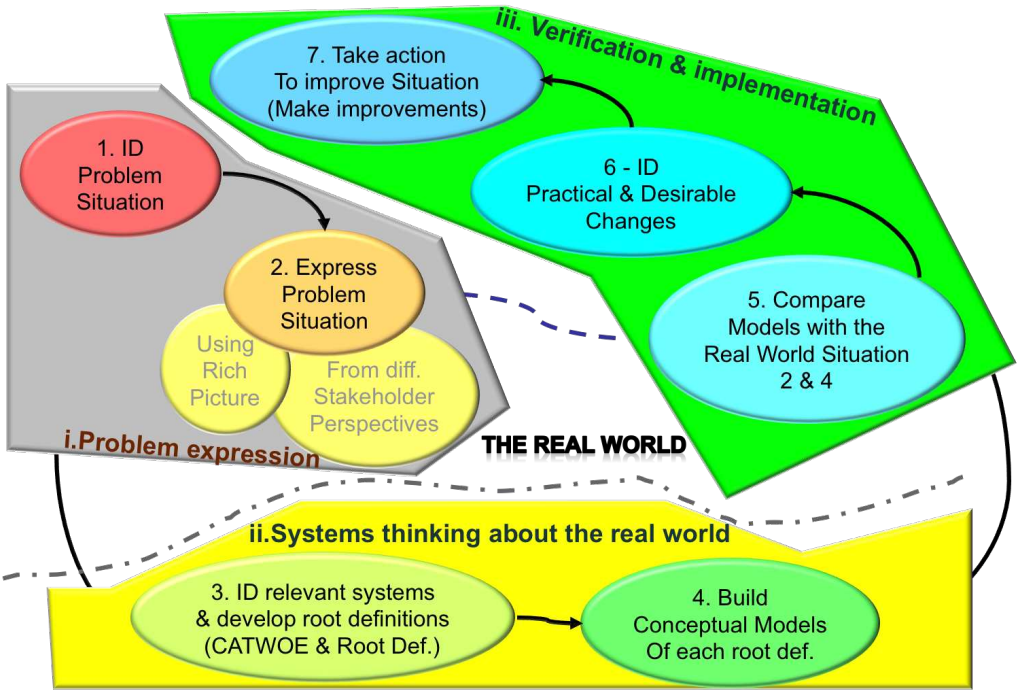

Figure 4.3.3: The SSM learning cycle (adapted from Jackson [63])

Figure 4.3.3 represents the seven steps of SSM along with the proposed three unified learning perspectives: (i) problem expression, (ii) system thinking about the real world, and (iii) verification and implementation. Some questions to consider are: What resources are deployed in what operational processes under what planning procedures with in what structures, in what environments and wider systems, by whom? How is resource deployment monitored and controlled?

- Problem expression (stage 1 and 2)

  1.Situation: Problem situation, identifying 'soft' elements within people, culture, and politics,

2.Expression: Express problem situation, creating 'rich' picture of the situation with respect to worldview.

- System thinking about the real world[†] (stage 3 and 4)

3.ID: Develop root definition and ID relevant systems by CATWOE: customers, actors, transformed, weltanschauung (worldview), owner, and environment.

4.Models: Build conceptual models of each root definition.

Primary: focusing on officially declared task of the system,

Issue-based: focusing on problem issues.

- Verification and implementation (state 5-7)

5.Compare: Models with the real-world situation (2. and 4.)

6.Identify: Practical versus desirable changes

7.Action: Implementation and make improvements

SSM is "sponge-like", soaking up as much as possible of the present situation. [93]. The learning in SSM is perceived when the structured process of the real situation is explored. It uses the model to provide the structure of purposeful activities, enfolding pure, stated world-views.

[†]Proposed an alternative weltanschauung representation in §3.2

4.4 Learning Mechanisms

The learning paradigm for situational improvement using soft systems methodologies was initiated by Checkland in 1981. This approach is also called the interpretivist (Jackson and Mingers) paradigm, while Ackoff calls it the 'design approach' that attempts to solve systems of problems or "messes." These learning paradigms differ from those of the 'research approach' in that they aim to tackle the context or environment where the mess takes place and tries to alleviate or solve the systems of problems rather than solving. Jackson included SAST - Strategic Assumption Surfacing and Testing which explores purposes and 'Interactive Planning' by Ackoff in this setting.

Definition 4.3 Settings for Learning from Examples

Assuming there exists two agents: a "learner" and a "knower," where a knower has knowledge about a certain universe of discourse U. That is, a knower is able to classify elements of the universe U, and classes of his classification form concepts to be learned by a learner.

4.4.1 Learning Scheme

The concepts of system adaptation and self-learning are quite general and can conjure up all sorts of ideas. However, a system can be configured in such a way that allows it to in some sense progressively organize itself towards a learned state in response to input signals. All learning has to be with respect to some appropriate context and suitable constraints. Based on the SSM's skeleton, proposition 4.4.1 highlights stage 5 to 7 of SSM, where a set of minimum necessary condition is being used and learning is developed.

Proposition 4.4 Essential Learning in SSM

From the machine learning view, Let E be the existence of a large set of examples , each example $e \in E$ is a pair $e = \{s, r\}$ where $s \in S$ represents the input from SSM stage 2 and $r \in R$ represents the conceptual model built when receiving s.

One must find a function f which maps $S \rightarrow R$ for as much as possible.

This procedure enables us to eliminate all unnecessary knowledge from the knowledge base, preserving only that part of the knowledge which is really useful. The root definition of a system relevant to the problem was modeled in stage 3 and identified as the "formal systems model." The last two stages of SSM are: comparing models with the original situation and implementation of agreed changes.

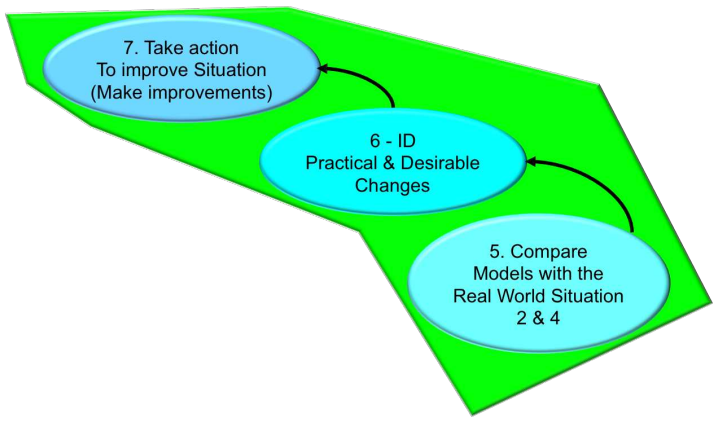

Figure 4.4.1: Verification and implementation of SSM

Let us regard the learning strategy as an event of knowledge acquisition throughout the modeling process in the absence of specific programming. This is done by choosing an appropriate information gathering mechanism, the learning protocol, and exploring the class of concepts that can be learned in a reasonable number of steps (i.e. in polynomial time).

This procedure is called 'reduction of knowledge' in some research communities. Reduction of knowledge consists of removing nonessential partitions, and categorizing in the knowledge base in such a way that the set of elementary categories in the knowledge base is preserved. According to Mitchell, Machine Learning is the study of computer algorithms that improve automatically through experience. The goal of Machine learning is the development of alternative algorithms that increase the ability of an agent to match a set of inputs to their corresponding outputs. Checkland introduced the intellectual functional learning framework as shown in Figure 4.4.2.

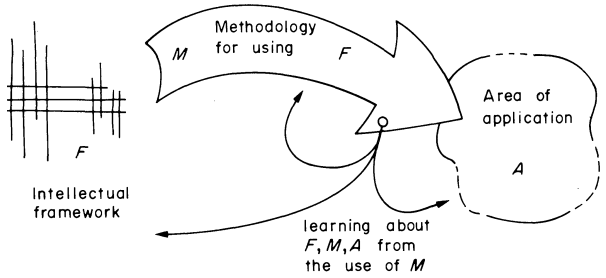

Figure 4.4.2: The functional learning concept (Checkland (1985)) [14]

A successful formalization of an ill-structured domain cannot avoid the implicit logical background of the following two contrasting definitions: the closed world assumption (CWA) and the open world assumption (OWA).

Definition 4.4 Closed & Open World Assumption

CWA : It is assumed a knower has complete knowledge about the universe U (a knower is able to classify every object of the universe U). Under this view, U is assumed to be closed that there is nothing besides U existing.

OWA: It is assumed a knower may have a lack of complete knowledge about the universe U. This does not imply falsity.

A learner's task is to learn knower's knowledge, assuming that a learner is able to distinguish some attributes of objects in the universe, U. A knower exhibits all steps of each object to a learner together with the information for the concept it represents. The first formalization of the CWA rule was introduced by Reiter (1978). It was called the Herbrand Universe of a Theory, $HB(T)$. He proposed the following syntactic formalization:

$$CWA(T) = T \bigcup \neg A : T \nvdash A \text{ and } A \in HB(T) \tag{4.4.1}$$

Decisions on CWA and OWA determine the understanding of the actual semantics of a conceptual expression with the same notation of concepts.

Essence of machine learning

Definition 4.5 Mitchell's broad denotation of 'learning'

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

According to machine learning philosophy, we must indicate these three features: the class of tasks, the measure of performance to be improved, and the source of experience.

Holland et.al. [60] augmented the conventional approach to problem solving with mechanisms for seeking additional knowledge stored in memory that may clarify ill-defined problems. Such restructuring implies that search takes place not only in the space of potential "next states" along a temporal dimension but also through a space of alternative categorizations of the entities involved in the problem.

Definition 4.6 Learning in a Connectionist View
In a connectionist learning model, the role of a unit in mental processing is defined by the strength of its connections to other units. In this sense, the knowledge is in the connections, rather than static and monolithic representations of concepts. Learning consists of the revision of connection strengths between units.

This type of processing depends on the parallel activity of multiple pieces of knowledge that both compete with and complement each other in revising the problem representation. Such interactive parallelism is a hallmark of the theoretical framework for induction.

