JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche
Fakultät

Inconsistencies in Decision Modeling

DISSERTATION

zur Erlangung des akademischen Grades

Doktor

im Doktoratsstudium der

Technischen Wissenschaften

Eingereicht von:
DI (FH) Alexander Nöhrer

Angefertigt am:
Institut für Systems Engineering und Automation

Beurteilung:
Univ.-Prof. Dr. Alexander Egyed, M.Sc. (Betreuung)
Univ.-Prof. Dr. Armin Biere

Linz, Juni, 2012

Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Dissertation ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, am 1. Juni 2012

Alexander Nöhrer

Danksagung

An dieser Stelle möchte ich mich bei all den Menschen bedanken, die mich im Laufe meines Studiums unterstützt haben. Ich möchte mich bei den Lektoren aller Vorlesungen sowie Übungen bedanken, denn im Nachhinein betrachtet konnte ich aus jeder Vorlesung und Übung etwas mitnehmen.

Besonders bedanken möchte ich mich bei meinem Dissertationsbetreuer Alexander Egyed. Zum Einen für seine Zeit und sein für mich wertvolles Feedback während der Erstellung dieser Arbeit. Zum Anderen dafür dass er mir diese Arbeit ermöglicht hat und mit Rat und Tat im Zuge der Entstehung zur Seite stand und immer für Diskussionen offen war. Auch bei meinem Zweitbeurteiler Armin Biere möchte ich mich bedanken für seine Hilfe. Weiters will ich mich beim Österreichischen FWF für die Finanzierung der Forschungsarbeit für meine Dissertation bedanken (P21321-N15).

Bei meinen Freunden möchte ich mich für das entgegengebrachte Verständnis bedanken, dass ich während des Studiums im fernen Oberösterreich, nicht immer Zeit für sie hatte.

Abschließend möchte ich mich bei den wichtigsten Menschen in meinem Leben bedanken: meiner Familie. Danke, dass ihr Alle an mich geglaubt und mir dieses Studium ermöglicht habt.

Allen anderen die sich bis jetzt noch nicht angesprochen fühlen, will ich ebenfalls meinen Dank aussprechen, ich habe euch nicht vergessen.

Kurzfassung

Inkonsistenzen sind Teil des täglichen Alltags in der Software Entwicklung. Aus diesem Grund ist es wichtig wie mit Inkonsistenzen umgegangen wird. Obwohl in einigen Software Entwicklungsdomänen das Leben mit Inkonsistenzen unterstützt wird, ist es in der Domäne der Entscheidungsfindung nicht erlaubt – entweder wird es unterbunden oder man wird dazu gezwungen Inkonsistenzen sofort zu beheben. Hauptsächlich Schuld daran sind die in dieser Domäne üblicherweise eingesetzten Reasoning Engines, da sie standardmäßig nicht mit Inkonsistenzen umgehen können, bzw. keine sinnvollen Resultate mehr liefern sobald mit inkonsistentem Wissen geschlussfolgert wird.

Diese Dissertation greift das Problem mit Inkonsistenzen während der Entscheidungsfindung umgehen zu können auf. Zwei Ansätze werden vorgestellt, einer der dabei hilft Inkonsistenzen zu vermeiden, indem den Entscheidungsträgern möglichst viel Freiheit gegeben wird in der Reihenfolge in der Entscheidungen zu treffen sind, unterstützt jedoch auf sinnvolle Weise um möglichst schnell das Ziel der Entscheidungsträger zu erreichen. Der zweite Ansatz vergleicht verschiedene Strategien um mit Inkonsistenzen während der Entscheidungsfindung umgehen zu können, wobei sich eine Strategie herauskristallisiert, die garantiert, dass bestehende Entscheidungsfindung Automatisierungen auf Basis von Schlussfolgerungen korrekt sind (zu Kosten der Vollständigkeit).

Obwohl der Fokus dieser Dissertation primär auf den Technologien "hinter den Kulissen" ist, wird ein Prototyp vorgestellt mit dessen Hilfe diese visualisiert werden können. Verschiedene Evaluierungsszenarien liefern Einblicke in die Effektivität unserer Ansätze und deren Kosten. Es wird sich zeigen, dass unsere Benutzerführung während des Entscheidungsfindungsprozesses, in Bezug auf die Minimierung des benötigten Aufwands des Entscheidungsträgers, fast optimal ist und der Ansatz schnell genug ist um interaktiv eingesetzt zu werden. Zusätzlich wird sich zeigen, dass das Ausmaß der Unvollständigkeit beim Schlussfolgern beim Leben mit Inkonsistenzen annehmbar ist. Außerdem kann sich das Tolerieren von Inkonsistenzen sogar positiv auf die Behebung jener auswirken.

Die wichtigsten Beiträge dieser Dissertation sind: *i)* Eine einzigartige Perspektive auf die Probleme mit Inkonsistenzen während des Entscheidungsfindungsprozesses; *ii)* Ein Ansatz der es erlaubt korrekt mit SAT-Solvern in der Präsenz von Inkonsistenzen zu schlussfolgern, um so vorhandene Automatisierungen unverändert weiter nutzen zu können; *iii)* Ein Ansatz der den kürzesten Pfad durch eine Reihe von voneinander abhängigen Entscheidungen bestimmen kann; *iv)* Eine Basis für zukünftige Forschung in generischen Modellierungsszenarien; *v)* Eine ausführliche Evaluierung von verschiedenen Aspekten und Technologien die in unseren Ansätzen Verwendung finden.

Abstract

Inconsistencies are a fact of life in software engineering. As a consequence, dealing and managing inconsistencies plays an important role in software engineering. However, while in some software engineering domains there exists support for living with inconsistencies, in the domain of decision-making they usually are disallowed – either by preventing them or requiring them to be fixed right away. This stems mainly from the fact that common reasoning engines used in the decision-making domain do not handle inconsistencies well out of the box, in fact most of them fail to produce usable output after an inconsistency has been discovered.

This thesis tackles the problem of managing and dealing with inconsistencies during decision-making. It presents two approaches, one that helps preventing inconsistencies by giving decision makers the maximum amount of freedom possible, while still providing meaningful guidance to reach one's goals faster. The other approach presents different reasoning strategies for living with inconsistencies during decision-making, where one strategy in particular is very promising because it ensures that decision making automations remain working correctly even after an inconsistency is encountered (at the expense of completeness).

While the focus of this thesis is primarily on the technologies "behind the curtains", it will also present one possibility of how to use and visualize these technologies in a prototype tool. Several evaluation scenarios will provide insight into the effectiveness of our approaches and their respective costs. It will show that our guidance calculation with respect to minimizing user input is nearly optimal and fast enough to be used interactively. It will also show that the cost of living with inconsistencies is acceptable, the level of incompleteness is reasonable and that living with inconsistencies can even have a positive impact on resolving inconsistencies.

The key contributions of this thesis are: *i)* A unique perspective on the problems currently encountered during decision-making involving inconsistencies; *ii)* An approach that allows correct reasoning with SAT-Solvers in the presence of inconsistencies and automations to continue working without any adaptions; *iii)* An approach on how to determine the shortest path through a series of related decisions; *iv)* A basis for further research in general modeling scenarios; *v)* An extensive evaluation of the different aspects and techniques used in our approach.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"...zwar hab ich ka Ahnung wo ich hinfahr, aber dafür bin i g'schwinder durt!" — ...although I have no idea where I am going, I will be there faster!

– *Gerhard Bronner*

This quotation in many ways describes the motivation and goals of this thesis. Guiding and supporting software engineers and end-users to meet their demands faster, even if they do not know exactly what their requirements are at the beginning. This thesis presents ideas and realizations thereof, for managing inconsistencies during decision-making, which are stumbling blocks in reaching ones goals and resolving them can be time consuming tasks to finish. Since the most precious resource during software engineering is the software engineer, this resource should be used as little as possible, but as much as necessary. This work demonstrates that any software engineering task that involves decision-making can be optimized to ensure just that: to automatically guide the software engineer such that some inconsistencies can be avoided, while still keeping the human interaction to a minimum; and dealing with the remaining inconsistencies without disturbing the workflow.

1.1 Background and Motivation

Model driven engineering (MDE), also known as its approaches model driven development (MDD) or model driven architecture (MDA), has come far since its beginnings and has many advantages over traditional software engineering methods [1, 2]. However, lack of adequate and suitable tool support is still an issue [1]. This lack of adequate tool support partially stems from the fact that models contain inconsistencies, and inconsistencies in models imply the presence of defects. For the software engineer, the main benefit of tolerating inconsistencies is the ability to continue working despite this presence of defects. This is useful when it is neither obvious how to fix the inconsistencies nor important to do so right away. Indeed, many inconsistencies can be tolerated. 20 years ago, Balzer wrote that "software systems, especially large ones, are rarely consistent (...) yet no principled basis exists for managing the development during the periods of inconsistency". He argued that inconsistencies should be detected and communicated to the developers; however, developers should not be hindered in continuing their work despite the presence of inconsistencies.

In model-driven software engineering, it is state-of-the-practice to allow inconsistencies [3, 4]. Modeling tools tend to indicate inconsistencies, but do not force developers to fix them right away [5]. This is especially important since software models typically contain many defects. However, in many other software engineering domains, tolerating inconsistencies is disallowed (i. e., usually by preventing decisions that cause inconsistencies). And there are good reasons for disallowing inconsistencies. First and foremost, many reasoning engines are rendered partially or fully useless in the presence of inconsistencies. Even if the reasoning engines were able to function (instead of failing outright), the implications on the quality of the results are typically not understood (clearly, we cannot expect a reasoning engine to compute correct results in the presence of inconsistent, aka erroneous input). This is a severe problem because as Balzer said, inconsistencies are a fact of (software engineering) life and to date reasoning engines support many vital automations such as analyzing properties of systems, understanding the effects of design decisions, helping configure products, etc.

While the tolerance to inconsistencies is state-of-the-practice in model-driven software engineering, the issues that surround inconsistencies are largely ignored in many other domains but of essential importance. It is far more important to fix the cause of an inconsistency (defect) than just the symptoms (the inconsistencies themselves). After all, the goal of an engineer is not just to resolve one inconsistency at a time but in the end to get a consistent model with all defects having been identified and resolved. While inconsistencies can be resolved by fixing the underlying defects, we must recognize that those defects may also have caused additional inconsistencies at other locations. In certain situations this could be reversed, meaning that several defects cause the same inconsistency. An example for such a situation would be a requirement change in an already consistent model. This requirement change should lead to several model changes where initial changes are likely to conflict with other existing parts of the model. As a consequence the first change(s) introduce inconsistencies, though these first changes would not be defects because they are the initial steps of a larger requirements change. It follows that the defects must be other model elements that need to be changed. Note that in this context, the term defect may be misleading because propagating a change is not the same thing as fixing a defect; however, the same principles apply and we consider them quite analogous. This basic idea is not entirely new and fairly established in the compiler community dating back as early as 1982 when Johnson and Runciman [6] wrote: *"It is important to distinguish between an error diagnosis and error reporting. Correct error diagnosis must rely upon the programmer as it may depend upon intentions that are not expressed in his program.*

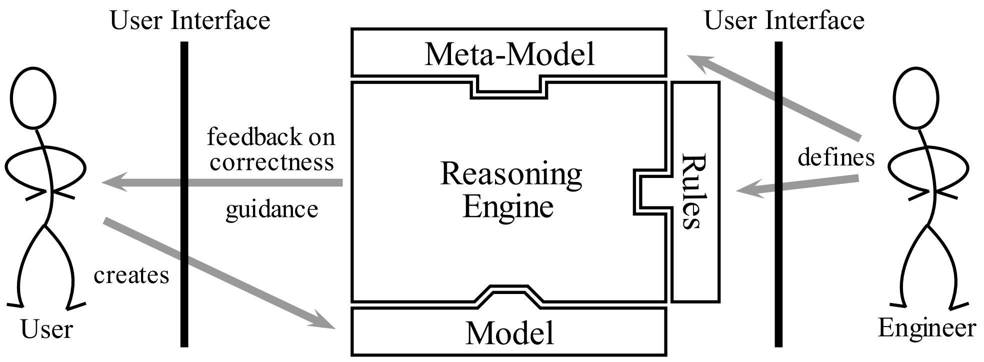


Figure 1.1: Abstraction of a typical modeling scenario.

The compiler's job is correct error reporting using a form and content of reports most likely to help the programmer in error diagnosis. We can compare error reports to the symptoms of a sick patient: the location at which the error is detected is not necessarily its source."

As stated before, we are interested in managing inconsistencies to support users in reaching their goals faster in modeling scenarios. As depicted in Figure 1.1, there coexist two sides to this problem: The user interface aspects of how to communicate certain things to the user in an intuitive and simple way and the technologies / approaches "behind the curtains" – some sort of reasoning engine. Figure 1.1 in addition points out that there are different use cases involved in modeling scenarios. On the one side we have engineers that define the meta-models and additional rules that have to be followed. On the other side we have users of modeling tools, which of course also can be engineers, that create / modify / work with models.

Although both sides are important, we are focusing primarily on the technologies "behind the curtains". It is our belief, that user interfaces can only be as helpful as the information provided to them and therefore depend heavily on technologies / approaches "behind the curtains". In addition improvements on the reasoning engine can benefit different use cases.

While originally the research started out to be for modeling scenarios in general, we decided to narrow our area of research and limit the number of variables, by concentrating on one specific modeling scenario namely decision-making. The main reasons being:

(i) All available decisions encountered during decision-making are determined by the decision model behind, making it a "simple" modeling scenario in comparison to for example modeling with the Unified Modeling Language (UML) [7]. Except for decision model evolution, making decisions by selecting choices (model) from a correct decision model (meta-model) is a more or less trivial task with no surprises. Each model element has predefined choices in the decision model to choose from, as a result each model element can only be modified, but not deleted and new model elements cannot be created. While this is a simplification of more

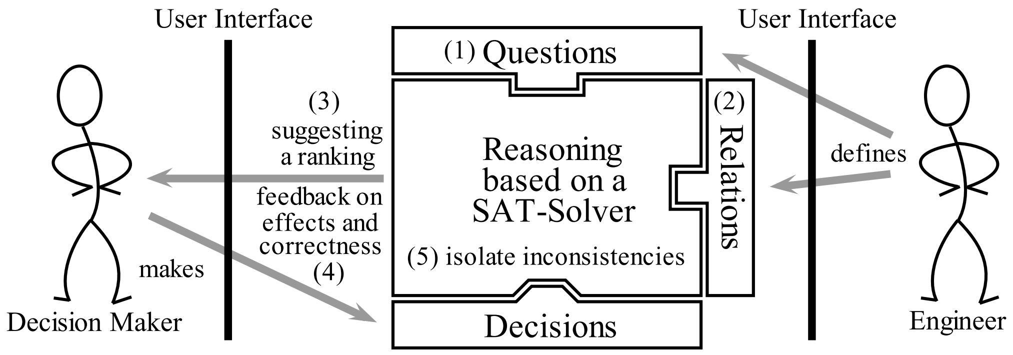


Figure 1.2: Decision-making approach overview.

general modeling scenarios, the same problems rise in this simpler environment of decision-making when it comes to inconsistencies.