4.4.2 The Modeling Gap

Models may inherently incomplete, biased, and ruled by its creator; by bounding attention to parts of the applications that are under examination and ruling out any other parts that has never been under consideration. Borrowing a point made elsewhere, incompleteness of specification is caused by incomplete knowledge being currently available, incomplete coverage of the specification, or by inability to represent the knowledge in the application. Incompleteness may be considered to be the main source of the modeling gap, besides culture and skills of modelers. The application is either partially known, or only partially specified, or cannot be properly specified. To overcome the problems of specifications I may either use:

- *negated specifications* that specify those cases which are not valid for the application,
- *robust specifications* that cover the main cases of the applications, or

- *approximative specifications* that cover the application on the basis of control parameters and abstract from order parameters.

Identifying a gap is crucial in improving modeling techniques. In general, a modeling gap is the void between the model's capture of requirements and system's specifications.

"Gap" in a hard paradigm

With the current computing power, the interest in mathematical programming has shifted tremendously. The advantage of mathematical models is that they can be analyzed in a precise way by means of mathematical theory and algorithms. From this advancement, mathematical models become the center of problem solving rather than problem description. In an ill-structured domain, the conclusions drawn from the model are qualitative statements that are derived by analytic means from mathematical theory. It is important to formulate the concise mathematical model that is tailored to the available analytical methods.

For illustration, considering a gap of the following minimization problem that is defined by two items, a set $P \in \mathbb{R}^n$ and a real valued function f. Solving this minimization problem to find x in P such that,$f(x) <= f(y),\ \forall y$ in P,$p \in P$ is defined by constraints (linear, logic, integer, quadratic, etc) on the vectors of $\mathbb{R}^n$.

This problem is solved to optimality when its solution meets the following two conditions: (i) find x in P, such that (ii) for all y in P, $f(x) <= f(y)$. If the solution x only satisfies the first condition, then x is called a feasible solution. The second condition can be much harder than the first one, depending on P and f. With the current optimization techniques, one can easily prove optimality.

For some problems, proving optimality may not be permitted in a reasonable computation time. One may find a lower bound, L_B, on all possible values of the objective function, such that

$$L \leqq f(y), \tag{4.4.2}$$

,then the feasible solution x can be evaluated by comparing with the gap, i.e. if $gap = 0$, then the second condition is satisfied; if $gap \neq 0$, then we know that this problem need more exploitation to achieve a better solution. This gap can be calculated by:

$$gap = \frac{f(x) - L_B}{f(x)} \tag{4.4.3}$$

For comparing different feasible solutions with different gaps on the same lower bound, it is obvious that the smaller the gap, the tighter the lower bound and the better the confidence we have for such a solution.

4.5 Guided Problem-discovery Wizard (GPW)

The guided problem-discovery wizard (GPW) employs 'first-principles thinking' where the idea is to break down complicated problems into basic elements and then reassemble them from the ground up. The basic elements conform to Geoffrion's Structured Modeling concept loosely. Therefore, the GPW framework, proposed in this research, is considered as a rugged system that is not limited to domain but can be trained to recognize outside its knowledge-base, exhibiting "common sense reasoning," jointly with an OR analyst. GPW

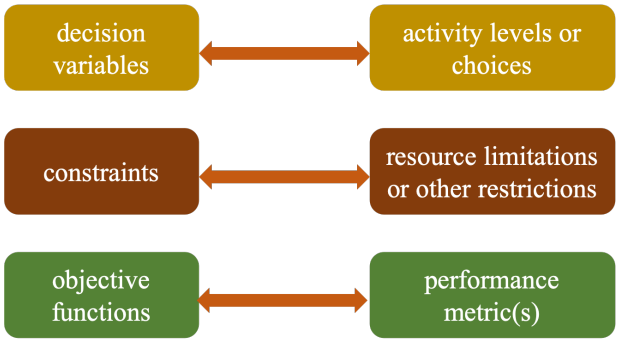

Figure 4.5.1: Optimization mindset

as outlined here, can only present a textual examples and implementation logic to illustrate essential aspects of the developed model (re)formulation framework. The software and interface implementation are stated as a future research perspective at this point (refer to §5.3). However, this section covers (i) input and interactions with ontology definitions, (ii) control of parameters and reformulation, and (iii) output; thus leaving the graphic user interface for future efforts.

Figure 4.5.1 presents the optimization mindset where each optimization element is paired with its own feature. GPW's advantage is adding the logical construct of intersection (semantic nature), compared to concepts present in the area. Therefore, any terms (grammatical categories) can be associated together. One way to deal with this front-end situation, i.e., *terms differentiation* is to express how closely the mapped terms are equivalent to each other using semantic mapping and similarity measurement. Another way is to employ *SoftOR* technique, as presented earlier in Chapter 3, using AWR techniques. Figure 4.5.2 illustrates the basic terms mapping example.

At this point, the discussion on Fuzzy Domains in §2.8 and the Strength of Confirmation in §3.8 become an essential function in managing the assigned weight of equivalence between

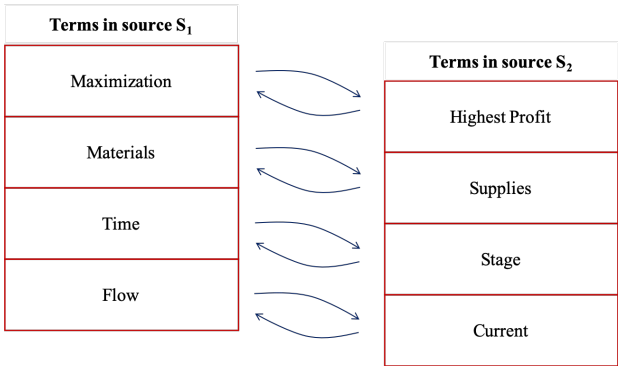

Figure 4.5.2: Terms mapping example

those mapped terms. The architecture of matching a query with a domain ontology process is presented in Figure 4.5.3.

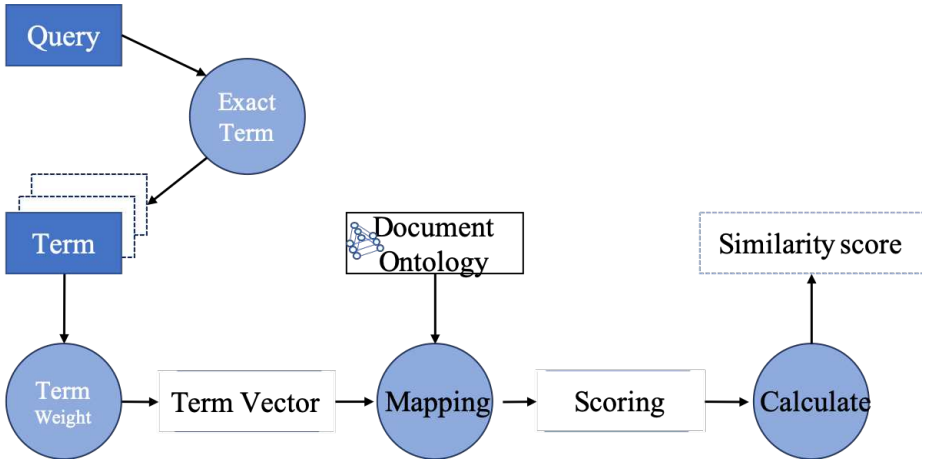

Figure 4.5.3: Ontology extraction process (Adapted from [75])

Similar to terms mapping, the ontology extraction is also aimed at measuring the similarity between the query term and a domain ontology. The input for this query matching process is a query term list containing at least one term and a domain ontology graph. The output is ranked concept model elements representing the input query with similarity scores.

For simplification, considering a toy example where the mapping definition is a 3-tuple

containing the elements: $M =< t_i, t_j, w >$ where t_i and t_j is the term appearing in source S_1 and S_2 respectively; and w is the assigned weight to the term mapping (Adapted from [75] for an idea illustration).

Toy Example GPW1– Domain Ontology Graph OG_ADefinition A sample of domain Ontology Graph OG_A containing only terms and relations representing the domain A is defined as follows. The tabular form OG_A is shown in Table 4.5.1.

$$T : OG_A = A, B, C, D, E$$

$$M : OG_A = \left\{ \begin{array}{l} (A,A,1),(A,B,0.1),(A,C,0.2),(A,D,0.3),(A,E,0.4),(B,A,0.1), \\ (B,B,1),(B,C,0.2),(B,D,0.3),(B,E,0.4),(C,A,0.1),(C,B,0.2), \\ (C,C,1),(C,D,0.3),(C,E,0.4),(D,A,0.1),(D,B,0.2),(D,C,0.3), \\ (D,D,1),(D,E,0.4),(E,A,0.1),(E,B,0.2),(E,C,0.3),(E,D,0.4), \\ (E,E,1) \end{array} \right\}$$

	A	B	C	D	E
A	1	0.1	0.2	0.3	0.4
B	0.1	1	0.2	0.3	0.4
C	0.1	0.2	1	0.3	0.4
D	0.1	0.2	0.3	1	0.4
E	0.1	0.2	0.3	0.4	1

Table 4.5.1: Table of the terms and relations in OG_A

Toy Example GPW2 – Concept Formation A "concept" represents a large knowledge object based on a single term or multiple terms with relations in the ontology graph OG_A. The most basic concept is defined by using a term t, where $t \in T : OG_A$. Hence, all single-term concepts in OG_A can be formulated including: C_A for the concept("A"), C_B for the

concept("*B*") and so forth. The formulating concepts are shown below.