(ii) Concepts developed in this "simple" modeling scenario should be adaptable to more general modeling scenarios and provide insights in what is possible and useful from a user's perspective.

(iii) Concepts that cannot be utilized due to computational complexity in general modeling scenarios, are easier to realize in the decision-making scenario. We thus can be more creative without being too concerned about computational complexity and identify solutions, that are worth pursuing in general modeling scenarios.

1.2 Outline of our Approach

As mentioned so far, our intend is to help users to avoid inconsistencies and help them dealing with the ones that cannot not be avoided. Our approach is based on using a SAT-Solver for reasoning purposes to realize just that. SAT-Solvers are reasoning engines that check Boolean formulae, or to be more specific, clauses in conjunctive normal form (CNF) for their satisfiability [8]. While encoding decision models and reasoning about them with SAT-Solvers is not new, we will show how to combine techniques and approaches that exist in the SAT community with information about the history of decisions made, to our advantage.

Figure 1.2 depicts an overview of our approach. First an engineer defines questions (1) and relations between those questions (2). With those serving as input, our reasoning engine based on a SAT-Solver then ranks the questions with respect to requiring minimal user input and suggests this ranking to the decision maker (3). Furthermore this ranking gets updated with every decision made by the user. Additionally the reasoning engine provides the user with feedback on effects of decisions as well as explanations why certain decisions are not allowed any longer (4). If the decision maker insists on making inconsistent decisions, our reasoning engine will isolate those inconsistencies from reasoning (5), allowing correct reasoning and other automations to continue working correctly, like for instance the ranking of questions (3) and the feedback to the decision maker (4). However, this is not restricted to those automations, but in fact is true for any kind of SAT-based automations.

In addition to our techniques “behind the curtains”, we will also present our configurator tool in Chapter 6 to illustrate how these techniques could be used for guidance in the user interface. The main ideas are to show almost all questions at once in a sort of a word cloud, bigger fonts representing a higher ranking, leaving it up to the decision maker to choose a question to answer. When a question gets selected, the list of choices how to answer this question is shown. For each possible answer either the effect of this answer can be viewed or the explanation why it is not allowed anymore. After making a decision the word cloud adapts itself to the new ranking, questions that are already answered will additionally show those answers. Inconsistent decisions are visually different from consistent ones and although the decision maker still can see how those questions were answered, these answers will be ignored in the reasoning engine. If the decision maker changes decisions so that inconsistencies are resolved, either automatic changes to inconsistent decision are visualized or the original decisions get accepted and are again used for reasoning.

1.3 Project History

The research for this thesis was done in the course of the the Austrian Science Fund (FWF[1]) project P21321-N15. The project started in March 2008 and is expected to end in 2012. Generally speaking the project is about researching methodologies on how to better understand the cause of inconsistencies and how to propose solutions for fixing them. One part of obtaining this knowledge is to understand design model changes and how they affect other models, or even the same model. Once these basic things are understood, resolving inconsistencies should become easier and better tailored to users' needs – which is vital to the future of software modeling and software engineering in general.

Publications that were created during our research covering different aspects and their relation to this thesis, will be explained next in detail. The publications are listed in the order they were finally published, which does not reflect the order, the work was done in.

(P1) *Conflict Resolution Strategies during Product Configuration* [9] describes our initial ideas and vision of how to avoid and deal with inconsistencies that rise in decision-making scenarios. It further provides an overview of possible ways of how to handle inconsistencies during decision-making and is the basis for Chapter 3 and 4.

(P2) *Utilizing the Relationships Between Inconsistencies for more Effective Inconsistency Resolution* [10], while not an integral part of this thesis, this paper already

[1]http://www.fwf.ac.at

looks beyond decision-making and takes the first step in the domain of UML modeling, showing the parallels of inconsistencies in UML modeling and decision-making. Some of the background information and motivation in Section 1.1 stem from this paper.

(P3) *C2O: A Tool for Guided Decision-making* [11] is a short tool paper, that shows off our decision-making tool, which is the basis for Chapter 6.

(P4) *Positive Effects of Utilizing Relationships between Inconsistencies for more Effective Inconsistency Resolution: NIER Track* [12] is the continuation of our work in (P2) [10] and is about first results in the domain of UML modeling, which will be briefly introduced in Chapter 7.

(P5) *Optimizing User Guidance during Decision-Making* [13] covers the aspect of how to avoid inconsistencies during decision-making, by giving the end-users the freedom to make decisions in whatever sequence they prefer. On the other hand it proposes a guidance strategy, based on reducing the needed user input to a necessary minimum, that allows guiding users without imposing a sequence, that even can benefit from users temporarily ignoring the guidance. Many details of this paper can be found in Chapter 5 and evaluations thereof in Chapter 7.

(P6) *Managing SAT Inconsistencies with HUMUS* [14] deals with the applicability of the HUMUS (High-level Union of Minimal Unsatisfiable Sets) to explain, tolerate and fix inconsistencies during decision-making and also discusses its general applicability in other domains. A detailed explanation of what HUMUS exactly is and how it works will be given in Chapter 5.

(P7) *A Comparison of Strategies for Tolerating Inconsistencies during Decision-Making* [15], covers the aspect of how to tolerate and live with inconsistencies that could not be avoided. It also provides a detailed analysis of how the presence of inconsistencies influences reasoning, in terms of completeness and correctness, and compares different techniques that could be used for tolerating inconsistencies in SAT-based reasoning. Many details of this paper can be found in Chapter 5 and evaluations thereof in Chapter 7.

1.4 Contributions of this Thesis

In general, this thesis contributes to the research area of decision-making and areas that involve decision-making. Guided decision-making is quite common whenever software engineers desire to restrict the space of possible answers: towards the end user (e. g., installation wizards or e-commerce) or towards fellow software engineers (e. g., product lines or process configurations). As such besides contributing to software engineering research, it may also benefit practitioners that are looking for new solutions and want to adopt them in their products.

The key contributions of this thesis are:

(i) A unique perspective on the problems currently encountered during decision-making combined with a vision and approach, that may inspire other researchers to think about the needs of the most important resource in software engineering namely the software engineer. The most important need being the freedom to make decisions and resolve inconsistencies the way the engineer prefers to. Not the other way around, where the tools need them to make decisions in a certain sequence and need them to fix inconsistencies right away (cf. Chapters 3, 4, and 5).

(ii) An approach that allows correct reasoning with SAT-Solvers in the presence of inconsistencies at the expense of marginally more incomplete reasoning, thus also allowing any existing automations to work correctly in the presence of inconsistencies without adaptions needed (cf. Chapter 5).

(iii) An approach on how to determine the near optimal path through a series of related questions to any possible configuration, without knowing in advance which configuration it is going to be (cf. Chapter 5).

(iv) A basis for further research in general modeling scenarios (cf. Chapter 7). We also will provide an assessment on the feasibility, based on preliminary results, of applying these techniques to general modeling scenarios in Section 7.5.

(v) An extensive evaluation of the different aspects and techniques used in our approach (cf. Chapter 7). The results of the evaluation are valuable for researchers and practitioners alike, as they show how effective these techniques are and also provide insights in what is possible.

1.5 Structure of this Thesis

This thesis is structured as follows: Chapter 2 will explain preliminaries and state-of-the-art, which serve as basis for this thesis. In Chapter 3 the world and terminology as we see it will be described in detail, including an illustrative example that will be used throughout this thesis to explain and highlight certain problems and approaches. The chapter will be round off with a description of the problems we wanted to tackle with our research. Chapter 4 will then provide insight into our vision and goals of how to tackle these problems. It will be followed by Chapter 5, which will give an overview of our approach and provide detailed explanations of different aspects, that were needed to realize our visions and goals. In Chapter 6 we will present a end-user tool, that was developed to show off the “behind the curtains” technologies we developed, in order to guide the user through the process of decision-making while dealing with inconsistencies. Representative case studies combined with extensive evaluations, for the different aspects we were aiming at with our goals, will then be used to demonstrate the feasibility and usefulness of our approach in Chapter 7. After that we will discuss similarities and differences of our approach with related work in Chapter 8. Finally, this thesis will be concluded in Chapter 9 with a summary of our contributions and an outlook to future work.

Chapter 2

Preliminaries

“ Before everything else, getting ready is the secret of success. ”
– *Henry Ford*

In this chapter we will provide the preliminaries needed for our approach and state-of-the-art technologies our approach is build on. In the next Section 2.1 a brief overview of the reasoning techniques relevant to our approach will be given, focusing on the application and results of those techniques instead of explaining the inner workings in great detail. After that we will provide an introduction into product lines in Section 2.2. And the chapter will be round off in Section 2.3 with a short introduction to guidance.

2.1 Reasoning

Consistency management cannot be realized without some sort of reasoning engine. Model checking in general depends largely on some sort of reasoning that checks specifications and rules, as already pointed out in Figure 1.1. However, reasoning can be used for more than just the verification of rules. It is our intention to provide users with guidance to avoid inconsistencies and support in resolving them, as mentioned in Section 1.1. So in addition to model checking we plan to use reasoning techniques for vital automations such as analyzing properties of systems, understanding the effects of design decisions, helping configure products, etc. Model checking and consistency management respectively can be done in different ways, and those of importance for our approach will be discussed next.

2.1.1 Consistency Management

One big part of supporting users in modeling scenarios is consistency management. In order to get correct models according to some specifications, it is important to enforce those specifications to get usable models.

First of all, to resolve inconsistencies they have to be detected. However, the knowledge if the whole model or a single constraint is consistent, is not enough to produce fixes. As Nentwich et al. for example stated in [4], it is important that trace links from the inconsistency to the model element(s) in question exist. In their work they propose to use first-order logic to express consistency rules and are able to provide trace links between inconsistent elements. Performance also is an issue when checking for consistency and approaches like the incremental consistency checking approach by Egyed [5] addresses this issue.

After being detected, developers should not be hindered in continuing their work despite the presence of inconsistencies [3]. To live with inconsistencies is a very important concept, because it gives engineers the freedom to ignore errors for the time being, this is especially valuable if there are several engineers working on conflicting parts of the same model. However, at some point those inconsistencies have to be resolved, preferably with the support of automated techniques.

For generating fixing or repair actions several approaches exist. On the one hand, Xiong et al. propose writing additional "fixing procedures" for each constraint, in order to produce fixes when needed [16]. On the other hand Nentwich et al. describe in their work [17] a method for generating interactive repairs from first-order logic formulae – the same formulae that they already used to detect inconsistencies [4]. Another approach described by Egyed et al. in their paper [18] shows how to generate choices for fixing an inconsistency without having to understand such formulae, which can be complex in case consistency rules are written in programming languages. These approaches look at other model elements already defined in the model and use them as choices. This generated choices are then reduced by looking at the impact of each choice [19, 20] and removing those that would cause additional inconsistencies. This can be problematic because during refactoring it could be necessary to introduce temporarily new inconsistencies [3], as a result dismissing fixing actions, because they cause new inconsistencies, could be counterproductive.

Despite the considerable progress in research for fixing inconsistencies, to the best of our knowledge no approach looks at more than one inconsistency at a time. However, the need for a more "global" approach during consistency checking itself is demonstrated by Sabetzadeh et al. in [21] but not used for fixing yet. Additionally Nentwich et al. already stated in their work [17], that one of the biggest challenges is not to look at one single inconsistency but to look at inconsistencies from a more "global" point of view. In our opinion this notion is also an important one and key to improved automated techniques for inconsistency resolution as described in our workshop paper [10].

While all of the approaches above have in common that they work towards consistency management in UML models, they apply different reasoning techniques including first-order logic, truth maintenance systems, constraint based reasoning, etc. Even though all of those techniques are viable in decision-making related reasoning, our research focused first on constraint satisfaction problems (CSP) [22] and transitioned later on towards Boolean satisfiability problems (SAT problems) [8].

$$CNF\,(low-level): \quad \underset{Literals}{\underset{\backslash\ /}{\overbrace{(a \vee b)}^{\overset{Clause}{/}}}} \wedge \underset{Literals}{\underset{\backslash\ \ /}{\overbrace{(b \vee \neg c)}^{\overset{Clause}{|}}}} \wedge \underset{Literals}{\underset{\backslash\ /}{\overbrace{(d \vee e)}^{\overset{Clause}{\backslash}}}}$$

$$Assumptions\,(high-level): \quad \neg a, \quad \neg b, \quad \neg c$$

Figure 2.1: Overview of SAT terminology.

2.1.2 Boolean Satisfiability Problems

Boolean Satisfiability Problems, also known as SAT problems, are relatively old [8] and simple problems. Next a short introduction to the terminology based on [23] is given and summed up in a small example in Figure 2.1: SAT problems are defined in conjunctive normal form (CNF) which is a conjunction of clauses. One *clause* is a disjunction of *literals* which are Boolean variables. *Assumptions* are assignments for literals that constrain the assignment possibility of a literal to either true or false. SAT solvers produce one of two results, either a CNF is satisfiable (SAT) or unsatisfiable ($UNSAT$) – SAT meaning that there exists an assignment for all literals such that the CNF evaluates to true, UNSAT meaning that such an assignment does not exist. If a problem is UNSAT it can be because of either an inconsistency in the clauses which would be *low-level*, or an inconsistency because of assumptions which would be *high-level*.

Even though SAT problems are relatively simple in their structure and SAT solvers are only able to differentiate between either SAT or UNSAT, the reasoning possible with them can be quite complex. By analyzing the CNF formula or changing assumptions one can find out if literals always have to be either true or false to make the formula satisfiable, e. g. given the CNF $(a \vee b) \wedge (a \vee \neg b)$ one can deduce that a always has too be true to make this CNF satisfiable. This can be helpful when calculating the effect of other assumptions, e. g. to answer questions like if one makes a certain assumptions which other assumptions have to be made in order for the formula to be satisfiable. There also exists a range of concepts to help with unsatisfiable formulae, like for example the concepts of a minimal unsatisfiable set (MUS), a minimal correcting set (MCS) and a maximum satisfiable set (MSS). Those concepts will be explained in Chapter 5 with the help of the example given in Section 3.1.3. For now it is sufficient enough to know that in the SAT domain concepts to help with inconsistencies exist.