Mapping of concept C_A

	A	B	C	D	E
A	1	0.1	0.2	0.3	0.4
B	0.1	-	-	-	-
C	0.1	-	-	-	-
D	0.1	-	-	-	-
E	0.1	-	-	-	-

Mapping of concept C_B

	A	B	C	D	E
A	-	0.1	-	-	-
B	0.1	1	0.2	0.3	0.4
C	-	0.2	-	-	-
D	-	0.2	-	-	-
E	-	0.2	-	-	-

Mapping of concept C_E

	A	B	C	D	E
A	-	-	-	-	0.4
B	-	-	-	-	0.4
C	-	-	-	-	0.4
D	-	-	-	-	0.4
E	0.1	0.2	0.3	0.4	1

Mapping of concept C_C

	A	B	C	D	E
A	-	-	0.2	-	-
B	-	-	0.2	-	-
C	0.1	0.2	1	0.3	0.4
D	-	-	0.3	-	-
E	-	-	0.3	-	-

Mapping of concept C_D

	A	B	C	D	E
A	-	-	-	0.3	-
B	-	-	-	0.3	-
C	-	-	-	0.3	-
D	0.1	0.2	0.3	1	0.4
E	-	-	-	0.4	-

Table 4.5.2: Visualized *OG* for concepts

Toy Example GPW3 – Query Ontology Graph Extraction Continuing the investigation of the Ontology Graph OG_A definition from the toy example GPW1, let us consider two examples about query d_1 and d_2 (document length=4), which are defined as follows.

Step 1: Obtain the query content

$d_1 : A - A - B - D$

$d_2 : D - D - D - E$

Step 2: Transformed to weighted-term-vector

To transform to weighted-term-vector, let's every term in each document is weighted by W_{t_i,d_j} where t_i represents the i^{th} distinct term in the document j. Let us denote $tf_{i,j}$ as the frequency of term i appearing in query j and dl_j as the query length j. The weighted is defined as $W_{t_i,d_j} = \frac{tf_{i,j}}{dl_j}$.

Our transformed term vectors of the two queries are as follows:

$$T_{d_1} = \{(A,0.5),(B,0.25),(D,0.25)\}$$

$$T_{d_2} = \{(D,0.75),(E,0.25)\}$$

Step 3: Term list creation

Thus, term list creation for the two queries, OG_{d_1} and OG_{d_2} are: $\begin{matrix} OG_{d_1} = \{A,B,D\} \\ OG_{d_2} = \{D,E\} \end{matrix}$.

Step 4: Concept formation

The results of concept formation are shown in Tables 4.5.3, where $\begin{matrix} OG_{d_1} : C_{A,d_1}, C_{B,d_1}, C_{D,d_1} \\ OG_{d_2} : C_{D,d_2}, C_{E,d_2} \end{matrix}$.

$C_{A,d1}$

	A	B	C	D	E
A	0.5	0.05	0.1	0.15	0.2
B	0.05	-	-	-	-
C	0.05	-	-	-	-
D	0.05	-	-	-	-
E	0.05	-	-	-	-

$C_{B,d1}$

	A	B	C	D	E
A	-	0.025	-	-	-
B	0.025	0.25	0.05	0.075	0.1
C	-	0.05	-	-	-
D	-	0.05	-	-	-
E	-	0.05	-	-	-

$C_{D,d1}$

	A	B	C	D	E
A	-	-	-	0.075	-
B	-	-	-	0.075	-
C	-	-	-	0.075	-
D	0.025	0.05	0.075	0.25	0.1
E	-	-	-	0.1	-

$C_{D,d2}$

	A	B	C	D	E
A	-	-	-	0.225	-
B	-	-	-	0.225	-
C	-	-	-	0.225	-
D	0.075	0.15	0.225	0.75	0.3
E	-	-	-	0.3	-

$C_{E,d2}$

	A	B	C	D	E
A	-	-	-	-	0.1
B	-	-	-	-	0.1
C	-	-	-	-	0.1
D	-	-	-	-	0.1
E	0.025	0.05	0.075	0.1	0.25

Table 4.5.3: Concept formation results of the toy example GPW3

Step 5: Ontology Graph mapping from the related concepts:

The max weighting is assigned, if the relation of terms t_i and t_j exist more than once among all of the formulated concepts. Hence, we have selected a $MAX(w_{t_i,t_j})$ for every t_i and t_j relation, $M_S(t_i,t_j,w_{t_i,t_j})$. Matching the concepts of the ontology graph for query d_1 and d_2:

$$OG_{d_1} = C_{A,d_1} \times C_{B,d_1} \times C_{D,d_1}, \quad OG_{d_2} = C_{D,d_2} \times C_{E,d_2}$$

Step 6: Relation set creation

Terms and relations in OG_{d_1} and OG_{d_2} are shown in the following Table 4.5.4.

Terms and relations in OG_{d1}

	A	B	C	D	E
A	0.5	0.05	0.1	0.15	0.2
B	0.05	0.25	0.05	0.075	0.1
C	0.05	0.05	-	0.075	-
D	0.05	0.05	0.075	0.25	0.1
E	0.05	0.05	-	0.1	-

Terms and relations in OG_{d2}

	A	B	C	D	E
A	-	-	-	0.225	0.1
B	-	-	-	0.225	0.1
C	-	-	-	0.225	0.1
D	0.075	0.15	0.225	0.75	0.3
E	0.025	0.05	0.075	0.3	0.25

Table 4.5.4: Terms and relations results in ontology graph extraction

Toy Example GPW4 – Query and Domain Ontology Graph Comparison Again, the domain Ontology Graph OG_A definition and the content of the two queries d_1 and d_2 used in this GPW4 example are from GPW1. The following four-step comparison process is the similarity measurement method.

Step 1: Obtain the Domain ontology graph OG_A

Step 2: Obtain the Query Ontology Graph by matching to the domain OG_A (Table 4.5.4 from GPW3 example)

Step 3: Obtain the score of each query by summing up all the relations, excluding all weight of self-relations.

Step 4: Determine the similarity scores:

$$sim(OG_{d_1}, OG_A) = score(OG_{d_1}, OG_A)/score(OG_A) = 1.425/5 = 0.285$$

$$sim(OG_{d_2}, OG_A) = score(OG_{d_2}, OG_A)/score(OG_A) = 2.175/5 = 0.435$$

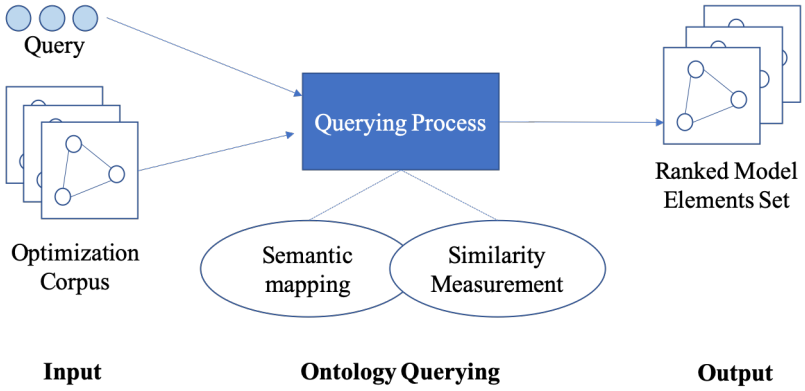

Figure 4.5.4: Ontology querying architecture

Overviews of Ontology Graph Based Querying

As shown in Figure 4.5.4, the querying process of an ontology graph involves a query and an optimization corpus (served as a document set) which are represented by the Ontology Model. The objective is to retrieve a set of model elements that are highly related to the query by ontology matching and similarity measurement. Note, that the weights of the queries and corpus can be obtained utilizing the Fuzzy domain. The provided query is processed with matching related concepts to every entity in the corpus and, then, calculating the similarity score by comparing the weight of those concepts between the query and corpus. The higher similarity score denotes higher relevancy about model entities in a corpus and query. A list of documents is retrieved as a query result after a ranking and sorting process using the calculated scores.

Toy Example GPW5 – MCFP Revisited Let us examine how an OR analyst interacts with the proposed framework by revisiting the previously introduced the minimum cost flow problem (MCFP) examples (introduced in §3.7.1). Consider again, a graph

$G = (V,E), V = \{1,2,...,n\}$ and let there exists non-negative edge capacities u, edge costs c, supply/demand b on each vertex, a flow of $f(v,w)$ on each edge (v,w). This problem has the following model representation:

$$\begin{aligned} &min \sum_{(v,w)\in E} c(v,w)f(v,w) \\ &\quad \text{subject to} \\ &\quad f(v,w) \leq u(v,w) \quad \forall (v,w) \in E \\ &\sum_{w\in V} f(v,w) - \sum_{w\in V} f(w,v) = b(v) \quad \forall v \in V \\ &\quad f(v,w) \geq 0 \quad \forall (v,w) \in E \end{aligned} \tag{4.5.1}$$

There are a couple of observations that need to be made before we proceed on the fuzzification process in using my framework on this example.

1. Residual Graph is a (residual) capacity, $u_f(v,w)$ as for flow, If $(v,w) \in E$ and $(w,v) \in E_F$ then $c(w,v) = -c(v,w)$;

2. We utilize node potentials, $\pi(v)$, by noting that $\pi(i)$ is a dual variable associated with node $i \in N$, therefore the reduced cost of edge (v,w) becomes $c^{\pi}(v,w) = c(v,w) - \pi(v) + \pi(w)$. Hence for any cycle X, we have

$$\sum_{(v,w)\in X} c^{\pi}(v,w) = \sum_{(v,w)\in X} c(v,w)$$

The question here is that given an optimal f, how do we compute π, and vice versa. Now, let us note three properties of a flow:

1 Negative cycles: A feasible flow f is optimal if and only if G_f has no negative cycles.

(a) A feasible is one satisfying all supplies/demands. The 0-flow is not feasible (unless all $b(v) = 0$).

(b) Flow decomposition for min-cost flow. The difference between any two feasible flows i a collection of cycles.

2 Reduced cost optimality: A feasible flow f is optimal if and only if there exists **potential** π such that $c^{\pi}(v,w) \geq 0$ for all $(v,w) \in G_f$. Hence, we looks closely at the node's potential.

3 Complimentary slackness: A feasible flow f is optimal if and only if there exists **potentials** π such that for all edges $(v,w) \in G$, **these rules** apply

(a) if $c^{\pi}(v,w) > 0$ then $f(v,w) = 0$

(b) if $0 < f(v,w) < u(v,w)$ then $c^{\pi}(v,w) = 0$

(c) if $c^{\pi}(v,w) < 0$ then $f(v,w) = u(v,w)$.

The underline idea of the ontology-represented abstract optimization models is allowing the decomposition of the models into more straightforward types of model entities. For instance, parameters, variables, goals, constraints, and specific sets in standard, reusable types with formalized semantics that are favorable to cognitive computing systems become more clearly evident. Figure 4.5.5 illustrates the type definitions as an extension to the previously introduced ontology concept in Figure 3.7.4 in §3.7.1.

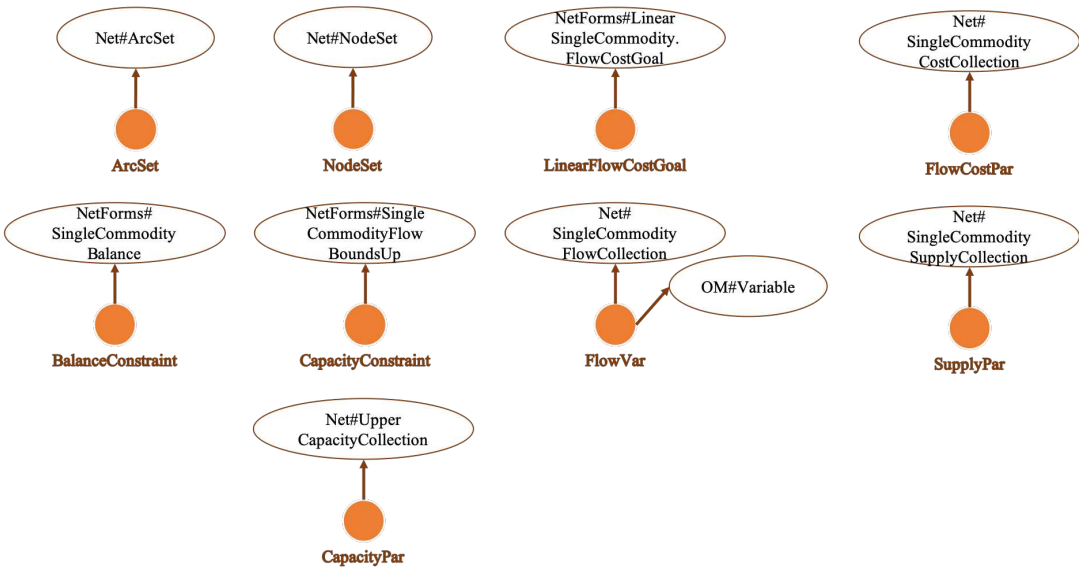

Figure 4.5.5: Type definitions for the MCFP model

4.6 Conclusion

The discussion in this chapter provides the concepts to, possibly, bring us a step closer to the intelligent modeling environment that aids an OR analyst. Note that Ontology plays important role in defining a mechanisms driven, cognitive computing system's "brain" for an understanding of the optimization problem formulation. Because the Structured Modeling (SM) framework, presented in Chapter 2, is not only extensible with object-oriented technologies, but it is also packed with useful fundamentals such as the means for model integration, separation of abstract model and data, and an extensible graph form. SM's features match perfectly with the ontology concept.

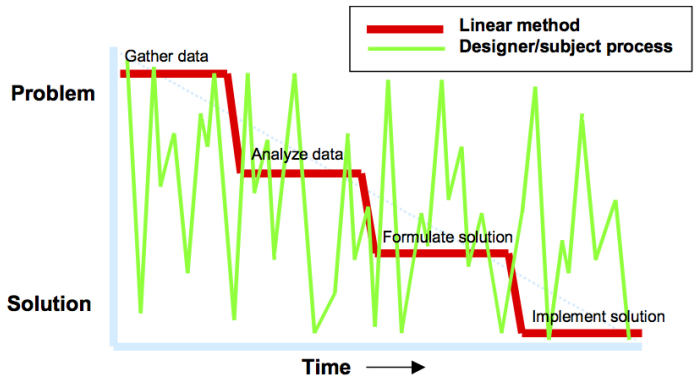

Figure 4.6.1: Cognitive in action

Indeed, the best method is unlikely to find the correct solution to the wrong problem. Errors in the user's formulation are adequately common enough, that they should always be checked for when unexplainable difficulties happen, in practice. Figure 4.6.1 illustrates the cognitive activity graph of a typical OR analyst when working with the model. Unlike the waterfall model of software development that appears to be linear, through the whole process, an OR analyst's brain functions cognitively.

Chapter 5

Conclusions and Suggestions for Future Research

> "Reasoning draws a conclusion, but does not make the conclusion certain, unless the mind discovers it by the path of experience." – Roger Bacon

There are three main sections for this chapter: conclusion, challenges, and future work. The future research part is especially important as the framework proposed within this research includes many ideas that can be extended. In summary, this research attempted a broader vision of model composition, complementing the mathematical modeling cycle which consists of the interpretation of numerical results and development and control of algorithms. Let's start with the conclusion, followed by the challenges, and future work.

5.1 Conclusion

The framework for the optimization problem (re)formulation process is based on the concept that every model can be viewed as a set of distinct elements. This research is motivated by recognizing that the formulation of a mathematical problem is a crucial step in optimization modeling. Algebraic modeling languages (AML) are widely acknowledged as a prime connection between an OR analyst's mathematical concept (portrayed in AML) and the solvers. However, the optimization models contain highly complex algebra that defines the variables and equations that do not, on their own, facilitate the learning mechanism found in cognitive computing systems. The goal was to study the possible backbone for an augmented intelligent system for model formulation.

A host of concepts and procedures from Artificial Intelligence and cognitive computing have the potential to impact the practice of mathematical modeling. Mainly, knowledge-based systems and ontology environments can provide problem representation and associated problem-solving methods that can be used to encode domain knowledge and domain-specific strategies for a variety of model generation and interpretation situations. In general, model generation tools can be categorized into three levels of increasing abstraction: (i) intelligence help system, (ii) customized preprocessors, and (iii) high-level model generation tools. This research endeavored to provide a mechanism for consulting advice to OR analysts.

This research sketched an extension to the original work by Geoffrion's structured modeling and structured modeling language where the model building is formulated from smaller elements, each communicating on its own with a necessary optimization tool to incorporate the corpus. The principal component is an ontology-based representation of abstract

optimization models. This idea of representation allows representing reusable fragments of optimization models, objectives, constraint formalization, and components as an ontology graph.

The mechanisms proposed in this dissertation act as interactive passive consultants to an OR analyst. The systems are not connected to analysis programs and are not meant to provide complete solutions to modeling problems. They simply guide the user (OR analyst, targeted novice analysts) in conducting some modeling tasks. The detailed implementation of AML parsing and of a user interface has been outlined in short, leaving further investigation and implementation open for future research. The main contributions of this research are summarized as follows:

5.1.1 Framework for adaptive decision making

Utilizing a cognitive computing technology platform for problem structuring is a major part of an adaptive decision-making grand scheme (discussed in Chapter 2). Fig. 4.1.1 (Chapter 4) summarizes the proposed framework by separating it into two main levels: (Level 1) Problem definition and (Level 2) Interface for decision makers using a unified modeling language representation. These two levels connect through hypothesis generation that link to the corpus where the shared model alternative is continuously updated.

5.1.2 Separation of General Structure and Instantiating Data

To allow the flexibility to model complex situations, SML (presented in §2.5) uses schema to represent General Structure and put Instantiating Data in the Elemental Detail Table.

Hence, one can differentiate the purposes of the general structure and elemental detail. In addition, a predetermined relational data table is benefitting from a different schema's Elemental Detail Table's rules set. Modelers do not have to worry about data structures or designing such a table.

There are three levels of model representation as discussed in §2.7. The modeling language level is the traditional formulation that an OR analyst implements when dealing with an optimization problem. An ontology representation (presented in §3.5) act as an interface layer between a traditional formulation and the semantic level. Employing this framework, an OR analyst can initiate at the modeling language level: editing, updating, and reusing a model. The cognitive system parser enables the digesting of such a model into an ontology representation with ease.

5.1.3 Exploitation of parallel structure

As discussed in the examples at the end of Chapter 2, 3, and 4, Geoffrion [43] emphasizes that grouping is one of the most appealing properties for definitional systems. The grouping property extends to modeling languages. Furthermore, it enables us to utilize existing models that have common parts along with extending any of the parts. Because such abstractions, once collected in the system are domain independent and hence general purpose, they can be used to suggest simplifications (without loss of generalization) to the model.

5.2 Challenge

To bridge the gap between an iterative process of mathematical model formulation and a classic modeling domain that is established firmly on algebraic modeling language is, to a large extent, an art, a cognitive activity in which one reflects and makes models to describe how systems of interest perform. In particular, an optimization model is a representation in mathematical terms of the behavior of a real world phenomenon. The formulation is clearly defined; however, the cognitive process to create a model is still not mature. Thus, it is an opportunity to consider a system that can advise an OR analyst to incorporate the sophisticating element that may have been missing from the model. As discussed by Pinsky and Karlin [87], there is no such thing as the best model for a given phenomenon.

In the final analysis, a model is judged by its usefulness. Some models are useful as detailed quantitative prescriptions of behavior, e.g., most optimization applications. Another model in a different context may provide only general qualitative information about the relationships among and relative importance of several factors influencing an event. That model is also useful and equally essential but serves the modeler's purpose differently. Our research attempts to identify the pragmatic modeling features that often allows the existence of two or more models for the same event, but still serving a clear purpose.

5.3 Future research

Recent developments in cognitive computing (CC) (understand, reason, learn, and interact) hold the promise of cognitive analytics that digests both formal and informal data and continually build knowledge and learning, naturally, with human rather than traditional

programmable systems. Cognitive computing systems are maturing rapidly, and many new applications are likely to be found. Undoubtedly, CC methodologies utilizing various kind of AI technology will eventually become a natural and integral component of any computing devices, perhaps to the same extent as present-day fuzzy logic tools and algorithmic tools. These tools will then significantly elevate the role and meaning of computers from the current emphasis on calculation to the much broader area of reasoning.