2.2 Product lines

The concept of product lines was first proposed by Parnas in 1976 [24] as "program families" based on the work of Dijkstra [25]. The probably most cited definition reads: *"A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"* [26]. In essence product line models capture the variability in product families. There exist different approaches of how to capture this variability, the most common ones being a feature-oriented approach (cf. Section 2.2.1) which emerged around 1990 [27] and a decision-oriented approach (cf. Section 2.2.2) which emerged in 1991 [28]. Even though different modeling approaches are used, in the end their purpose is on the one hand to capture the variability and on the other hand enable reasoning with the models (cf. Section 2.2.3) and the derivation of products.

According to Linden et al. [29] software product line engineering (SPLE) distinguishes between two life-cycles: development for reuse and development with reuse. On the one hand development for reuse, typically also called domain engineering, focuses on building up a product line infrastructure of reusable artifacts, also called assets (these assets encompass all artifacts relevant throughout software development, e. g., requirements, architecture, code, documentation, test cases, etc. [30]). On the other hand development with reuse, also called application engineering, means developing individual products using this infrastructure. After the development of these products, one needs to derive them from the product line.

This product derivation is the process of configuring a product, therefore also often called the configuration process or more general the decision-making process. In most cases configuring a product involves one or more end users also called decision makers, that translate the requirements they have on the product into decisions how the variability should be handled. Typically this configuration process is not done completely manually, but rather with the support of configurator tools. There exist a number of different configurator tools for product lines [31, 32, 33, 34, 11, 35], that support different kind of approaches and differ on the level of support the provide to users. However, all configurators share the common goal of guiding the user(s) to a valid configuration / product. Due to all this effort needed, SPLE only pays off if the infrastructure can be reused efficiently and often, so that the development cost of building the infrastructure do not outweigh the costs of building the needed application with alternative methods [36]. For a broader and more detailed overview of relevant work and other aspects of SPLE, it is recommended to consult Rabiser's PhD thesis [30] or one of the several surveys on variability modeling [37, 38]. For a detailed comparison of feature modeling vs. decision modeling we would like to refer to Czarnecki et al. [39].

2.2.1 Feature Models

According to Kang et al. [27] feature modeling is about identifying common and distinctive aspects, the so-called features, between related systems in a domain. These features

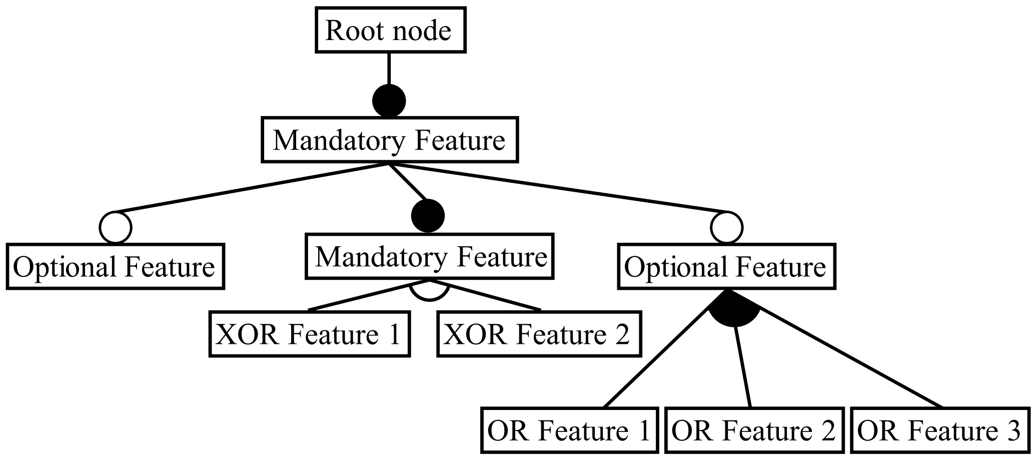


Figure 2.2: Example of a basic feature model (in the notation proposed by Czarnecki [40]).

are user-visible aspects or characteristics of the domain. The probably most cited definition reads: *"A feature is a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among systems in a family"* [40].

Feature models are hierarchical tree-based models that follow a common structure [27, 40, 41, 42, 43], as depicted in Figure 2.2, slightly different notations that exist, will not be described here. In the tree, features are represented as nodes, while the edges describe relations between the features. The root node of the tree represents the domain concept being modeled and is included in every product derived from the feature model. Feature models have mandatory features, decorated with filled circles, that are common to all related products represented by the feature model. Furthermore they have optional features, decorated with hollow circles, that can either be part of the target product or not. And there are grouped features, that represent cardinality relations in the form of $[n, m]$, where n is the minimum number of features that have to be selected and m is the maximum number of features that are allowed to be selected and $1 \leq n \leq m$ with m being smaller than or equal to the number of features in the group. The selection rules of features contained in a feature group apply as soon as the parent feature or one of the child features is selected to be in the product. The most common feature groups, an exclusive-or-group and an or-group, are depicted in Figure 2.2. An exclusive-or-group always has a cardinality of $[1, 1]$, so that only exactly one feature has to be selected from the feature group. Such exclusive-or-groups are visualized as hollow arcs. On the other hand an or-group always has a cardinality of $[1, \#numberOfFeatures]$, so that at least one has to be selected and as many as are available. Such or-groups are visualized as filled arcs. In addition to such implicit constraints (stemming from the tree structure) additional explicit constraints can be expressed in Boolean logic as cross-tree constraints, e. g. $(XOR\ Feature\ 1 \rightarrow OR\ Feature\ 3) \wedge (XOR\ Feature\ 2 \rightarrow \neg OR\ Feature\ 2)$ meaning that if $XOR\ Feature\ 1$ is selected $OR\ Feature\ 3$ has to be selected (*requires*) and if

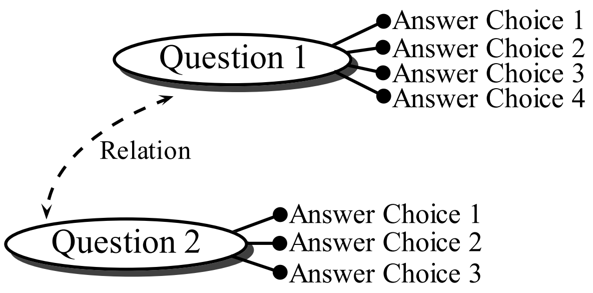


Figure 2.3: Example of a basic decision model.

XOR Feature 2 is selected *OR Feature* 2 must not be selected (*excludes*).

Sometimes the tree structure and such additional constraints might not be enough to model all the requirements, e. g. certain features might need additional information for the initialization of artifacts. To solve this issue feature models can be extended with attributes [43], meaning that to each selected feature different attributes can be added which can be constrained too, e. g. an integer between 1 and 5 with a default value of 1.

2.2.2 Decision Models

The basic idea behind decision modeling in product lines, e. g. [28, 44, 32], is to separate concerns and focus on the user perspective for the decision model. As a consequence, in contrast to feature models decision models are not used to represent commonalities. However, decision models always have an asset model behind where of course common assets are represented and trace links exist between the decision and asset model. While hierarchy is essential for feature models, it is only secondary for decision models. In other words decision models rely only on decisions that need to be made to derive a specific product form the product line [30]. According to Rabiser [30] decisions are typically represented in form of questions with a defined set of possible answers. However, in contrast to features, questions are not limited to Boolean types. This is depicted in Figure 2.3 with questions as ellipses, various choices per questions connected via solid lines to the ellipses and a relation shown as a dashed line. The sets of answer choices are similar to feature groups and the selection of answers can also be tied to cardinality constraints. There can also exist relations between those questions as shown in Figure 2.3. Such relations are similar to cross-tree constraints in feature models, but allow a different level of granularity: relations can be expressed in Boolean logic between the questions itself or between the answer choices of questions or a mixture of both.

As mentioned before decision models only represent parts of a product line where decisions need to be made. As a consequence the decision model is not enough for representing a whole product line infrastructure. This issue can be addressed differently, for example the DOPLER (Decision-Oriented Product Line Engineering for effective

Reuse) variability modeling approach [30, 32, 45] uses three separate, yet connected, models to represent a product line [30]:

(i) A domain model that is specific to the domain. Managing different kinds of assets in a product line relies on the precise definition of their characteristics in a domain-specific meta-model. Building such a meta-model requires knowledge about a domain and / or organization settings and specifics. The meta-model defines the types of assets to be included in the product line (e. g., component, service, documentation, setting, etc.), their attributes (e. g., file name, location, etc.), and the possible relationships between different asset types. The DOPLER variability modeling approach allows defining asset relationships with an arbitrary name and cardinality based on eight basic relationship types: inclusion, exclusion, parent, child, abstraction, implementation, predecessor, and successor [46, 47].

(ii) Asset models that are created on the basis of the domain-specific meta-model and describes the concrete reusable elements in a product line, their attributes, and the dependencies among them. The meta-model denotes which assets can be created and which attributes and dependencies can be modeled. Asset models can often be created semi-automatically if product line development does not start from scratch and core assets already exist. For example, call dependencies defined in existing system configuration files can be utilized to automatically derive requires dependencies among software components that reflect the underlying technical restrictions.

(iii) Decisions represent the variations in a product line variability model. Decisions have a name and a type (i. e., Boolean, String, Double, or Enumeration) and are represented by questions. Arbitrary additional decision attributes can be defined. A decision type denotes the range of allowed values to be set on the decision by answering its question (the possible choices). Validity conditions allow restricting the range of allowed values. Decisions can depend on each other hierarchically (e. g., a decision needs to be taken before another one) and / or logically (e. g., taking a decision changes the value of another one). Variability stemming from technical or business considerations is expressed using decisions. Decision models link external variability (relevant for customers, sales people, or marketing staff) with internal variability (relevant for engineers). A decision model is a graph where the nodes represent decisions and the edges represent the relationships between them. Decision models reduce modeling complexity as they represent variability at a higher level of abstraction. For instance, variability mechanisms in the asset base can be changed without an effect on the decisions. Assets and decisions are connected using inclusion conditions that denote when a particular asset will be part of a product

2.2.3 Reasoning with Product Lines

Besides using reasoning techniques to derive products, they can also be used to check for inconsistencies within the product line model before the derivation process even begins. Such analysis can range from simple things, e. g. detecting anomalies [43] like for instance dead features and false optional features, to reasoning about non-functional properties of the product lines [48]. Basically product line analysis is checking for anything that could cause problems during product derivation.

Once the system is modeled, these effects can be calculated with SAT-Solvers (see Section 2.1.2) and used for eliminating conflicting choices [49]. Translating configuration problems / feature models / decision models to SAT problems is solved by e. g. [50, 42] and will not be discussed here.

SAT-based reasoning is state-of-the-practice [34, 33]. It has several primary uses: SAT reasoning is used *i)* to validate products [43], *ii)* to find viable alternative solutions if a product is not valid [51] or auto complete partial products [34], and *iii)* to provide guidance during the configuration process [49, 13]. To validate a product one call to the SAT solver is sufficient. For the other uses several SAT solver calls are necessary with different assumptions to find out if combinations of assumptions are valid. So basically in these cases the SAT solver is used as an oracle and the reasoning process is based on querying this oracle.

2.3 Guidance

Guidance is foremost about tool support to get the users to their goal [52, 53]. Thus guidance "behind the curtains" consists of two major components: *i)* automation and *ii)* additional information to help users make decisions. Although providing as much guidance as possible may seem like a good idea, this is not always the case as suggested by Nimwegen et al. [54]. They make the argument that too much guidance triggers the behavior in users to completely trust the guidance and stop thinking for themselves even in cases where the guidance fails. Vicente also makes a similar argument in [55] using the example of automatically set VHS recorder clocks which very often fails miserably in the most unusual ways. He presents a good example for how automations can fail, although one would think eliminating the human is a good thing if you think about improving safety or productivity, basically arguing that automations are not foolproof and one should not rely solely on them.

But guidance is not just about the technology "behind the curtains", the visualization plays the most crucial part, but can only be as good as the information it gets. this visual guidance should not be to overwhelming by providing too much information, as Miller already pointed out in 1956 [56] seven plus or minus two is the magic number of the humans capacity for processing pieces of information at a time. An overview of different visualization techniques – like for example different types of trees, flow charts, venn diagrams, etc. – and their applicability to product lines is given by Pleuss et al. [57].

Chapter 3

Background

“ The whole is more than the sum of the parts. ”

– *Aristotle*

In this chapter we will provide the background needed for our approach, including a definition of terms and examples we will use and a detailed problem description. In the next Section 3.1 we will define our notation for decision models and explain an example used throughout this thesis. After that we will provide a definition of terms we use for the process of decision-making in Section 3.2. And the chapter will be round off in Section 3.3 were we will give a detailed problem description.

3.1 Decision Models

We work with decision models, since in our opinion they provide a more intuitive interface during decision-making in comparison to feature models. Not being bound to an hierarchical tree structure also simplifies the development of such models. As mentioned in Section 2.2.2 decision models consist of questions, choices of how to answer these questions, and relations between questions and / or choices, which will be broke down in the next sections. Formally speaking, $Questions$ denotes the set of all questions and $Relations$ denotes the set of all relations contained in a decision model. Thus we can define a decision model as a tuple of questions and relations:

$$DecisionModel = (Questions, Relations)$$

3.1.1 Questions and Choices

Questions and their choices of how to answer them are an essential part of decision models. Each $Question \in Questions$ has a set of choices (depending on the question those could range from different labels, to numbers, to Boolean values, etc.) which we call *Choices*. Note that the same *Choices* set can be used for different questions. Furthermore each question has a cardinality constraint determining how many choices (at a minimum and at a maximum) can be selected as an answer to the question. Thus we can define a question as a tuple of *Choices* and a cardinality constraint:

$$Question = (Choices,\ [n, m] \mid n, m \in \mathbb{N},\ n \leq m \leq |Choices|)$$

To simplify things, we also define the following syntactic equivalence:

$$Question = Choices \equiv Question = (Choices,\ [1, 1])$$

Examples for question definitions are: $NumberOfBackups = \{x \mid x \in \mathbb{N},\ x < 5\}$, $Color = \{red,\ blue,\ green\}$, $Extras = (\{A,\ B,\ C\},\ [1, 2])$, $Name = \{s \mid s\ is\ a\ \texttt{String}\}$.

Selecting an answer for question is about selecting as many choices from it, as specified by the cardinality constraint. Thus the following conditions apply to the *Answer* set: $Answer \subseteq Choices$ and $n \leq |Answer| \leq m$. Assigning this *Answer* set to the *Question* is what we call a *Decision*:

$$Decision = Question \leftarrow Answer$$

To allow for better readability in text, we also define the following syntactic equivalences:

$$Question \leftarrow Answer \equiv Question_{Answer}$$

We also define two special *Answer* sets we call *undecided* and *irrelevant*, to have a convenient way of writing about unnecessary decisions and decisions not yet made. Additionally we allow *Answer* sets containing only one element to be written without the curly brackets. Examples for decision are: $Color_{undecided}$, $Extras_{\{A, B\}}$, $Color_{red}$, $NumberOfBackups_3$, $Extras_{irrelevant}$.

3.1.2 Relations

As already mentioned relations are between questions and / or choices. We use three different types of relations: *i)* relevancy relations, *ii)* constraint relations, and *iii)* CNF relations. In our experience these threes types of relations are sufficient enough to express any dependency or constraint in decision-making, and constraints that cannot be expressed with the help with of CNF relations can also not be handled by our reasoning engine which is based on a SAT solver.

Relevancy relations are used to express relations of relevance between questions. The intention is to model such dependencies in a convenient way by, for example, specifying that it makes no sense to ask questions about the configuration of software components if they are not even used, hence irrelevant. A relevancy relation maps from one source question *Source* to a set of target questions *Targets* and is itself a set of implications from choices of the source question $Choices_{Source}$ to either *relevant* or *irrelevant*, formally we define a relevancy relation the following way:

$$\begin{aligned}&(Source \rightarrow Targets)_{\,RelevancyRelation} = \{\\&\quad (choice \Rightarrow state) \mid choice \in Choices_{Source},\ state \in \{relevant,\ irrelevant\}\\&\}\end{aligned}$$

In order for it to be a well-formed relevancy relation at least one choice should map to *relevant* and at least one other choice to *irrelevant*. An example for the definition of a relevancy relation is:

$$\begin{aligned}&UseSubversion = \{yes,\ no\}\,,\ \ldots\\[1em]&(UseSubversion \rightarrow \{Address,\ User,\ Password\})_{\,RelevancyRelation} = \{\\&\quad (yes \Rightarrow relevant)\,,\\&\quad (no \Rightarrow irrelevant)\\&\}\end{aligned}$$

Constraint relations are used to express constraints of one choice onto choices of other questions, like for example *requires* and *exclude* dependencies. A constraint relation maps from one source question $Source$ to one target question $Target$ and is itself a set of implications from choices of the source question $Choices_{Source}$ to sets of choices of the target question $Choices_{Target}$. Formally we define a constraint relation the following way:

$$\begin{aligned}&(Source \rightarrow Target)_{\,ConstraintRelation} = \{\\&\quad (choice \Rightarrow C) \mid choice \in Choices_{Source},\ C \subseteq Choices_{Target}\\&\}\end{aligned}$$

An example for the definition of a constraint relation is:

$$\begin{aligned}&Device = \{Phone,\ Desktop,\ Laptop\}\\&Application = \{Accounting,\ Inventory,\ CRM,\ Sales\}\\[1em]&(Device \rightarrow Application)_{\,ConstraintRelation} = \{\\&\quad (Phone \Rightarrow \{Inventory\})\,,\\&\quad (Desktop \Rightarrow Choices_{Application} \setminus \{Sales\})\,,\\&\quad (Laptop \Rightarrow \{CRM,\ Sales\})\\&\}\end{aligned}$$

CNF relations are used to express more complex relations that are not covered with the simpler constraint relations, but in essence are also constraint relations. Instead of having a source and target questions like the other relations, CNF relations are formulated in CNF with decisions used as variables:

$$CNFRelation = \bigwedge \left(\bigvee \left(x \mid x \in \{Decision,\ \neg Decision\}\right) \right)$$

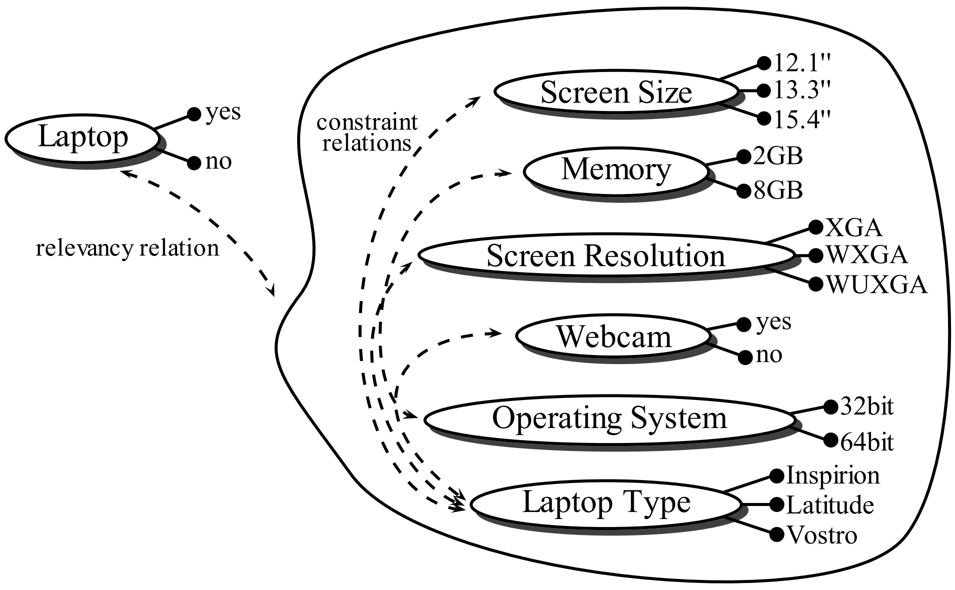


Figure 3.1: Illustrative decision model for buying a laptop.

An example for the definition of a CNF relation is:

$$Color = \{red,\, blue,\, green\}$$
$$Extras = (\{A,\, B,\, C\}\,,\, [1,2])$$

$$CNFRelation = \left(Color_{red} \vee \neg Extras_{\{A,\,B\}}\right) \wedge \left(Color_{blue} \vee Extras_{\{A,\,C\}} \vee Color_{red}\right)$$

3.1.3 Illustrative Example

As an illustrative configuration example, shown in Figure 3.1, we will be using an excerpt from a real e-commerce decision-oriented product line. This decision model was reverse engineered from the the DELL website (during February 2009), a complete version of the model was also used in the evaluation and it can be downloaded from the C2O website[1]. This excerpt was chosen because while still being relatively small and compact, it allows to highlight and explain the different facets of the problems we want to tackle with this thesis. The illustration includes seven questions which are defined next:

[1]www.sea.jku.at/tools/c2o

$$\begin{aligned}
Laptop &= \{yes,\ no\}\\
Screen\,Size &= \{12.1'',\ 13.3'',\ 15.4''\}\\
Memory &= \{2GB,\ 8GB\}\\
Screen\,Resolution &= \{XGA,\ WXGA,\ WUXGA\}\\
Webcam &= \{yes,\ no\}\\
Operating\,System &= \{32bit,\ 64bit\}\\
Laptop\,Type &= \{Inspirion,\ Latitude,\ Vostro\}
\end{aligned}$$

Thus the question set of the decision model tuple is: $Questions = \{Laptop, Memory, Screen\,Size, Screen\,Resolution, Webcam, Operating\,System, Laptop\,Type\}$. But a decision model is not complete without its relations of which five are indicated in Figure 3.1 and defined next:

$$\begin{aligned}
&RR = (Laptop \rightarrow \{Screen\,Size,\ Memory,\ Screen\,Resolution,\\
&\quad Webcam,\ Operating\,System,\ Laptop\,Type\})_{RelevancyRelation} = \{\\
&\quad (yes \Rightarrow relevant),\\
&\quad (no \Rightarrow irrelevant)\\
&\}
\end{aligned}$$

$$\begin{aligned}
&CR_1 = (Screen\,Size \rightarrow Laptop\,Type)_{ConstraintRelation} = \{\\
&\quad (12.1'' \Rightarrow \{Inspirion,\ Latitude\}),\\
&\quad (13.3'' \Rightarrow \{Latitude,\ Vostro\}),\\
&\quad (15.4'' \Rightarrow \{Inspirion,\ Latitude,\ Vostro\})\\
&\}
\end{aligned}$$

$$\begin{aligned}
&CR_2 = (Memory \rightarrow Operating\,System)_{ConstraintRelation} = \{\\
&\quad (2GB \Rightarrow \{32bit,\ 64bit\}),\\
&\quad (8GB \Rightarrow \{64bit\})\\
&\}
\end{aligned}$$

$$\begin{aligned}
&CR_3 = (Screen\,Resolution \rightarrow Laptop\,Type)_{ConstraintRelation} = \{\\
&\quad (XGA \Rightarrow \{Latitude,\ Vostro\}),\\
&\quad (WXGA \Rightarrow \{Inspirion,\ Latitude,\ Vostro\}),\\
&\quad (WUXGA \Rightarrow \{Latitude,\ Vostro\})\\
&\}
\end{aligned}$$

$$\begin{aligned}
&CR_4 = (Webcam \rightarrow Laptop\,Type)_{ConstraintRelation} = \{\\
&\quad (yes \Rightarrow \{Inspirion,\ Vostro\}),\\
&\quad (no \Rightarrow \{Latitude,\ Vostro\})\\
&\}
\end{aligned}$$

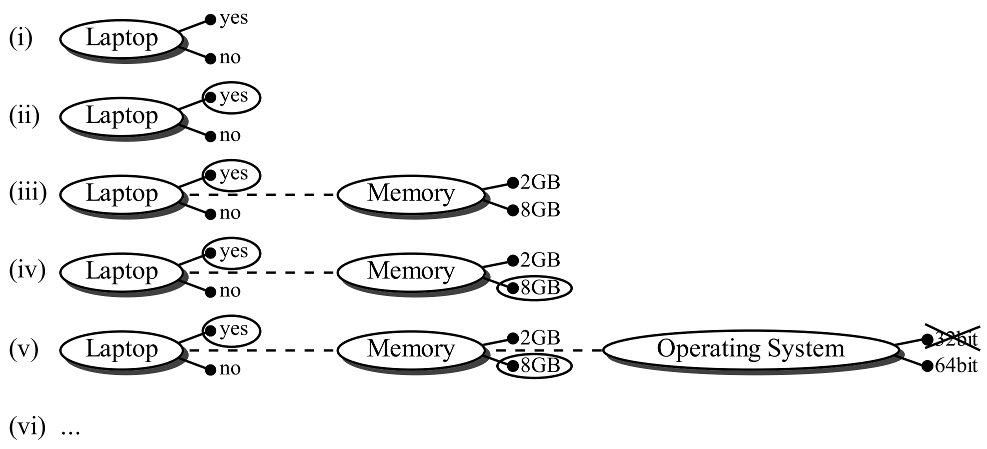


Figure 3.2: Graphical illustration of the decision-making process with our example.

Thus the relation set of the decision model tuple is: $Relations = \{RR, CR_1, CR_2, CR_3, CR_4\}$, making the decision model: $Example = (Questions, Relations)$.

3.2 Decision-Making

The process of decision-making involves one or more *decision makers* and a decision model. The decision model's questions are always unanswered at the beginning, giving us for our example from Section 3.1.3 the initial state: $State = \{Laptop_{undecided}, Screen\,Size_{undecided}, Memory_{undecided}, Screen\,Resolution_{undecided}, Webcam_{undecided}, Operating\,System_{undecided}, Laptop\,Type_{undecided}\}$, which is also our *problem space*. The goal of decision-making is to get a decision (other than undecided) for each question in the decision model, resulting in a *configuration* of a system, our *solution space*. We define a configuration to be a complete set of decisions other than undecided for all questions. For example, the configuration $C_1 = \{Laptop_{no}, Screen\,Size_{irrelevant}, Memory_{irrelevant}, Webcam_{irrelevant}, Operating\,System_{irrelevant}, Laptop\,Type_{irrelevant}, Screen\,Resolution_{irrelevant}\}$ is a valid configuration for deciding upon a Laptop in our illustration. In fact for our example there exist 70 valid configurations, but there exists an infinite number of configurations that are invalid (given that one can assign choices that are not valid for the question). For example, the configuration $IC_1 = \{Laptop_{yes}, Screen\,Size_{12.1''}, Screen\,Resolution_{XGA}, Laptop\,Type_{Latitude}, Operating\,System_{32bit}, Memory_{8GB}, Webcam_{no}\}$ is invalid because it violates relation CR_2 that specifies that $Memory_{8GB}$ is only valid in combination with $Operating\,System_{64bit}$.

To reach this goal of a complete valid configuration, the process of decision-making is about guiding the decision maker through the questions and let the her make one decision at a time. After each decision the problem space becomes smaller and the solution space more complete. This process is visualized in Figure 3.2, first the decision maker selects a question in our case the *Laptop* question (i). Given the choices to either buy or not to buy a laptop, she decides on $Laptop_{yes}$ (ii). After that as a

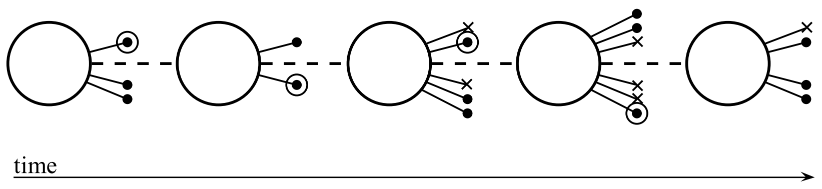


Figure 3.3: Graphical abstraction of the decision-making process over time.

Table 3.1: Illustration of the decision-making process with our example in tabular form.

Decision	$1^{st}q$		$2^{nd}q$		$3^{rd}q$		
	u	s	u	s	u	s	
$Laptop_{yes}$	**1**	**1**		**1**		**1**	
$Laptop_{no}$		0		0		0	
$Memory_{2GB}$				0		0	
$Memory_{8GB}$			**1**	**1**		**1**	
$Operating\,System_{32bit}$				0		0	
$Operating\,System_{64bit}$				1	**1**	1	

q…question, u…user decisions, s…derived state, 0…false, 1…true

next question $Memory$ is selected (iii). After making the decision $Memory_{8GB}$ (iv), the question $Operating\,System$ is chosen (v). At this point one can observe for the first time the typical guidance a configurator tool offers, namely that the $32bit$ choice is disabled because of the relation CR_2, that states that in order to have 8GB of memory one must use a 64bit operating system. This process then continues in a similar fashion as indicated (vi). An abstraction of this process is visualized in Figure 3.3, with the big circles representing questions, small circles representing decisions, the small dots representing still viable choices and the small x-es representing invalid choices at this point in the configuration. This schematic diagram depicting the time-line of a decision-making process will be used next to describe different problems that come with inconsistencies.

The decision-making process could also be visualized in tabular form as shown in Table 3.1. For the first question the user decided $Laptop_{yes}$ which is indicated in the column $1^{st}q$, u, the effect and resulting state of this decision is shown in the column $1^{st}q$, s. This is repeated for the other questions, where user decisions are bold and decisions belonging to questions that already have been answered are grayed out.