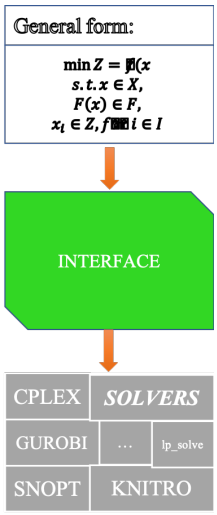

Figure 5.3.1: From Optimization Modeling to Solving

The creation of models–mathematical, qualitative or rule-based–is difficult, time-consuming, and expensive. Optimizing is the overall process by which, starting from data, an optimal solution is identified (including preprocessing/scaling, feature selection, choice of architecture, and choice of critical parameters of the architecture). Figure 5.3.1 summaries the optimization workflow, where an OR analyst interacts with the framework in a way equivalent with a restricted, domain-specific language, while the interface sorts through the properties of the input to configure a proper model representation to give the particular solver in achieving the solution.

One interesting idea is the high-level model generation tools discussed in [23], which incorporates techniques that are more flexible than the heuristics classification approaches. The goal of these tools is to provide a set of powerful representations to the analyst, which, in addition, provides a collection of reasoning mechanisms that can be invoked on the fly. This idea serves as a means of high-level model specification for the model formulation operations.

Consequently, work needs to be done toward ever more refined crisp problem definition, as well as a framework for restructuring the optimization problem and decision analysis template to achieve a pragmatic way for generating sustainable models — ultimately intelligent model formulation. The rest of this section briefly describes some specific potential future research projects.

1 *Emerging Patterns in the Modeling of the Optimization Problem*: One strategy to speed up the ingestion (learning) process of the optimization modeling corpora for cognitive computing systems is to discover the patterns occurring in the successful models (such as models that are used in a testbed). The selection from patterns, contrast pattern mining, and building robust classifiers, then, are the top three research issues. This initiative aims to create a pattern-retrieval mechanism from the models; the mechanism serves as cognitive systems ingestion boosting to expand its knowledge foundation.

2 *Smart Modeling: A Generalized Platform for Decision Model Formulation*: This extension offers a showcase of the Intelligence Augmented Based (IA-Based) model formulation in a decision model that can be executed efficiently. The generalized platform based on the advanced cognitive augmentation modeling scheme provides

an interface with the current decision logic: cognitively learned rule sets for perceiving, formulating, organizing, and managing a business decision. Smart Modeling incorporates desirable characteristics in the decision model such as high flexibility, shareability (intra organization), and replicability, thus fostering opportunity and pragmatics.

3 *Decision Optimization in the Age of Cognitive Technology*: In making use of predictive analytics that captures one field of decision optimization recently pioneered by IBM/Watson, this study explores real-world applications of the interplay between cognitive technology and decision optimization. The integration of various machine learning models feeds into the analysis, presented in this research, offering indicators and tradeoffs to the OR analyst. Evolving goals and reaction to risks and change interventions utilizing cognitive technologies would be of prime interest here.

4 *Cognitive Solutions: An Application for Model Formulations*: The reflections of the OR analyst's (and the mathematician's), the thought process for mathematical model formulations, are captured by cognitive computing technology. The cognitive solutions are the working prototype of the novel model formulations process. Integrating self-learning algorithms that assess and function the same way a mathematician's brain works allow the utilization of on-hand techniques such as existing natural language processing software in accordance with data mining and pattern recognition. The cognitive solutions involve real-time analyses of problem context, environment, and intent (among many other variables) that inform the OR analyst's ability to solve problems.

Appendix A: Definitions used in SML

Definition #	Core Concepts	Informal Explanation
1	Primitive entity element[1] */pe/*	A primitive definition representing any distinctly identifiable entity
2	Compound entity element[2] */ce/*	A definition based on the definitions of certain primitive entity elements or other compound entity elements
6	Calling sequence	A segmented tuple containing all the elements on which a nonprimitive element's definition depends
7	Closed collection of elements	Calling sequences only call other elements in the collection
8	Acyclic collection of elements	No definition in the collection is ultimately circular with respect to calls
9	Elemental structure	Nonempty, finite, closed, acyclic collection of elements
10	Generic structure, genus	An elemental structure partition that does not mix element types; each cell called *a genus*
12	Modular structure, module, default modular structure	A rooted tree whose leaf nodes are 1:1 with the genera of a generic structure; each nonleaf node called a module; degenerate tree with only one module (the root) called the *default modular structure*
13	Monotone ordering	An order for the modular structure tree such that preorder traversal yields all genera in a no-forward-reference order
14	Structured model	An elemental structure together with an associated generic structure and monotone ordered modular structure
23	Modular outline	The indented list representation of the preorder traversal of a (not necessarily monotone) ordered modular structure
24	Element graph	A directed acyclic graph representing all elements of an elemental structure and the calling sequence references among them; has certain node and arc attributes
25	Genus graph	A directed graph representing all genera of a generic structure and the calling sequence references among them
27	Adjacency matrix	A node-node adjacency matrix for an element graph or a genus graph (shows direct calls only)
28	Reachability matrix	A node-node reachability matrix for an element graph or a genus graph (shows indirect as well as direct calls)

Definition #	Core Concepts	Informal Explanation
3	Attribute element[3] */a/*, value, range	A definition with a user-supplied value in a certain range, based on the definitions of certain primitive or compound entity elements
4	Function element[4] */f/*, value, rule	A definition with a value calculated by a certain rule, based on the definitions of certain other elements
5	Test element[5] */t/*, value, rule	A definition with a logical value calculated by a certain rule, based on the definitions of certain other elements
15	Complete specification	A structured model with every detail fully specified
16	Variable attribute	An attibute element whose value the modeler expects to change often or to place under solver control
17	Evaluation	The task of determining the value of every function and test element

APPENDIX

Appendix B: Gagliardi and Spera's Preliminary Results [39]

Propositions

Proposition 1. An Elemental Structure

Let E be a non-empty and finite set of elements, and let G be a set of partitions constructed on E, one for each of the five types. E is an *Elemental Structure* if:

1 G satisfies generic similarity;

2 G is a closed set;

3 G is an acyclic set.

Proposition 2. The Construction of a Normal model

Given a Structured Model S_i, it is always possible to construct a normal model $N(S_i)$ using the neutral set of operations N.

Proposition 3. Creating an Integrated Structured Model

Given $S_1, ..., S_n \in SM$, with $n \geq 2$, it is possible to create an *integrated Structured Model* S_k using a set $\{E_1, ..., E_k\}$ of closed sets of operations.

Proposition 4. Transformation

Given a Structured Model S_i and arbitrary singleton function* genus $f \in FT_i \subset S_i$, there exists *a transformation* T, which is a neutral set of operations with respect to f, such that

$$T(S_i) = SubS_i(f)$$

Proposition 5. Singleton function genus

Given two Structured Models S_1 and S_2, it is always possible to replace the attribute genus $g_i \in A_1 \subset S_1$ with a singelton function genus $f \in FT_2 \subset S_2$. The result is a Structured Model.

Proposition 6. Integrated Model

Given two normal models $N(S_i)$ and $N(S_j)$, the integrated model $[N(S_i), N(S_j)]$ obtained by replacing the input parameter $g_i \in A_i \subset N(S_i)$ with the output parameter $f_j \in FT_j \subset N(S_j)$ is a Structured Model if $i(g_i) = i(f_j)$.

Proposition 7. Replacing Output Model

Given a normal model $N(S_i)$, it is possible to replace the input parameter $g_i \in A_i$ with the output parameter $f_j \in FT_j$ if

$$i(g_i) = i(f_i); \tag{1}$$

*The function from a given nonempty set X to the power set $P(X)$ that maps every element x of X to the set x.

f_i does not have direct or indirect definitional dependencies on any genus having -

direct or indirect definitional dependencies on g_i. (2)

Proposition 8. Replace_Attribute

The Replace_Attribute procedure is closed under *SM*.

Proposition 9. Replace_Index_Component

The Replace_Index_Component procedure is closed under *SM*.

Definitions

Definition 1. Connected module

A Structured Modeling module is *connected* if its genera and their calling sequences from a connected sub-graph.

Definition 2. Sub-model

A *sub-model* is a connected module with at least one primitive entity genus.

Definition 3. Neutral set of operation

Given a model $S_i \in SM$, where *SM* is a set of Structured Models, defining the set T of operations to be *neutral* if the resulting model $T(S_i)$ returns the same output values when instanced with the same data of S_i.

APPENDIX

Definition 4. Neutral set of operation with respect to g_i

Given a sub-model $SubS_i \in SM$, where SM is a set of Structured Models, and a genus $g_i \in SubS_i$, defining the set T of operations to be *neutral with respect to* g_i if the resulting model $T(SubS_i)$ returns the same output values given by g_i when it is instanced with the same data of $SubS_i$ for the genera called directly or indirectly by g_i.

Definition 5. Normal model

A model is called *normal* if the following conditions are satisfied:

- there is a 1:1 correspondence between attribute and compound genera;
  - given a pair of matching genera, there is a 1:1 correspondence between their elements.

Definition 6. Index basis

An *index basis* of a normal model $N(S_i)$ is a couple of genera $B_j = \{a_j, c_j\}$, where $a_j \in A_i \subset S_i$ is an attribute genus, and c_j is the compound entity genus called by a_j. The genus a_j is called the value component of B_j, while the genus c_j is called the index component.

Definition 7. Index basis set

The set $BS_i = \{B_j, j : 1, ..., h\}$ containing all the index basis of $N(S_i)$ is called the index basis set.

APPENDIX

Definition 8. Index function

Suppose L to be a language for the definition of Structured Models. An *index function* $i(g_j)$ is a rule which associates to every genus $g_j \in N(S_i)$, expressed using the language L, the cardinality of its generic index tuple[†].

Definition 9. Closed set of operations

A set of elementary operations E is *closed* if $E(S_1,...S_n) = S^* \subset SM$ for every $S_i \subset SM$, $i : 1,...,n$, $n \geq 1$.

Definition 10. Function sub-model

A Structured Model $SubS_i(f)$ is called a function sub-model if it satisfies the following conditions:

- $SubS_i(f)$ is a normal model.
  - $SubS_i(f)$ has at least one function genus $f \in FT_i$which is a singleton.

Lemmas

Lemma1. Any genus $g \in PC_i$ does *not* have references to any other genera $g_k \in (A_i \vee FT_i)$.

Lemma2. Any genus $g_j \in A_i$ has *only* references to other genera $g_k \in PC_i$.