3.3 Inconsistencies during Decision-Making

In an ideal world there would not be any inconsistencies, however inconsistencies can rise easily during decision-making. The first and foremost problem would be if the decision model itself already contained inconsistencies, as mentioned in Section 2.2.3 there

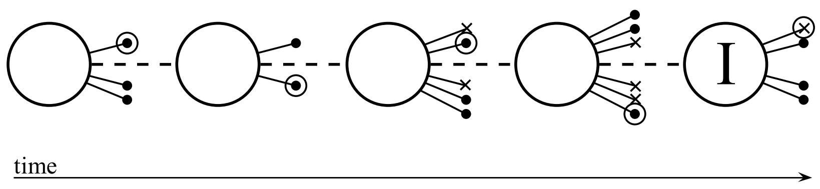


Figure 3.4: Inconsistencies during decision-making.

exist techniques how to detect such inconsistencies. For the moment we assume that the decision model by itself is consistent. Since the model is correct any inconsistencies raised during decision-making must stem from the conflicting decisions by the decision maker. As such the decision maker is also the first to realize there is an inconsistency when a desired choice for a question is no longer available, as depicted in Figure 3.4 the desired choice is already disabled and the inconsistency detected (as indicated by the capital I). We call the point in time when an inconsistency is detected *failure*. So what possibilities are there to react to such a failure or even prevent it? However, before we go into detail, we must distinguish two basic cases:

(i) No valid configuration exists: this happens when the decision maker configures a product that in this manner does not exist. Eventually, after encountering many inconsistencies and not finding a suitable alternative this case can be identified. The only solution is to adapt the decision model, changes of assets included if necessary, to satisfy a customer's need.

(ii) A valid configuration exists, but some of the questions need to be answered differently. Eventually after changing some answers a configuration is found, that maybe does not fulfill all requirements 100%, but at least is good enough so that the decision maker can live with it.

In the next sections we will focus on discussing the second case and always assume the decision model to be correct and immutable. For the purpose of illustration we present one interesting inconsistent decision-making process in Table 3.2. In this illustration technically after the fourth decision the configuration process could be stopped, since a decision was provided $\{Laptop_{yes}$, $Screen\,Size_{12.1''}$, $Memory_{8GB}$, $Screen\,Resolution_{XGA}\}$ or derived $\{Webcam_{no}$, $Operating\,System_{64bit}$, $Laptop\,Type_{Latitude}\}$ for all questions. However what if the user is not satisfied with some of the derived decisions as illustrated and wants $Webcam_{yes}$? Strategies on how to prevent and manage such inconsistencies are discussed next.

3.3.1 Strategies for Preventing Inconsistencies

For preventing inconsistencies we have identified three basic strategies:

(i) The easiest solution from the decision maker's point of view to facilitate him with a less complex decision model with respect to relations. With less complex

Table 3.2: Configuration progression of the example given in Figure 3.1.

Decision	1st q		2nd q		3rd q		4th q		5th q	
	u	s	u	s	u	s	u	s	u	
$Laptop_{yes}$	**1**	**1**		**1**		**1**		**1**		
$Laptop_{no}$		0		0		0		0		
$Screen\ Size_{12.1''}$			**1**	**1**		**1**		**1**		
$Screen\ Size_{13.3''}$				0		0		0		
$Screen\ Size_{15.4''}$				0		0		0		
$Memory_{2GB}$						0		0		
$Memory_{8GB}$					**1**	**1**		**1**		
$Screen\ Resolution_{XGA}$							**1**	**1**		
$Screen\ Resolution_{WXGA}$								0		
$Screen\ Resolution_{WUXGA}$								0		
$Webcam_{yes}$								0	**1?**	
$Webcam_{no}$								1		
$Operating\ System_{32bit}$						0		0		
$Operating\ System_{64bit}$						1		1		
$Laptop\ Type_{Inspirion}$								0		
$Laptop\ Type_{Latitude}$								1		
$Laptop\ Type_{Vostro}$				0		0		0		

q...question, u...user decisions, s...derived state, 0...false, 1...true

relations the possibility to cause inconsistencies decreases drastically. However, providing products with features combined in any possible combination is not always easy and to just rely on this mechanism for preventing inconsistencies is not a good idea.

(ii) The second strategy is to provide the decision maker with as many pieces of information as possible, so that the decision can be informed and the decision maker is hopefully aware of the consequences for further decisions. This approach is largely what the research in decision support systems is all about [58]. This strategy has one main disadvantage, namely that in large decision models with complex relations that include combination effects it may not be enough to provide this information to the decision maker for her to understand the complications that could rise with future decisions.

(iii) The third strategy is simple but in our opinion a rather effective one. Usually configurator tools impose a certain sequence onto decision makers, the idea behind it is to guide users through the configuration, for instance the most logical or shortest way and so on. However, every decision maker has different priorities, when configuring a product, that cannot be foreseen. In such cases allowing the decision maker to deviate from the imposed sequence might be the smartest way

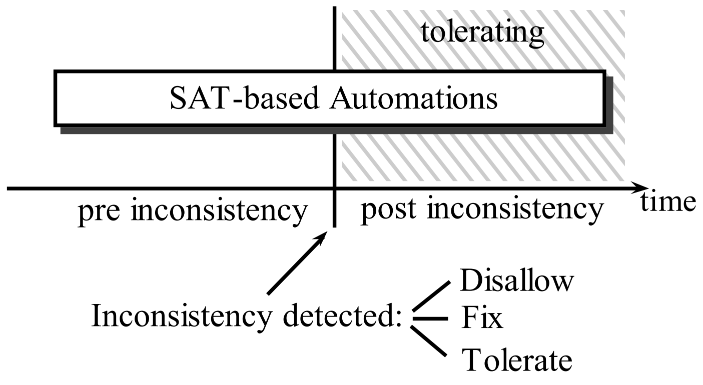


Figure 3.5: How to manage inconsistencies during decision-making.

to prevent inconsistencies because decisions that are subjectively important can be made first, before the choices get constrained through other decisions.

3.3.2 Strategies for Managing Inconsistencies

Dealing with inconsistencies is something that can rarely be avoided during the configuration of complex decision models. Figure 3.5 illustrates the three basic strategies of how to mange inconsistencies and also hints at the problem when faced with inconsistencies and wanting to tolerate them. Any kind of automations, in our case SAT-based automations need to continue working otherwise tolerating inconsistencies is a nuisance and not really that helpful. In this section, we keep the resolving strategies simple and focused on product configuration, but we believe that the basic strategies discussed here also apply to more general user-guided scenarios. The three basic strategies are:

(i) The first strategy is disallowing inconsistencies, which is also the easiest way to handle them. For this strategy to work automations that reason about the effects of decisions have to be in place. So when the decision maker encounters a desired choice that is already disabled, like for example *yes* for the question *Webcam* in Table 3.2, she basically has no other option than to select another choice or start over again and select different choices for earlier questions or start with the *Webcam* question if the configurator tool allows it.

(ii) The second strategy is to fix inconsistencies right away. At the exact moment the decision-maker introduces an inconsistency into the system by selecting a choice that has been eliminated through some relation; a fixing strategy can be applied to return the configuration to a consistent state immediately. Fixing an inconsistency right away ensures that the model stays consistent and never contains an inconsistency (no reasoning in the presence of inconsistencies is necessary). It is fairly simple to realize and handle with reasoning engines, since the knowledge base stays consistent. Different strategies to fix an inconsistency right away will be explained in detail in Section 3.3.2.1.

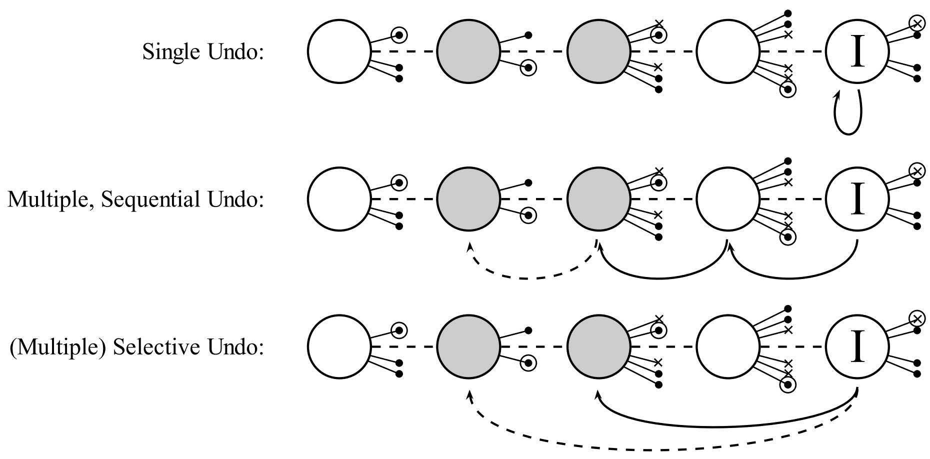


Figure 3.6: Inconsistency fixing strategies.

(iii) The third strategy is to tolerate inconsistencies and fix them later. Tolerating can be easily realized if we stop reasoning until the inconsistencies are fixed. However, as mentioned before, without existing automations working, it is not really that helpful to just stop reasoning. Different strategies to tolerate an inconsistency will be explained in detail in Section 3.3.2.2.

3.3.2.1 Fix Right Away Strategies

To fix an inconsistency right away we can use different strategies like the ones illustrated in Figure 3.6, in the figure decisions involved in the inconsistency besides the one that allowed the system to detect the inconsistency are represented by the shaded circles. However, those decisions contributing to an inconsistency are usually not known. In the case of our illustrative example from Table 3.2 user decisions involved in the inconsistency would be $\{Screen\,Size_{12.1''}, Screen\,Resolution_{XGA}, Webcam_{yes}\}$ and they are all connected via the *LaptopType* question and only one of those decisions needs to be removed in order to be consistent again. If not all involved decisions are known, fixing becomes a more or less random act. Next, the strategies depicted in Figure 3.6 are described in detail:

(i) *Single Undo:* The simplest way to fix an inconsistency and return to normal working mode is to retract the decision that caused the inconsistency as illustrated in Figure 3.6. The decision maker is told to try something else instead. Often this is not desired by the decision maker since she wants the offending choice. In a more general modeling scenario it could also be the case that a different developer is continuing the work on a model she is not completely familiar with; in such a case Undo might not be such a bad idea. In approaches that do not care about the sequence this could also be the solution identified as the minimal solution, since it typically is less effort than changing other decisions involved in the inconsistency. Applied to our example from Table 3.2 this would mean retracting the decision $Webcam_{yes}$ which certainly would resolve the inconsistency but may not be desired.

(ii) *Multiple, Sequential Undo:* Assuming the decision that caused the inconsistency is important to the decision maker and therefore assumed to be correct, the problem must be an earlier decision. To find the root of the problem the simplest way is to retract the given decisions until the desired choice for the most recent decision becomes available. This could also imply retracting decisions that did not contribute to the inconsistency as illustrated in Figure 3.6 (unshaded circles), which is not desirable. In addition to this it could be the case that it is sufficient enough to retract only one of the inconsistent decisions. Multiple, sequential undo would retract the most recent one first which could fix the inconsistency but may not be the desired one. This is also the case in our example, since retracting either $Screen\,Size_{12.1''}$ or $Screen\,Resolution_{XGA}$ would be sufficient to resolve the inconsistency. Which one gets retracted would depend on the sequence the decisions were made in.

(iii) *Selective (Multiple) Undo:* To avoid retracting valid decisions that do not contribute to the inconsistency, the involved decisions need to be identified (at least partially). It does not matter if the identification of contributors is done manually or automatically, but when it is done they can be retracted directly as illustrated in Figure 3.6. This approach helps reducing the needed user input compared to the multiple, sequential undo approach (valid decisions do not have to be made more than once). Nevertheless, in situations where the desired choice is excluded because of the combination of other decisions, it is not that simple. Retracting one decision or the other could be sufficient, however, without further information this cannot be decided automatically. Either all participating decisions or randomly selected among them are retracted, or the decision maker has to be asked which one she wants to retract – a question the decision maker may not be able to answer correctly at this point during the configuration! Again this situation can also be found in our example, since retracting either $Screen\,Size_{12.1''}$ or $Screen\,Resolution_{XGA}$ would be sufficient to resolve the inconsistency, but this cannot be automatically decided. Selecting one of those decisions randomly or both of them is not a desirable solution.

(iv) A special form of a selective (multiple) undo might be to automatically calculate a fix that is "close" to the intent of the decision maker. The work by White et al. [51] or Felfernig et al. [59] solved this problem with respects to a minimal solution, so we do not go into more detail here.

3.3.2.2 Tolerating Strategies

The "fix right away" strategies are thus valid but often not desirable. Instead of fixing an inconsistency right away, it is more beneficial to let the decision maker answer more / all questions. The more information is collected, the better any reasoning works. This additional information may, for example, help with deciding between two alternative options for resolving an inconsistency. It should thus be the decision maker's decision when she wants to resolve an inconsistency. Different strategies to continue the configuration process with an inconsistency are described next:

(i) *Stop Reasoning:* Since reasoning with inconsistencies is hard, the simplest way to continue the configuration process is to let the decision maker continue answering questions without such reasoning. However, without reasoning, the user no longer benefits from automations, e. g. knowledge of how choices are affected by decisions. The negative effect would be that the decision maker is not guided through the remaining questions and as a consequence has to memorize the constraints limiting the product configuration options. This is realistically not possible and unless the decision maker is an expert, this resolution strategy leads to follow-on inconsistencies where the decision maker unintentionally makes additional errors. For any reasonably complex system, asking the decision maker to configure a system without automated guidance is a recipe for failure.

(ii) *Continue Reasoning:* A better choice would be to continue reasoning, while tolerating inconsistencies. This is only possible if the used reasoning technique supports living with inconsistencies, or if the inconsistency is (partially) isolated from reasoning so that the reasoning continues to work with some decisions that are consistent. How the selection for the isolation can be achieved will be explained in Chapter 5. With this approach the decision maker is still guided through the remaining questions and informed about decisions that would cause new inconsistencies.

(iii) *Continue Reasoning with Trust:* Continue reasoning with trust means, that assumptions are made on how much certain decisions provided by the decision maker can be trusted. For example, the decision that introduced the inconsistency is a decision that could be trusted to be important to the decision maker – and perhaps even to be final. After all, if a choice is no longer available and the decision maker insists on selecting that choice then the user states that this choice is a "must have". Obviously, all decisions made earlier that are participating in the inconsistency could thus be considered less trustworthy. Based on this implicit trust (implied through the sequence in which questions were answered), the reasoning would be more complete when continued. In the best case new trusted decisions could even help to resolve the inconsistency by reducing the number of alternative choices for fixing the inconsistency. In addition to the implicit trust, trust could also be based on user queries: As mentioned in the Single Undo fixing strategy, in more general scenarios different users can be involved in making decisions. In such cases it could be interesting to ask the user different questions to get a better feeling of what decisions to trust. These questions could range from high-level questions like: How familiar are you with the given model on a scale from 1 to 5? to low-level questions like: Select the decisions that are important to you (and thus can be trusted) from the list of inconsistent decisions.

Chapter 4

Vision and Goals

“ Any intelligent fool can make things bigger and more complex... It takes a touch of genius – and a lot of courage to move in the opposite direction. ”

– *Albert Einstein*

In this chapter we will clarify our vision and goals for decision-making. For the most part the vision as well as the goals are applicable in more general modeling scenarios. Indeed, looking at the bigger picture, our vision is to provide such guidance not only to decision-making, but also to design modeling and in the future also in traceability management. First we will make the argument why we think automation is generally speaking a good thing in Section 4.1, followed by Sections 4.2 and 4.3 which will contemplate the fine line between too little and too much automation.

4.1 Automation is good!

Generally speaking automation is a good thing, the more can be automated the less work is left to do. In the case of decision-making, automations can be very useful to guide the decision makers and relieve them of making all decisions manually and help them to identify inconsistencies in the decisions and even resolve them.

User guidance during decision-making is perceived to be a straightforward activity where a decision maker answers a set of predefined questions, usually by selecting among their choices. Usually during the decision-making process the information about the relations between questions is not available to the decision-maker. For example, the Dell laptop configurator, from which we used an excerpt as our illustrative decision model in Figure 3.1, hides those relations completely from the user. Without detailed expert knowledge, users are confronted with the exponentially complex task of navigating among interdependent choices and their implications without explanations. It is state-of-the-art to support users by asking questions in a predefined sequence and presenting only those choices of the remaining questions that are still available [45]. For example, after selecting $Operating\,System_{32bit}$ for the laptop, $Memory_{8GB}$ becomes unavailable. Initially all choices are available, these are then incrementally reduced as the user answers questions (decides on a choice). Having this kind of automation is very useful for a non-expert decision maker. This way the decision maker becomes aware of the relations incrementally and does not need to understand the decision model behind it, as long as she does not wish to select choices that are unavailable, then a different kind of automation is needed to explain the inconsistency and to help fixing it. In addition to the state-of-the-art it is our vision to also provide users with automated support in case of managing inconsistencies once they are detected.

As mentioned before, it is state-of-the-art to order the questions to support users, however this is still done mostly manually. The goal of such a sequence can be, for example, to provide a certain flow of questions that belong together semantically or to minimize user input. Rearranging the sequence of questions reduces effort because questions have relations and answering questions makes other questions irrelevant or reduces their choices (e. g., choosing $Operating\,System_{32bit}$ makes asking about the $Memory$ irrelevant in our example because $Memory_{2GB}$ can be derived automatically). For our simple illustration, a user may understand these relations and may be capable of answering the questions in a close to optimal manner without automation. With such a small decision model, it would not even matter if the user answered the questions sub-optimally since it takes very little time to answer them. However, the configuration problem suffers from exponentially increasing configurations and factorial increasing ways of arranging their sequence – a daunting scalability problem. In such a context, a user is no longer capable of optimizing the sequence of questions manually [60]. If the answering of questions takes time (e. g., steel plant manufacturing) then any reduction in the number of questions asked or choices provided will save significant effort. Thus as part of our vision, we plan to also unburden the creators of decision models to have to think about the sequence of questions by automating this process.

4.2 Tools adapting to users, not the other way around!

It was made clear in the last section that automations are generally speaking a good thing, but one has also to be careful. In our opinion sometimes automations go to far or restrict users in such a way that it can be counter-productive. For instance, just think of the auto-correct functions built-in in Microsoft Office, those automations that try to help the user by automatically correcting misspelled words, or automatically capitalize words at the beginning of enumerations. While those automations certainly have their purpose and help many times, at other times they can be quite annoying and changing things that are actually correct. In those cases one either realizes the automatic change right away and takes it back, or realizes it later and wonders what happened or even if it was an own mistake. In such cases the user has to do some extra work and might even adapt his working style to fit the tool's automations.

The message here being, that sometimes automations do not work as intended [55], or do not fit every user. So it is our vision to allow decision makers to ignore our automated guidance without any added effort, because in the end only she knows what she wants. In case of automatically determining the sequence of questions, giving the decision maker such freedom may seem contradictory for example to the goal of minimizing user input. After all, any deviation from our proposed sequence (likely) leads further away from the optimal path. Yet, we must recognize that an arbitrary sequence of questions, even an optimal one, may not be intuitive or appropriate [54] for every user. We should thus not impose but only suggest a particular sequence of questions. Allowing the decision maker to deviate from the suggested sequence, while still trying to suggest the optimal sequence for the remaining questions implies that the user interaction must be optimized incrementally and in real-time, as it is not possible to analyze all possible user interactions in advance (there are too many).

Allowing the decision maker this freedom can already have an impact on preventing inconsistencies. Using the example given in Section 3.1.3, a tool that forces the decision maker to answer the question about the laptop type first, could facilitate inconsistencies later. To elaborate this, for example, if the decision maker has no idea what features a certain laptop type allows to be selected and she actually does not care about the type, she basically has to make a random decision in order to get to the questions she cares about, thus potentially leading to inconsistencies if desired features (e. g. $Webcam_{yes}$) are then not available for the selected laptop type.

4.3 Imposing solutions only on request by users!

In addition to giving the user the freedom to circumvent automations it is also our vision to not impose certain decisions, particularly impose fixes for inconsistencies. It is our strong belief that as long as there is more than one solution the decision maker should be made aware of this fact and the decision of how to fix the inconsistency should lie with her. The best case scenario being that all the direct and indirect contributors of the inconsistency are known, so that she can make an informed decision on how to fix the inconsistency. Some automations tend to search for a single solution to an inconsistency, sometimes even with a certain goal, and then impose this solution onto the decision maker. While this may be helpful in certain situations where the decision maker does not care about how the inconsistency was fixed, it should only be the solution if she wishes so.

Furthermore, we also want to allow decision makers to live with inconsistencies and delay the fixing to a later point in time, if there are too many solutions to make a decision right away. This vision also plays into the user not needing to adapt to the tool's needs but rather it to her needs.

4.4 Goals

Given those visions we can formulate the following three research questions that in essence cover them completely:

Table 4.1: Precise goals based on our research questions.

Category	*Label*	*Relates to*	*Question*
Optimality	O1	RQ1, RQ3	Is it possible to calculate the optimal sequence for any given configuration, without knowing it in advance?
	O2	RQ2	How well can an inconsistency be pinpointed?
Correctness	C1	RQ2	Is it possible to always identify the defective decision at the point the inconsistency is detected?
	C2	RQ2, RQ3	Is correct reasoning possible in the presence of inconsistencies?
	C3	RQ2, RQ3	What about multiple inconsistencies?
Scalability	S1	RQ3	Is the calculation of the optimal sequence fast enough to be used in real-time?
	S2	RQ2	How well do the computations for living with inconsistencies scale?
Other Goals	G1	RQ2	Is there an approach that can do that without isolating all decisions?
	G2	RQ2, RQ3	Are there still benefits from reasoning with the remaining decisions?

RQ1: How can we prevent users from getting to dead ends during the configuration process?

RQ2: How can we support users in dealing with dead ends if they cannot be avoided without imposing a certain solution?

RQ3: With keeping RQ1 and RQ2 in mind, how can users still be guided through the configuration process, with respect to minimizing the needed user input?

To answer those questions we have defined several goals and categorized them in Table 4.1. Answering those questions will be part the approach in Chapter 5 and the evaluation in Chapter 7.

Chapter 5

Approach

" Intelligence is the ability to avoid doing work, yet getting the work done. "

– *Linus Torvalds*

In this chapter we present our approach for fulfilling our vision and goals of dealing with inconsistencies in decision-making. In the next Section 5.1 we will outline our reasoning architecture. After that two sections covering the independently solvable problems of how to optimize the sequence of questions to avoid inconsistencies (Section 5.2) and how to live with inconsistencies (Section 5.3) are given.

5.1 Reasoning Architecture Overview

The reasoning architecture is based on SAT-based reasoning and split into several layers as depicted in Figure 5.1. For each layer there exists an interface to provide flexibility in the implementation. The model reasoning layer is initialized with a decision model and a SAT reasoner and uses a model encoder to transform the decision model into CNF. A more detailed description of the inner workings and components is given next.

Starting out with a *Decision Model*, as described in Section 3.1 the first step is to encode the questions with their respective choices and the relations between the choices. This crucial step is performed by the *Model Encoder*. During encoding each decision is assigned a numerical literal, since SAT solver implementations work with numerical literals, e. g., for our illustrative example $Laptop_{yes} \rightarrow 1$, $Laptop_{no} \rightarrow 2$, $Screen\,Size_{irrelevant} \rightarrow 3$, $Screen\,Size_{12.1''} \rightarrow 4$, etc. Note that questions that can become irrelevant because of relevancy relations are automatically recognized and extended with the irrelevant decision if not already present. After that the cardinality constraint of each question is encoded as clauses. For example, exactly one choice must be selected for the laptop question, to ensure this constraint we need two clauses $(Laptop_{yes} \vee Laptop_{no})$ and $(\neg Laptop_{yes} \vee \neg Laptop_{no})$ – the first clause ensures that

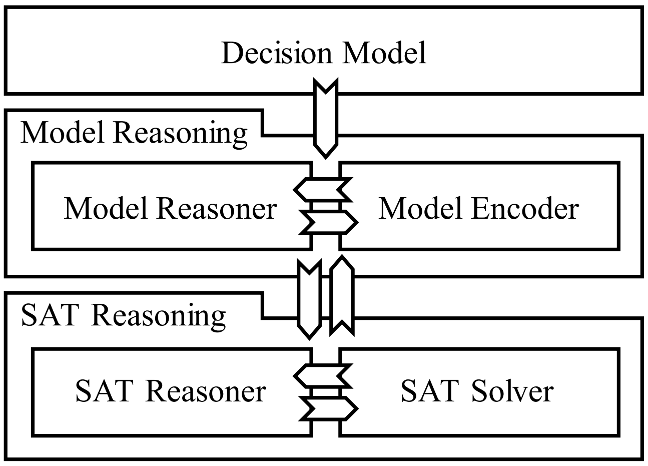


Figure 5.1: Reasoning architecture overview.

at least one choice is selected, in combination with the second clause it is also ensured that at most one choice is selected. Of course for the same constraint more clauses are needed if a question contains more choices. These clauses are later provided to the SAT solver implementation as a stream of integers 1 2 0 and $-1\ \ -2$ 0, the zero signifying the termination of one clause and the negative literals representing the negation. After the encoding of the questions and their choices finally the relations between the questions are encoded. Basically what happens is that relevancy relations are transformed into implications between the decisions that make something relevant or not onto the irrelevant decisions of the target questions, constraint relations are already implications between decisions. Those implications are then transformed into CNF in the manner that any implication can be transformed due to the equivalence $a \Rightarrow b \equiv \neg a \vee b$. The CNF relations are already in CNF and therefore no transformation is needed.

The *Model Reasoner* is the interface between input from a configurator tool and the SAT Reasoning, it gets the information about decisions made and keeps an internal configuration state. With the help of the Model Encoder these decisions are mapped to assumptions which are forwarded to the SAT Reasoning layer for reasoning. Generally speaking it is an extension to the SAT Reasoning layer so it can deal with decision models instead of plain CNF problems.

The *SAT Reasoning* layer contains two vital components, a SAT Reasoner and a SAT Solver. The *SAT Solver* is the "heart and soul" of our reasoning. We use the PicoSAT [61] solver by Armin Biere. However, other solvers can be used too, for instance as discussed later our prototype tool also works with the Sat4j solver [62]. The SAT solver gets as input all the clauses that describe the decision model. During reasoning it will learn new clauses for a faster evaluation, however concerning the configuration it is stateless. This stems from the fact that we use an assumption based configuration, meaning that while the decision model is encoded in CNF decisions made by the user are represented as assumptions. This approach has two key advantages over representing decisions as clauses: *i)* Assumptions can easily be undone due to the nature of most SAT solver implementations. Assumptions need to be assigned before each SAT call and are only considered for one SAT call without triggering the learning of new clauses. In contrast to adding clauses for decisions which would trigger the learning of new clauses. Since removing clauses is expensive because the solver would have to know which clauses were derived from the one that should be removed and this information is not readily available, the most used solution is to reset the solver and initialize it with the original CNF. *ii)* The second advantage of using assumptions over clauses is the simple fact that this way the model is easily distinguishable from decisions made by the decision maker, which is very helpful when dealing with inconsistencies.

Due to the fact that the SAT solver is stateless and produces only SAT or UNSAT as a result, it is only used as an oracle for reasoning purposes by the *SAT Reasoner*. The SAT reasoner provides the missing functionality to perform more complex reasoning tasks and also keeps track of the configuration state and possible inconsistencies. The main reasoning task it performs is to calculate the effects a new decision, given the model and the configuration state, has. Given such a new decision it is encoded and the assumption is added to the configuration state, as a next step the SAT solver is called with the new state. Should the SAT solver return UNSAT we have encountered an inconsistency and need to isolate decisions from reasoning otherwise we cannot use the SAT solver any longer for reasoning because any subsequent call with the same or more assumptions will result in UNSAT. What possibilities one has for such an isolation will be discussed in detail in Section 5.3. For now, let us assume the SAT solver returned SAT, then we can request a complete assignment for all the literals that satisfy the CNF. Based on this assignment we can remove our state, which must be part of the assignment. For each of the remaining literals in the assignment one further call to the SAT solver is made with the assumptions contained in the state and a new additional assumption that is the inverted literal. Depending on the result of this SAT call we can deduce if the new decision had an effect on this literal. Given our example starting with an empty state $\{\}$, the decision maker could decide $Laptop_{yes}$ which would give us the satisfiable state $\{1\}$ and one possible assignment $\{1, -2, -3, 4, \dots\}$. Removing thus the state from the assignment we would be left with $\{-2, -3, 4, \dots\}$. Next we would call the SAT solver with the assumptions $\{1, 2\}$ in order to see if $Laptop_{yes}$ and $Laptop_{no}$ are satisfiable (the information that these assumptions are the laptop decisions is not known in the SAT reasoner and not needed). As we know from the example this is not possible and the solver would return UNSAT, so we can deduce that the effect of $Laptop_{yes}$ is at least $\neg Laptop_{no}$. As mentioned before, other SAT calls would be made with different assumptions: e. g. $\{1, 3\}$, $\{1, -4\}$ and so on determining for each literal, we do not have an assumption yet, if it is still a free variable or already a constrained one. Already constrained ones would be included in the state.