[†]Gagliardi and Spera used Geoffrion's Structured Modeling Language as Blooms language.

Appendix C: The Single customer IRP problem

Consider the situation where we want to fill up the empty tank of a customer. Let c be delivery cost, I be the initial inventory, and C be tank capacity. It is easy to see that the optimal policy is to fill up completely when it is empty. The cost v_τ for planning period τ is[13]

$$v_\tau = max(0, \left\lceil \frac{\tau u - I}{min(C, Q)} \right\rceil C \tag{5.3.1}$$

Considering a basic VMI inventory-routing problem description where a single product from a single facility is delivered to a set of n customers over a given planning horizon τ, the objective is to *minimize* the distribution costs during the planning period without causing stock outs at any of the customers. The following are basic parameters found in [13, 73], among others.

APPENDIX

τ - length of planning horizon, $t \in \tau = \{1, ..., H\}$

u_i - consumption rate (usage)

B_t - starting inventory at the supplier in period t

U_s - maximum inventory level of customer s

I_{st} - inventory level of customer s in period t

$I_{so} \leq U_s$ - a given starting inventory level

x_{st}* - quantity shipped to retailer s at time t

*depending on replenishment policy:

OU policy: x_{st} is the difference between U_s and the current inventory levelI_{st}

ML policy: the quantity x_{st} can take any non-negative value that does not violate U_s

Q_k - vehicle k capacity

Now, let h_s, h_0 be unit inventory cost at customer s and at the supplier, respectively, and let M be the number of customers to be visited. We consider the following model:

$$Minimize \sum_{t \in \tau'} h_0 B_t + \sum_{t \in \tau} \sum_{s \in M} h_s I_{st} + \sum_{i,j \in M} \sum_{t \in \tau} c_{ij} y_{ij}^t \tag{5.3.2}$$

Subject to

(1) Inventory definition at supplier

$$B_t = B_{t-1} + \mathrm{u}_{0,t-1} - \sum_{s \in M} x_{x,t-1} \qquad t \in \tau',$$

(2) Stock-out at supplier

$$B_t \geq \sum_{s \in M} x_{x,t} \qquad t \in \tau,$$

(3) Inventory definition at customer

$$I_{s,t} = I_{s,t-1} + x_{s,t-1} - u_{s,t-1} \qquad s \in M,\ t \in \tau',$$

(4) Stockout at customer

$$I_{s,t} \geq 0 \qquad s \in M,\ t \in \tau',$$

(5) Order-Up to level (OU) replenishment policy

$$\begin{aligned} x_{s,t} &\geq U_s Z_{s,t} - I_{s,t}, \\ x_{s,t} &\geq U_s - I_{s,t} \qquad s \in M,\ t \in \tau, \\ x_{s,t} &\leq U_s Z_{s,t}, \end{aligned}$$

(6) Capacity constraints

$$\sum_{s \in M} x_{x,t} \leq Q_k \qquad k \in K,\ t \in \tau,$$

(7) Routing constraints

(a) if at least one customer is visited

$$\sum_{s \in M} x_{x,t} \leq Q_k Z_{0,t} \qquad t \in \tau,$$

(b) if deliveries are made at time t, then the route traveled at time t has to contain one arc entering every vertex i on the route and one arc leaving every i,

$$\sum_{\substack{j \in M' \\ j<i}} y^t_{i,j} + \sum_{\substack{j \in M' \\ j>i}} y^t_{i,j} = 2Z_{i,t} \qquad i \in K,\ t \in \tau,$$

(8) Non-negativity and integrality constraints

$$\begin{aligned} & x_{s,t} \geq 0, \\ & y^t_{i,j} \in \{0,1\}, \\ & y^t_{i,0} \in \{0,1,2\} \\ & Z_{i,t} \in \{0,1\}. \end{aligned} \qquad i,j \in M,\ t \in \tau,$$

The following four valid inequalities are additional constraints to better model the problem:

(1) For inventory level at time $t-k$, if the customer is not served in times $t-k, t-k+1, ..., t$, then the inventory level $I_{s,t-k} \geq (1 - \sum_{j=0}^{k} Z_{s,t-j})$. The valid inequality to express this become,

$$I_{s,t} \geq (1 - \sum_{j=0}^{k} Z_{s,t-j}) \sum_{j=0}^{k} u_{s,t-j} \qquad s \in M,\ t \in \tau,\ k = 0, ..., t-1,$$

(2) For OU policy, if $t-k$ is the last time customer s was visited before time t,

$$I_{st} \geq U_s Z_{s,t-k} - \sum_{j=t-k}^{t-1} u_{s,t-j} \qquad s \in M,\ t \in \tau,\ k = 0, ..., t-1,$$

(3) If the customer s is visited at time t, then the supplier has to be included in the route,

$$Z_{s,t} \geq Z_{0,t} \qquad s \in M,\ t \in \tau,$$

(4) If the supplier is the successor of customer in the route traveled at time t (i.e. $y^t_{i0} = 1$ *or* 2), then i has to be visited at time t,

$$\begin{aligned} & y^t_{i0} \leq 2Z_{it}, \\ & y^t_{ij} \leq Z_{it}, \end{aligned} \qquad i,j \in M,\ t \in \tau.$$

Appendix D: Stock-Flow Identities and Non-negativity Conditions

Stock-Flow Identities

For any given ordering z_1, let x_t be the initial stock level in a period, y_t the stock level after ordering, and r_t the amount sold. The stock-flow identity can be written as

$$z_t = y_t - x_t, \tag{5.3.3}$$

$$x_{t+1} = y_t - r_t. \tag{5.3.4}$$

For non-perishable commodity, there is the stock on hand at the end of any period equals the stock on hand at the beginning of the period plus the amount delivered to the firm less the amount sold. As shown in equations (5.3.3) and (5.3.4), where stock holds in each time period (valid with discrete variable).

For continuous taken time, demand and ordering or production as continuous or piecewise continuous function of time, then the relation between initial stock, x_t and the stock level after ordering, y_t disappears, then (5.3.3) and (5.3.4) are replaced by

$$\frac{dy}{dt} = z(t) - r(t), \tag{5.3.5}$$

in integral form,

$$y(t) = y(0) + \int_0^t [z(\tau) - r(\tau)]d\tau. \tag{5.3.6}$$

The distinction between initial stock and stock level after ordering in equations (5.3.5) and (5.3.6) above is an important question of the relation between amount sold (r_t) and demand (ξ_t), as they cannot be equal, otherwise the inventory (x_{t+1}) will be negative.

Non-negativity Conditions

Arrow Et. Al. [8] explain that in case of discrete continuous demands, where $r_t = \xi_t$, if $\xi_t \leq y_t$, then some assumptions need to be made to prevent the inventory to be negative (which is not physically possible):

(a) Assert the inventory policy where inventory x_{t+1} is *never negative*, in other words, the demands are always met so that

$$y_t \geq \xi_t, \text{ for all } t \tag{5.3.7}$$

when demands are uncertain, equation (5.3.7) may be costly or impossible to hold for all possible demands. A *shortage* $\xi_t - y_t$ may occur, leading to a penalty p represent by function $p(\xi_t - y_t)$.

(b) To maintain customer satisfaction, the firm pays a premium over the usual price because some high priority order needs an immediate delivery. The *penalty* becomes the premium cost, so equation (5.3.4) is reshaped to

$$x_{t+1} = max(0, y_t - \xi_t) \tag{5.3.8}$$

where $x_{t+1} \geq 0, z_t \geq 0, y_t$ is nonnegative.

(c) Asserting the inventory policy where *shortage* occur, is to leave an unfilled order on the books and try to satisfy it as soon as possible. Here, assuming that there is also some penalty occurs because the firm fails to satisfy the order. In this case, the negative inventory is usually called a *backlog*, then equation (5.3.8) is replaced by

$$x_{t+1} = y_t - \xi_t. \tag{5.3.9}$$

(d) In the same situation of backlog in 3., in case of customers go somewhere else for their ordering. The firm has a penalty. This reflects the lost of customer goodwill, therefore equation (5.3.8) holds. This occurrence may call *non-backlog* case.

Appendix E: Manchester Syntax of Ontologies

```
OM-Top-Optimization-Modeling
[Manchester syntax rendering:]
refix: : <http://www.semanticweb.org/ci/fl/ontologies/2018/8/OM.owl#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>

Ontology: <http://www.semanticweb.org/ci/fl/ontologies/2018/8/OM.owl>

Datatype: xsd:string

ObjectProperty: dataMProp
   Domain:
      DataEntity
   Range:
      <http://www.semanticweb.org/ci/fl/ontologies/2018/8/OM.owl#DataEntityProperty>

ObjectProperty: expRequires
   Domain:
      ExpressionEntity
   Range:
      owl:Thing

 ObjectProperty: formulationMProp
   Domain:
      FormulationEntity
   Range:
     <http://www.semanticweb.org/ci/fl/ontologies/2018/8/OM.owl#FormulationEntityProperty>

ObjectProperty: requires
   Domain:
      FormulationEntity
   Range:
      owl:Thing

DataProperty: EName
   Domain:
      ModelEntity
   Range:
      xsd:string
```

Manchester syntax for the top-level Optimization Model Ontology (OM) (Part 1 of 2)

APPENDIX

```
Class: <http://www.semanticweb.org/ci/fl/ontologies/2018/8/OM.owl#DataEntityProperty>

Class:
<http://www.semanticweb.org/ci/fl/ontologies/2018/8/OM.owl#FormulationEntityProperty>

Class: Constraint
   SubClassOf:
      Constraints

Class: Constraints
   SubClassOf:
      FormulationEntity

Class: DataEntity
   SubClassOf:
      ModelEntity

Class: ExpressionEntity

Class: FormulationEntity
   SubClassOf:
      ModelEntity

Class: Goal
   SubClassOf:
      FormulationEntity

Class: ModelEntity

Class: Parameter
   SubClassOf:
      DataEntity

Class: Set
   SubClassOf:
      DataEntity

Class: Variable
   SubClassOf:
      DataEntity

Class: owl:Thing
```

Manchester syntax for the top-level Optimization Model Ontology (OM) (Part 2 of 2)

APPENDIX

OM.MProp Manchester syntax rendering:

```
Prefix: : <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProps.owl#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>

Ontology: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl>

Datatype: xsd:double

DataProperty: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#hasLowerRange>
   Domain:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#RangeProperty>
   Range:
      xsd:double

DataProperty: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#hasUpperRange>
   Domain:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#RangeProperty>
   Range:
      xsd:double

Class: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#BinaryData>
   SubClassOf:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#IntegerData>

Class: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#C1Formulation>
   SubClassOf:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#ContinuousFormulation>

Class: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#C2Formulation>
   SubClassOf:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#C1Formulation>

Class: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#ContinuousFormulation>
   SubClassOf:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#FormulationEntityProperty>

Class: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#ConvexFormulation>
   SubClassOf:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#FormulationEntityProperty>

Class: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#DataEntityProperty>
   SubClassOf:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#TopLevelMathProp>
```

Manchester syntax for the Mathematical Properties Ontology (OM.MProp) (Part 1 of 3)

APPENDIX

```
Class:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#FormulationEntityProperty
   SubClassOf:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#TopLevelMathProp>

Class:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#GeneralNonLinearFormulation>
   SubClassOf:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#FormulationEntityProperty>

Class: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#IntegerData>
   SubClassOf:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#DataEntityProperty>

Class: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#LinearFormulation>
   SubClassOf:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#FormulationEntityProperty>

Class:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#NonNegativeLowerBound>
   SubClassOf:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#DataEntityProperty>

Class: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#RangeProperty>
   SubClassOf:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#DataEntityProperty>,
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#hasLowerRange> max 1
xsd:double,
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#hasUpperRange> max 1
xsd:double

Class:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#StrictlyPositiveLowerBound>
   SubClassOf:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#DataEntityProperty>

Class: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#TopLevelMathProp>




Individual: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#BinaryPropertyInd>
   Types:
      <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#BinaryData>
```

Manchester syntax for the Mathematical Properties Ontology (OM.MProp) (Part 2 of 3)

APPENDIX

Individual: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#C1FormulationInd>
Types:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#C1Formulation>

Individual: <http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#C2FormulationInd>
Types:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#C2Formulation>

Individual:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#ContinuousFormulationInd>
Types:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#ContinuousFormulation>

Individual:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#ConvexFormulationInd>
Types:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#ConvexFormulation>

Individual:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#GeneralNonLinearFormulationI
nd>
Types:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#GeneralNonLinearFormulation>

Individual:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#IntegralityPropertyInd>
Types:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#IntegerData>

Individual:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#LinearFormulationInd>
Types:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#LinearFormulation>

Individual:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#StrictlyPositiveLowerBoundInd
>
Types:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#StrictlyPositiveLowerBound>

Individual:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#ZeroLowerBoundInd>

Types:
<http://www.semanticweb.org/ci/fl/ontologies/2017/6/OM.MProp.owl#NonNegativeLowerBound>

Manchester syntax for the Mathematical Properties Ontology (OM.MProp) (Part 3 of 3)

Bibliography

[1] R. L. Ackoff. *Scientific Method Optimizing Applied Research Decisions*. John Wiley & Sons, Inc., 1962.

[2] R. L. Ackoff. *The Art of Problem Solving*. Wiley-Interscience Publication, 1978.

[3] R. L. Ackoff. The future of operational research is past. *Journal of the Operational Research Society*, 30(2):93–104, 1979.

[4] R. L. Ackoff. Resurrecting the future of operational research. *Journal of the Operational Research Society*, 30(3):189–199, 1979.

[5] R. L. Ackoff, S. K. Gupta, and J. S. Minas. *Scientific Method, Optimizing applied research decisions, 3rd.* John Wiley & Sons, Inc., 1967.

[6] R. L. Ackoff and M. W. Sasieni. *Fundamentals of Operations Research*. John Wiley & Sons, Inc., 1968.

[7] R. Aljafari. Distributed model management systems: A proposal for an ontology-based approach. In *Midwest Association for Information Systems (MWAIS)*, 2009.

[8] K. J. Arrow, S. Karlin, and H. Scarf. *Studies in the Mathematical Theory of Inventory and Production*, chapter 2: The Nature and Structure of Inventory Models, pages 16–36. Stanford University Press, 1958.

[9] W. J. Bell, L. M. Dalberto, M. L. Fisher, A. J. Greenfield, R. Jaikumar, P. Kedia, R. G. Mack, and P. J. Prutzman. Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. *INFORMS*, 13(6):4–23, 1983.

BIBLIOGRAPHY

[10] T. Bhrammanee and V. Wuwongse. Oddm: A framework for model bases. *Decision Support Systems*, 44:689–709, 2008.

[11] D. C. Blair. The data-document distinction revisited. *The DATA BASE for Advances in Information Systems*, 37(1):77–96, Winter 2006.

[12] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, 1984.

[13] A. Campbell, L. CLarke, A. Kleywegt, and M. Savelsbergh. The inventory routing problem, May 1997.

[14] P. Checkland. From optimizing to learning: A development of systems thinking for the 1990s. *Journal of the Operational Research Society*, 36(9):757–767, 1985.

[15] P. Checkland. Model validation in soft systems practice. *Systems Research*, 12(1):47–54, 1995.

[16] P. Checkland. *Systems Thinking, Systems Practice*. John Wiley & Sons, LTD, 1999.

[17] P. Checkland. Researching real-life: Reflections on 30 years of action research. *Systems Research and Behavioral Science*, 27:129–132, 2010.

[18] P. Checkland and J. Poulter. *Learning for action: a short definitive account of soft systems methodology and its use for practitioner, teachers, and students*. John Wiley & Sons, Inc., 2007.

[19] P. Checkland and J. Scholes. *Soft Systems Methodology in Action*. John Wiley & Sons, LTD, 1999.

[20] C. W. Churchman, R. L. Ackoff, and E. L. Arnoff. *Introduction to Operations Research*. John Wiley & Sons, Inc., 1957.

[21] A. A. Cire, J. N. Hooker, and T. Yunes. Modeling with metaconstraints and semantic typing of variables. *INFORMS Journal on Computing 28(1)*, 28(1):1–13, 2016.

[22] K. Corral, D. Schuff, G. Schymik, and R. St.Louis. Enabling self-service bi through a dimensional model management warehouse. In *Twenty-first Americas Conference on Information Systems, Puerto Rico*, 2015.

[23] National Research Council. *Artificial Intelligence in Mathematical Modeling*. 1991.

[24] E. Cox. *Fuzzy Logic for Business and Industry*. Charles River Media, 1995.

[25] A. Dahanayake and B. Thalheim. Co-evolution of (information) system models. In *Enterprise, Business-Process and Information Systems Modeling Lecture Notes in Business Information Processing*. SpringerLink, 2010. http://www.springerlink.com/content/g07n811r383m7224/.

[26] A. V. Deokar, O. F. El-Gaya, N. Taskin, and R. Aljafari. An ontology-based approach for model representation, sharing and reuse. In *Americas Conference on Information Systems (AMCIS)*, 2008.

[27] A. V. Deokar and O. F. El-Gayar. Decision-enabled dynamic process management for networked enterprises. *Inf Syst Front*, 13:655–668, 2011.

[28] A. V. Deokar and O. F. El-Gayar. On semantic annotation of decision models. *Inf Syst E-Bus Manage*, pages 93–117, 2013.

[29] D. Dolk. *Structured Modeling and Model Management*, chapter 5, pages 63–86. Springer, 2010.

[30] C. M. Eastman. Cognitive processes and ill-defined problems: a case study from design. In *International Joint Conference on Artificial Intelligence*, 1969.

[31] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York, 1972.

[32] D. Gasevic et al. *Model Driven Engineering and Ontology Development, 2nd edn.,*, chapter Ontologies, pages 45–80. Springer-Verlag Berlin Heidelberg, 2009.

[33] J. Hurwitz et. al. *Cognitive Computing and Big Data Analytics*. Wileys, 2015.

[34] J. R. Evans. *Creative Thinking In the Decision and Management Sciences*. South Western Publishing Co., 1990.

[35] J. R. Evans. Ms/or: Improving problem solving through creative thinking. *Interfaces*, 22(2):87–91, 1992.

[36] R. Fourer. On the evolution of optimization modeling systems. *Doumenta MATH.*, pages 377–388, 2012.

[37] R. Fourer. *Encyclopedia of Operations Research and Management Science*, chapter Algebraic Modeling Languages for Optimization, pages 43–51. Springer Science, 2013.

[38] E. Fragniere and J. Gondzio. Optimization modeling languages. Technical report, Department of Mathematics and Statistics, The University of Edinburgh, 1999.

[39] M. Gagliardi and C. Spera. Toward a formal theory of model integration. *Annals of Operations Research*, 58:405–440, 1995.

[40] S.I. Gass. *Encyclopedia of Operations Research and Management Science*. Springer, 2013.

[41] A. M. Geoffrion. An introduction to structured modeling. *Management Science*, 33(5):547–588, 1987.

[42] A. M. Geoffrion. Computer-based modeling environments. *European Journal of Operations Research*, 41:33–43, 1989.

[43] A. M. Geoffrion. The formal aspects of structured modeling. *Operations Research*, 37(1):30–51, 1989.

[44] A. M. Geoffrion. Integrated modeling systems. *Computer Science in Economics and Management*, 2:3–15, 1989.

[45] A. M. Geoffrion. Reusing structured models via model integration. *IEEE*, 073-1129:601–611, 1989.

[46] A. M. Geoffrion. Sml: A model definition language for structured modeling. Technical report, Working Paper No.378. Western Management Science Institute, University of California, Losangeles. (Revised) August, 1990.

[47] A. M. Geoffrion. The sml language for structured modeling. Technical report, Working Paper No.378. Western Management Science Institute, University of California, Los Angeles, 1990.

[48] A. M. Geoffrion. Fw/sm: A prototype structured modeling environment. *Management Science*, 37(12):1513–1538, 1991.

[49] A. M. Geoffrion. The sml language for structured modeling: Levels 1 and 2. *Operations Research*, 40(1):38–57, 1992.