5.2 Guidance Calculation

As part of our vision we proposed to unburden the engineers of thinking about the sequence of questions and suggested to determine an optimal sequence automatically. Optimal with respect to a minimum number of questions that need to be answered so that the rest of the configuration can be deduced automatically due to constraints. In this section we present our solution to this problem. On the one hand we hope that this automation can help reduce the required user input and therefore reduce potential errors. On the other hand keeping in mind that too much guidance can be counterproductive [54] and that the tool should adapt to the user's needs, we developed a robust algorithm that does not depend on the user actually following the proposed sequence. However, there are many pitfalls to consider when thinking about the optimal sequence which are discussed next.

5.2.1 Pitfalls

The main dilemma is that, the optimal sequence of questions differs depending on the configuration. As such, *there cannot be a single sequence of questions that is optimal for every configuration*. This is especially a problem because of the sheer number of configurations; a decision model containing n questions and each question q_i having m_i choices, the following boundaries are given:

(i) The number of different configurations can be greatly reduced by relations, but is at most $\prod_{i=1}^{n} m_i$, however usually even the reduced number of configurations is very high and cannot be computed in reasonable time for mid-size and large decision models.

(ii) Each possible configuration has $n!$ sequence permutations to be generated from.

Since we do not have a priori knowledge of how the questions will be answered, we must optimize the sequence of questions such that any configuration can be derived with a small number of answers. This is only possible through incremental reasoning. As the user starts answering questions, we rearrange the questions on-the-fly such that the sequence of the remaining questions is optimized according to the now reduced set of possible configurations (i. e., every answer tells us something about the user's intention).

We assert that the task of optimizing the sequence of questions requires the understanding of as many configurations as possible (ideally all configurations). However, the number of configurations increases exponentially with the number of questions and their choices (despite the relations) and it becomes unlikely for a human user (creator and / or decision maker) to guess the appropriate sequence of questions. Even for users with domain knowledge, it becomes virtually impossible to manually devise an optimal (or even good enough) sequence of questions. Moreover, simple changes to the model (add / remove questions, choices, or relations) may affect the optimal sequence significantly, making a manual solution to this problem obsolete quickly. Additionally, an optimal sequence might not be an intuitive one, answering certain questions before other questions might not make sense to some users. If a model contains relevancy relations, one must ensure that questions that determine the relevancy of other questions precede those questions. Otherwise the user may decide questions that become irrelevant later and therefore the needed user input has unnecessarily increased.

Another pitfall are questions with choices that have very different impacts on the configuration space, such questions should not be overestimated. An extreme case of such an asymmetric question would be, for instance, a question with one choice that leads to a configuration instantly – meaning decisions can be derived automatically for all the other questions – and other choices that have no impact on other questions whatsoever. The reason that such questions should not be overestimated is, that choices with a big impact on the system limit the number of reachable configurations greatly. On the contrary the other choices of such an asymmetric question leave more configurations within reach. As mentioned before the dilemma is, that we want to provide the shortest sequence to as many configurations as possible. Thus if there is a chance that a decision could be made automatically for such a question it should not be the next question suggested to the user. The previous pitfall is also the basis of the following dilemma: Assuming no domain knowledge is incorporated each choice of a questions by itself has an equal chance of getting selected. But the fact that each choice can lead to a different number of configurations contradicts this statement, assuming each configuration is equally likely.

Finally one has to be aware of the cyclic nature of decision models, if the questions are nodes and the relations the edges cyclic graphs are not that uncommon. Also relations can be modeled in different ways without changing the impact of the relation. Furthermore, although the relation is defined in one direction, reasoning is effected in both directions. This implies that even in graphs containing no cycles the approach has to handle cycles during reasoning.

In summary, the pitfalls are:

(i) No single, optimal sequence of questions exists.

(ii) The optimal sequence changes with every answer.

(iii) The optimal sequence changes with every model change.

(iv) The optimal sequence is not necessarily an intuitive sequence.

(v) The optimal sequence must respect relevancy relations.

(vi) Choices how to answer questions are not equally likely to be selected.

(vii) Questions are connected in a cyclic manner via relations.

5.2.2 Approach

Our approach automatically and quickly determines the near optimal sequence in which questions are presented to the decision maker, specifically to reduce the number of questions a user has to answer. Our premise: a reduction in manual input implies an increase in automation. For the decision maker, however, the optimal sequence of questions may not necessarily be the most intuitive sequence. A user likely prefers to answer those questions first that are easy to answer or most significant from their perspective. However, these criteria differ for every user and as such there is no single, ideal sequence (neither intuitive nor optimal). Even though our approach fully automatically optimizes the sequence of questions, it does not require the user to answer the questions in the sequence presented. The user may answer questions arbitrarily in which case our approach fully automatically optimizes the sequence of the remaining questions. It is also possible for a creator to provide a "rough guide" for the decision maker, if so desired (i. e., partial sequence of questions). Indeed, our approach effectively only optimizes the sequence for the next question because the sequence of the remaining question almost always changes with additional answers (i. e., the more answers provided, the more obvious becomes the user's intention which is then considered as part of our reasoning process). A satisfactory next question should be one that fulfills the following criteria:

(i) It has the least potential to be answered automatically.

(ii) It has the most potential of answering many other questions.

On a finer level of granularity, these criteria can be applied to choices of questions also. If a question cannot be answered automatically but its choices can be reduced automatically then the probability of reduction is a benefit as well. Questions or choices that are independent of others can be answered in a random sequence. This is also true for "clusters of questions", where questions within clusters are dependent on one another but independent of questions in other clusters.

5.2.3 Computing the Ideal Solution

If it were possible to compute all possible configurations of a system in advance, we could generate an a priori optimal sequence of questions for the decision maker. Although this solution is practically infeasible due to the exponential number of possible configurations. We will discuss it next to better illustrate the workings of our solution. The optimal, yet unscalable approach is fairly easily understandable. Our near-optimal, yet scalable solution in essence approximates its reasoning.

For the sake of simplicity we just consider two questions out of the original seven defined in our illustration, *LaptopType* and *Screen Resolution*. The matrix in Figure 5.2 (left side) depicts all possible configurations with these two questions. The exponential growth is evident in the fact that each question adds another dimension, e. g. adding a third question would result in a cube and so on. Relations can be seen as an a priori elimination of configurations: a priori because relations implicitly eliminate configurations of the pool of all possible configurations. Those eliminated configurations are represented by the grayed out cells (dark gray) in Figure 5.2. It is important to note

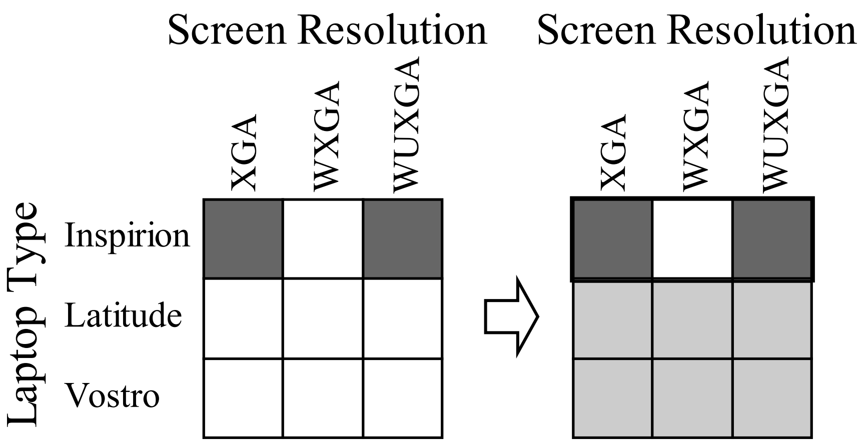


Figure 5.2: Valid configurations considering relations.

that relations are not directional. We previously defined the *Screen Resolution* question to restrict the *Laptop Type* question; however, in return, the *Laptop Type* question also restricts the *Screen Resolution* question. Therefore, relations do not impose a sequence on how to answer questions.

For our approach, the expected elimination is the most important criterion for deciding how to order questions. Our goal is to order questions such that they maximize the expected elimination. In other words, the faster we eliminate configurations the user does not care about (through selection), the faster we reach the one configuration the user is aiming for. Without a priori knowledge about likelihoods of configurations, we must assume that all configurations are equally likely. Counting the number of valid configurations for each row and column in Figure 5.2, we see that some choices of questions lead to a single valid configuration (e. g., $Laptop\,Type_{Inspirion}$ as indicated on the right side of Figure 5.2, in addition to the relation the user selection also eliminates configurations as indicated with the light grayed out cells), some choices lead to two valid configurations (e. g., $Screen\ Resolution_{XGA}$), and some choices lead to three valid configurations (e. g., $Laptop\ Type_{Latitude}$). The elimination of configurations through relations thus changes the impact of questions. Without the relation, the *Laptop Type* and *Screen Resolution* questions would be equal. The relation, however, tips the scale in favor of the *Screen Resolution* question, as is discussed next:

Elimination Likelihood of *Laptop Type* Choices

The expectation value defines the expected outcome for discrete, random events. The following states the standard expectation value as the sum of a random variable x_i multiplied by its probability of occurring $p(x_i)$: $E(x) = \sum_{i=1}^{n} p(x_i) * x_i$

Applied to our problem of eliminating configurations, x_i is the number of eliminated configurations by selecting a choice for a given question and $p(x_i)$ is the likelihood that this elimination occurs. Here we have two possibilities: *i)* each choice has an equal chance or *ii)* each configuration has an equal chance. As was discussed above, in the absence of any feedback on the likelihood of configurations occurring, all configurations become equally likely and $p(x_i)$ is then the ratio of remaining configurations after selection divided by all available configurations:

$$E(q_i) = \sum_{i=1}^{\#choices} \frac{\text{remaining configurations}}{\text{all configurations}} * \text{eliminated configurations}$$

The selection of $LaptopType_{Inspirion}$, as shown in Figure 5.2, would eliminate 6 configurations with a probability of $1/7$ (1 configuration out of 7 is still possible). The selection of $LaptopType_{Latitude}$ eliminates 4 configurations with a probability of $3/7$. The selection of $LaptopType_{Vostro}$ also eliminates 4 configurations with the probability of $3/7$. Adding up these benefits and likelihoods gives us: $6 * 1/7 + 4 * 3/7 + 4 * 3/7 \approx 4.29$

Elimination Likelihood of *Screen Resolution* Choices

The selection of both $ScreenResolution_{XGA}$ and $ScreenResolution_{WUXGA}$ eliminate 5 configurations with a likelihood of $2/7$. $ScreenResolution_{WXGA}$ eliminates 4 configurations each with a likelihood $3/7$. The expectation value is thus: $5*2/7+4*3/7+5*2/7 \approx 4.57$

The sum of eliminations and their likelihoods reveal that the choices for *LaptopType* are likely to eliminate less configurations than the choices for *ScreenResolution*. As we discussed before, the problem of deciding the optimal sequence of question is simply about deciding on the highest potential for eliminating configurations. The *Screen Resolution* question is thus more significant and should be asked / answered first, the *Laptop Type* question next.

It is important to note that for configuration problems, the expectation value favors symmetric questions where the choices' probabilities are more evenly distributed. The choices of the *ScreenResolution* question had the likelihood $2/7$, $3/7$, and $2/7$. The choices of the *LaptopType* question had the likelihood $1/7$, $3/7$, and $3/7$. The probabilities of the *ScreenResolution* question where thus more symmetric, although the difference in this case is not too big, consequently the expectation value favored the *ScreenResolution* question.

5.2.4 Approximating the Ideal Solution

Our approach consists of three essential steps to optimize the sequence of questions. Since an answer affects the sequence of the remaining questions, our approach repeats these three steps after every answer given. These steps are:

(i) Simulating the effects of selecting choices. The effect of selecting a choice is either the elimination of other choices or the elimination of entire questions, or none at all. We quantify the effect as the gain of each selection.

(ii) Combining the simulation effects (gains) to consider the impact of questions as a whole rather than their choices.

(iii) Presenting our findings to guide the decision maker.

Our approach uses simulation to assess a choice's expectation value. It can be best described as exploring a series of "*What happens if I answer the question by selecting this choice?*" questions. This problem is simplified by the fact that our approach only ever investigates the next ideal question to answer. This can be done independently for every question which is an important scalability property (i. e., we know that the sequence changes after an answer and as such there is no benefit in investigating such "what happens if" questions recursively).

The simulation considers the effect of a choice's selection on itself and other questions. This benefit is quantified in our approach as a *gain*. How the gain is computed depends on the relation. Recall that we distinguish between *constraint relations* and *relevancy relations*. Next, we discuss the computation of gains for constraint relations only. We discuss the computation of gains for relevancy relations later.

Constraint Relations Gain

For constraint relations, the gain is based on the percentage of choices eliminated per question: 100 is equivalent to answering a question, 50 is equivalent to eliminating half the choices of a question (i. e., answering 50% of a question). To simulate the gain for selecting a choice, the following steps have to be taken:

(i) The gain starts at 100 always because selecting a choice answers its question at the very least.

(ii) Next we investigate the relations the question is involved with and simulate the effects of the selection. For all questions affected by the selection, we then add the degree of reduction (e. g., 100 for a complete question answered, or 66 for 2/3 of the choices eliminated).

It is important to note that we value a question answered ($gain + 100$) analogous to two questions' choices reduced by 50% ($gain + 50 + 50$). Also note that this simulation considers the effect of a single choice's selection but not the combined effect of different choices as an optimal solution would do – the latter is needed for optimality; however, we will demonstrate later that we are near optimal and thus the combined effect is less important.