[50] A. M. Geoffrion. The sml language for structured modeling: Levels 3 and 4. *Operations Research*, 40(1):58–75, 1992.

[51] A. M. Geoffrion. Structured modeling: Survey and future research directions, May 20, 1996.

[52] A. M. Geoffrion. An informal annotated bibliography on structured modeling. Interactive Transactions of ORMS, June 1999.

[53] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Aspects of mathematical modeling related to optimization. *Applied Math. Modeling*, 5:71 –83, April 1981.

[54] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. 1981.

[55] H. J. Greenberg. A bibliography for the development of an intelligent mathematical programming system. *Annals of Operations Research*, 65(1):55–90, 1996.

[56] F. H. Gregory. Cause, effect, efficiency and soft systems models. *Journal of the Operational Research Society*, 44(4):333–344, 1993.

[57] F. H. Gregory. Soft systems models for knowledge elicitation and representation. *Journal of the Operational Research Society*, 46(5):562–578, 1995.

[58] F. H. Gregory and S. P. Lau. Logical soft systems modeling for information source analysis-the case of hongkong telecom. *Journal of the Operational Research Society*, 50(2):124–137, 1999.

[59] R. Heyer. Understanding soft operations research: The methods, their application and its future in the defence setting. Technical report, Australian Government, Department of Defence, Defence Science and Technology Organisation, 2004.

[60] J. H. Holland, K. F. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Process of Inference, Learning, and Discovery*. The MIT Press, 1986.

[61] S. Howick, , and F. Ackermann. Mixing or methods in practice: Past, present and future directions. *European Journal of Operational Research*, 215:503–511, 2011.

[62] Y. Hu, J. Xiao, G. Rong, and X. Hu. A structured discrete event system specification (devs) model representation based on extended structured modeling. In *Proceedings of the 2014 Winter Simulation Conference*, 2014.

[63] M. C. Jackson. *Systems Approaches to Management*. Kluwer Academic/Plenum Publishers, 2000.

[64] F. R. Jacobs and R. Chase. *Operations and supply chain management, the core, 3Ed.* McGraw-Hill/Irwin, 2012.

[65] M. Kac and S. M. Ulam. *Mathematics and Logic*. 1968.

[66] J. Kallrath, editor. *Modeling languages in mathematical optimization*. Kluwer Academic Publishers, 2004.

[67] M. Kang, G. P. Wright, R. Chandrasekharan, R. Mookerjee, and N. D. Worobetz. The design and implementation of or/sm: A prototype integrated modeling environment. *Annals of Operations Research*, 72:211–240, 1997.

[68] D. Kendrick and R. Krishnan. A comparison of structured modeling and gams. *Computer Science in Economics and Management*, 2:17–36, 1989.

[69] C. J. Khisty. Soft-systems methodology as learning and management tool. *J. of Urban Planning and Development*, 121(3):91–107, 1995.

[70] H. Kim. An xml-based modeling language for the open interchange of decision models. *Decision Support Systems*, 31:429–441, January 2001.

[71] C. A. C. Kuip. Algebraic languages for mathematical programming. *European Journal of Operational Research*, 67:25–51, 1993.

[72] R. Kujansuu and M. Lindvist. Efficient algorithms for computing s-invariants for predicate/transition nets. In *Proceeding of the 5th European Workshop on Applications and Theory of Petti Nets, Aarhus University*, 1984.

[73] G. Laporte L. C. Coelho, J. F. Cordeau. Thirty years of inventory-routing. CIRRELT homepage, September 2012.

[74] J. Lamp. Using petri nets to model weltanschauung alternatives in soft systems methodology. In *In Proceedings of the Third Australian Conference on Requirements Engineering (pp. 91-100).*, 1998.

[75] E. H. Y. Lim and R. S. T. Lee J. N. K. Liu. *Knowledge Seeker - Ontology Modelling for Information Search and Management, A Compendium*. 2011.

[76] J. Ma. *Optimization Services (OS)*. PhD thesis, Department of Industrial Engineering and Management Sciences. Northwestern University, 2005.

[77] J. Ma and D.Zhou. An object-oriented approach to structured modeling. In *Information Resources Management Association International Conference*, 1998.

[78] J. Mingers. Soft or comes of age - but not everywhere! *Omega*, 39:729–741, 2011.

[79] J. Mingers and L. White. A review of the recent contribution of systems thinking to operational research and management science. *European Journal of Operational Research*, 207:1147–1161, 2010.

[80] C. Minkowitz. Formal process modeling. *Information and Software Technology*, 35(11/12):659–667, 1993.

[81] P. Mitra and G. Wiederhold. *An Ontology-Composition Algebra*, chapter 5, pages 93–113. Springer, 2004.

[82] G. Nadler. Human purposeful activities for classifying management problems. *OMEGA*, 11(1):15–26, 1983.

[83] K. Tandekar O. F. El-Gayar. An xml-based schema definition for model sharing and reuse in a distributed environment. *Decision Support Systems*, 43:791–808, 2007.

[84] RJ Ormerod. Justifying the methods of or. *Journal of the Operational Research Society*, 61:1694–1708, 2010.

[85] A. Paucar-Caceres. Operational research, systems thinking and development of management sciences methodologies in us and uk. *Scientific Inquiry*, 9(1):3–18, 2008.

[86] A. Paucar-Caceres. Mapping the changes in management science: A review of 'soft ' or/ms articles published in omega. *Omega*, 38:46–56, 2011.

[87] M. A. Pinsky and S. Karlin. *An Introduction to Stochastic Modeling*. 2011.

[88] G. Polya. *How To Solve It: A New Aspect of Mathematical Method*. Princeton University Press, 1945.

[89] S. K. Probert. The epistemological assumptions of the (main) soft system methodology advocates. In *1994 International System Dynamics Conference*, 1994.

[90] Proceedings of the Twenty-Third Annual Hawaii International Conference. *Towards a Logical Reconstruction of Structured Modeling*, volume 3. IEEE, 1990.

[91] J. R. Quinland. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[92] A. Reisman and M. Oral. Soft systems methodology: A context within a 50-year retrospective of orms. *Interfaces*, 35(2):164–178, 2004.

[93] M. Reynolds and S. Holwell, editors. *Systems Approaches to Managing Change: A Practical Guide*. Springer, 2010.

[94] T. L. Saaty and J. M. Alexander. *Thinking with Models, Mathematical Models in the Physical, Biological and Social Sciences*. Pergamon, 1981.

[95] J. S. Sagoo and J. T. Boardman. Towards the formalization of soft systems models using petri net theory. *IEE Proc.-Control Theory Appl.*, 145(5):463–471, 1998.

[96] S. Sarkar, A. Dong, and J. S. Gero. Learning symbolic formulations in design optimization. *Design Computing and Cognition*, pages 533–552, 2008.

[97] H. A. Simon. The structure of ill structured problems. *Artificial Intelligence*, 4:131–201, 1973.

[98] J. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole, 1999.

[99] F. Stapel. *Ontology-Based Representation of Abstract Optimization Models for Model Formulation and System Generation*. PhD thesis, Paderborn University, 2016. DS & OR Lab Department of Business Informatics Faculty of Business and Economics University of Paderborn.

[100] B. Thalheim. Towards a theory of conceptual modeling. In *Advances in Conceptual Modeling - Challenging Perspective*. SpringerLink, 2009.

[101] Y. C. Tsai. Model integration using sml. *Decision Support Systems*, 22:355–377, 1998.

[102] W. Ulrich. Operational research and critical systems thinking – an integrated perspective part 1: Or. as applied systems thinking. *Journal of the Operational Research Society*, advance online publication, 14 Dec 2011:1–20, 2011.

[103] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[104] R. Bauke van der Meer. *OR Modeling for Public Sector Performance Measurement*. PhD thesis, University of Strathclyde, Glasgow, 2008.

[105] S. R. Watson, J. J. Weiss, and M. L. Donnell. Fuzzy decision analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):1–9, 1979.

[106] E. W. Weisstein. Decision problem. From MathWorld – A Wolfram Web Resource.

[107] B. Wilson. *Soft Systems Methodology: Conceptual Model Building and its Contribution*. John Wiley & Sons, LTD, 2001.

[108] B. Wilson and C. C. Berg, editors. *Efficiency of Manufacturing Systems*. Plenum Press, 1983.

[109] I. S. Yeoman. *The Development of a Conceptual Map of Soft Operational Research Practice*. PhD thesis, Napier University, 2004.

[110] L. A. Zadeh. Fuzzy sets. *Information and Control*, 1965.

[111] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems Science and Cybernetics*, SMC-3(1):28–44, 1973.

[112] Z Zhu. After paradigm: why mixing-methodology theorising fails and how to make it work again. *Journal of the Operational Research Society*, 62:784–798, 2011.

Vitae

Choat Inthawongse, son of Dr.Kosol and Sunun (Rungratpattana) Intawongse was born in Muang, Cholburi province, Thailand. He graduated high-school from Assumption College Sriracha (ACS) in 1997. He received his Bachelor degree in Industrial Engineering from Kasetsart University, Sriracha Campus, Cholburi, Thailand in 2001. He earned a Master degree in Engineering Management (International Program) from Burapha University, Cholburi, Thailand in 2004. He was a Royal Thai Scholar in 2007 for being a recipient of the Thai Government Science and Technology Scholarship for pursuing his doctoral study at Lehigh University.

He worked as an Industrial Engineer at Kinco Inter Tech (Thailand) Co., Ltd. during his college years. He attended a Young Investment Family (YIF batch 17) organized by the Industrial Finance Corporation of Thailand (IFCT) after graduation. He became a teaching assistant in the department of Industrial Engineering at Ramkhamhaeng University in 2001, at the same time he was pursuing his graduate school study. Afterward, he worked as a lecturer in 2004. He served as an assistant dean for research affair from 2005-2008, at the same time he served as a chair of the Engineering Management Program. He ran the engineering research center from 2007-2008. He came to Lehigh University in June 2008.

He was a volunteer at the Engineering Institute of Thailand under H.M. the King patronage. He was a committee for the Young Chapter of Engineering Institute of Thailand in 2006 - 2007. He was serving as a sub-committee of the network development and information systems for the Industrial Engineering section at the same time period.