To illustrate this approach, let us take another look at our illustration in Figure 3.1. Initially, a simulation for each choice has to be performed. The sequence in which these simulations are performed is irrelevant. For instance, we could start with the choice *XGA* of the *Screen Resolution* question.

```
SelectionGain(Screen Resolution, XGA) := 100 +
  EliminationGain(Laptop Type, {Inspirion})
EliminationGain(Laptop Type, {Inspirion}) := 1/3 * 100

//result
SelectionGain(Screen Resolution, XGA) :≈ 133
```

Selecting XGA has a gain of 100 (because it answers the *Screen Resolution* question). Furthermore, the selection of XGA affects the *Laptop Type* question. We know that $Screen\,Resolution_{XGA}$ is not available for $Inspirion$ laptops, thus it can be eliminated when XGA is selected ($1/3^{rd}$ of the *Laptop Type* choices are eliminated: $1/3*100$). The gain for the selection is therefore: $Screen\,Resolution_{XGA} = 100 + 1/3 * 100 \approx 133$

The above example showed that the selection gain is based on the elimination gain (for eliminating choices in other questions). Elimination, in turn, may trigger a subsequent selection if all but one choice of a question is eliminated. We encounter such a situation while simulating the selection of the $Laptop\,Type_{Inspirion}$:

```
SelectionGain(Laptop Type, Inspirion) := 100 +
  EliminationGain(Screen Size, {13.3''}) +
  EliminationGain(Screen Resolution, {XGA, WUXGA}) +
  EliminationGain(Webcam, {no})

EliminationGain(Screen Size, {13.3''}) := 1/3 * 100

EliminationGain(Screen Resolution, {XGA, WUXGA}) :=
  SelectionGain(Screen Resolution, WXGA)

SelectionGain(Screen Resolution, WXGA) := 100

EliminationGain(Webcam, {no}) :=
  SelectionGain(Webcam, yes)

SelectionGain(Webcam, yes) := 100

//result
SelectionGain(Laptop Type, Inspirion) :≈ 333
```

The simulation of $Laptop\,Type_{Inspirion}$ is similar to the simulation of $Screen\,Resolution_{XGA}$ except that the $Laptop\,Type_{Inspirion}$ eliminates all but one choice from both the *Screen Resolution* and $Webcam$ questions. Reducing a set of choices for a question to one choice is synonymous to answering that question (i. e., we can automatically select the only remaining choice). The gain for $Laptop\,Type_{Inspirion}$ thus includes the gains for $Screen\,Resolution_{WXGA}$ and $Webcam_{yes}$. Normally, the gain calculation for $Webcam_{yes}$would include the elimination of $Laptop\,Type_{Latitude}$. However, since the selection of $Inspirion$ already eliminated this choice no additional gain is added (i. e., we do not double count gains).

Thus far, we discussed gains for constraint relations in detail. Before we can discuss gains for relevancy relations, we must first discuss how to compose the gains of choices to the gains of their questions.

Gain Composition

Until now, we computed the gain for single choices only. To assess the gain for the overall question, we have to compose the gains of its choices. The most obvious method of composition would be to compute the average gain of the choices (mean value); however, doing so would not give satisfactory results (although this method would be sufficient for our small illustration). The reason for that is the symmetry issue discussed in Section 5.2.3. Expectation values favor symmetric likelihoods and since our gain represents likelihood, we must favor symmetric gains (e. g., a question with two choice $gains = \{200, 200\}$ would be preferred over a question with choice $gains = \{300, 100\}$). The choices of asymmetric questions divide the possible configurations unevenly. If one choice has a very high gain, the selection of this choice is not as likely as those of others since it would lead to a smaller subset of configurations. Statistically, a question with choices having symmetric gains should thus be favored. Mathematically, this is simply achieved through computing the product of the gains of the choices:

$$Gain(Question) = \prod_{choice \in Choices_{Question}} Gain(Choice)$$

Due to the fact that all choice gain values are greater or equal to 100, the product of choice values is always greater than each individual choice value. To illustrate this, the following computes the gain for the *Screen Resolution* question:

```
Gain(Screen Resolution) := SelectionGain(Screen Resolution, XGA) *
  SelectionGain(Screen Resolution, WXGA) *
  SelectionGain(Screen Resolution, WUXGA)

Gain(Screen Resolution):= 133 * 100 * 133 = 1768900
```

The expectation value (Section 5.2.3) also favors questions with more choices over questions with fewer choices. By multiplying the choice gains to a question gain, we in essence also favor questions with more choices; however, a question with fewer choices can overtake a question with more choices if its choice gains are significantly higher or more symmetrical.

Relevancy Relations Gains

Until now, we ignored the gain computation for relevancy questions. We now address this issue. Relevancy relations identify questions that only need / should be answered in certain situations. For example, the *Memory*, *Screen Size*, *Screen Resolution*, *Webcam*, *Operating System*, and *Laptop Type* questions should only be answered if the user chooses to configure a laptop ($Laptop_{yes}$). Relevancy relations thus impose a sequence among questions and we need to ensure that questions that triggers the relevancy are answered before the potentially irrelevant questions (e. g., the *Laptop* question is asked before the other questions in our illustration). To ensure that the potentially irrelevant questions are never offered before the questions that trigger the relevancy, we add the gain of the potentially irrelevant question to that of the triggering one. The simulation of the choices for the *Laptop* question is depicted next:

```
SelectionGain(Laptop, yes) := 100

SelectionGain(Laptop, no):= 100 + Gain(Memory) +
  Gain(Screen Size) + Gain(Screen Resolution) + Gain(Webcam) +
  Gain(Operating System) + Gain(Laptop Type)
```

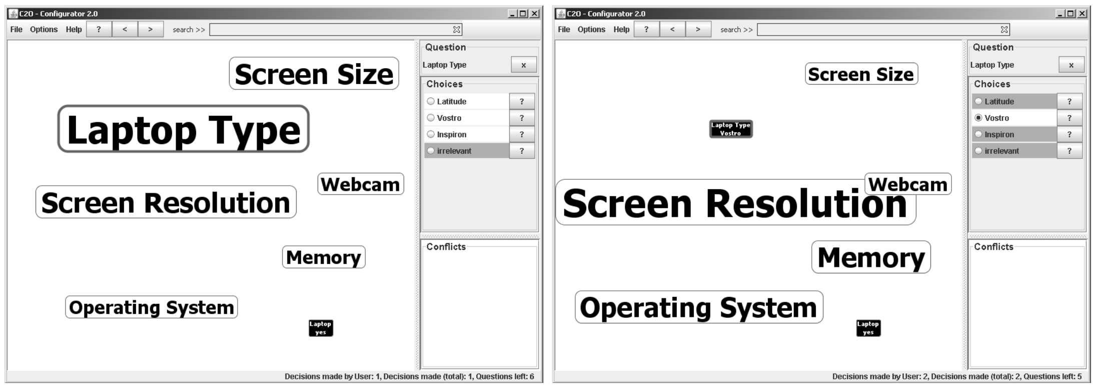


Figure 5.3: C2O Configurator screenshots – the bigger the font the higher the calculated gain for the question. Answering a question changes the importance of the other questions.

The computation reflects the fact that answering *Laptop* with the answer no also answers six other questions (i. e., not having to ask those questions is equivalent to having answered them). The gain computation for relevancy relations appears similar to that of constraint relations. However, note that here we add the gains for whole questions rather than the gains for individual choices. This ensures that the question that triggers the relevancy has a bigger gain than the potentially irrelevant questions.

Impact of Answering Questions

After choice gains are composed into question gains, the questions are ordered from the most promising (highest gain) to the least promising one (lowest gain). In this case, "promising" means in the question that is most likely to lead to the desired configuration optimally if answered next. The initial sequence for our example would be *Laptop*, *Laptop Type*, *Screen Size*, *Screen Resolution*, *Memory*, *Operating System*, and *Webcam* according to their calculated gain values:

```
Gain(Laptop) := 100 *
  (100 + 8857800 + 1768900 + 1768900 + 20000 + 20000 + 17689) =
     1245338900
Gain(Laptop Type) := 333 * 200 * 133 = 8857800
Gain(Screen Size) := 133 * 133 * 100 = 1768900
Gain(Screen Resolution) := 133 * 100 * 133 = 1768900
Gain(Memory) := 100 * 200 = 20000
Gain(Operating System) := 200 * 100 = 20000
Gain(Webcam) := 133 * 133 = 17689
```

The *Laptop* question is the most important question due to the relevancy relation. The next important question is the *Laptop Type* question and the *Webcam* question is the least significant one of all. We discussed earlier that the sequence of questions may change with answers. Ignoring the *Laptop* question, the *Laptop Type* question is the most significant one. One way to visualize this, is to set the font size of a question according to its significance, as can be seen in a screenshots taken from our C2O Configurator tool that implements our approach, depicted in Figure 5.3 (left-hand side). If we answer this question, the gains for the remaining questions change. For example, answering $Laptop\,Type_{Vostro}$ eliminates $Screen\,Size_{12.1''}$ as a result the gains for the remaining questions become change (Figure 5.3 right-hand side). More details on our configurator tool will be presented in detail in Chapter 6.

5.2.5 Choice Gain Algorithm

The following summarizes the algorithm for computing the selection gain and elimination gain in pseudo code. The computation of the relevancy gains is omitted for brevity and simplicity. We see that `SelectionGain` takes as the parameter the question and choice selected. `SelectionGain` then explores all relations, identifies which choices they eliminate, and then assesses the gains of these eliminations by calling `EliminationGain`. `EliminationGain` returns a gain, if it was not considered previously. It is similar to `SelectionGain`, except that it handles the special case where the size of the remaining choices (after elimination) becomes one, in which case the remaining choice is selected automatically (thus triggering a `SelectionGain` call).

```
function SelectionGain(question, selection)
  SelectionGain := 100
  for each relation in GetRelations(question)
    relatedQuestions := GetRelatedQuestions(relation)
    choices := GetEliminated(relatedQuestions, selection)
    gain := EliminationGain(relatedQuestions, choices)
    SelectionGain := SelectionGain + gain
  end
end

function EliminationGain(question, exclude)
  if IsAnswered(question) then EliminationGain:=0, exit
  remaining := GetValidChoices(question) \ exclude
  if Size(remaining) == 1 then
    EliminationGain := SelectionGain(question, remaining)
  else
    all := Size(GetValidchoices(question))
    EliminationGain := (Size(exclude) / all) * 100
    for each relation in GetRelations(question)
      relatedQuestions := GetRelatedQuestions(relation)
      choices := GetEliminated(relatedQuestions, remaining)
      gain := EliminationGain(relatedQuestions, choices)
      EliminationGain := EliminationGain + gain
    end
  end
end
```

To avoid gains getting too big we use a little trick for the gain composition. Our `ComposeGain` function uses a logarithmic addition instead of the multiplication, introducing some small but negligible rounding errors but avoiding overflows with standard integer types.

```
function ComposeGain(gains)
  ComposeGain := 0
  for each gain in gains
    ComposeGain := ComposeGain + Log(gain)
  end
end
```

5.3 Living with Inconsistencies

As part of our vision we proposed not imposing solutions onto decision makers, like for example the sequence questions have to be answered in or the time when a fix has to be performed in case of an inconsistency. In this section we present our solution to living with inconsistencies. We hope that this enables decision makers to work more naturally and adapt the configurator to the decision maker's needs.

5.3.1 SAT-based Reasoning in the Presence of Inconsistencies

As mentioned in Section 2.1.2, as long as a CNF with assumptions evaluates to SAT, no inconsistency is detected. Adding additional clauses and / or assumptions may change SAT to UNSAT, but once the SAT solver is in an UNSAT (inconsistent) state, adding additional clauses and / or assumptions will have no effect at all because the SAT solver will continue to evaluate to UNSAT. As explained in Section 2.2.3, since the SAT solver is used for more then just detecting inconsistencies, automations are lost too. For example, the ability to automatically derive assumptions *i)* to (partially) auto-complete the configuration process or *ii)* show decision effects, is lost.

If the tolerance to inconsistencies should not change the SAT-based automation then the only option is isolation. It is important to distinguish between isolating and fixing an inconsistency at this point: Isolating means sandboxing clauses and / or assumptions that cause an inconsistency, in other words identify contributors and ignore them for reasoning purposes. Fixing would go one step further and in addition change those clauses and / or assumptions in such a way the inconsistency would be resolved. So the isolation can be seen as a first step of an actual fix without committing on how to fix.

In the domain of decision-making, isolating clauses and assumptions account for different parts. Assumptions are used to express user decisions and derived decisions (high-level), whereas clauses are used to define the decision model (low-level). As mentioned earlier we work under the assumption that the decision model is correct, and that the user decisions are contradictory to the model (much like we presume that in case of inconsistencies in design models, the user model is at fault and not the meta model that defines the modeling language). Therefore we only care about isolating user assumptions, ignoring the clauses. So looking at the example from Table 3.2 with the inconsistency detected at the fifth answer, given the constraints and considering all possibilities, the following decisions contribute to the inconsistency: $Screen\,Size_{12.1''}$, $Screen\,Resolution_{XGA}$ and $Webcam_{yes}$. As a matter of fact isolating any one of these is sufficient to get meaningful results again. At this point the user could continue configuring the product and fix the problem later on. Once the user decides to fix the problem, the defective contributor has to be identified by the user, since choosing a random contributor often does not suffice. The user then has to provide a different, valid decision for the identified question, resulting in the fix of the problem.

5.3.2 SAT Concepts that deal with Managing Inconsistencies

Before we go into detail explaining different isolation strategies, we first need to introduce additional (high-level) SAT concepts [23, 63]. First an important SAT concept is a minimal unsatisfiable set (MUS) which is defined by the properties of being minimal and that removing any single assumption results in the remaining set being satisfiable. In our example configuration illustrated in Table 3.2 only one MUS of user assumptions is present $\{Screen\,Size_{12.1''}, Screen\,Resolution_{XGA}, Webcam_{yes}\}$. However, usually inconsistencies consist of many possibly overlapping MUSes, as a consequence isolating one assumption of one MUS does not necessarily result in a satisfiable SAT model. Another concept is the minimal correcting set (MCS) which is defined by the properties of being minimal and that removing it from reasoning, results in a satisfiable SAT model. In our illustration the MCSes are $\{Screen\,Size_{12.1''}\}$, $\{Screen\,Resolution_{XGA}\}$, and $\{Webcam_{yes}\}$. Generally speaking MUSes and MCSes are connected via hitting sets, meaning that every MCS is composed of a single element from every MUS. In addition to this relation MCSes are the complement of a maximum satisfiable set (MSS). As the name already states a MSS is a set of assumptions that is satisfiable and of the maximum size it can be. An example for one possible MCS and its complementary MSS given our illustration, is the MCS $\{Screen\,Size_{12.1''}\}$ and the MSS $\{Memory_{8GB}, Screen\,Resolution_{XGA}, Webcam_{yes}\}$.

5.3.3 Different Isolation Strategies

In the following the four different isolation strategies we investigated will be explained in detail, with the help of Table 5.1. This table continues the configuration example from Table 3.2 where the derived state after the fourth questions was complete, but introduces a conflicting user assumption at the fifth question. We will solely focus on isolating user assumptions and recalculating derived assumptions (high-level facts during SAT-based reasoning) and assume that the decision models themselves are correct and therefore the clauses (low-level facts during SAT-based reasoning) do not need to be changed. However, it should be pointed out that two out of the four isolation strategies (MaxSAT and HUMUS) could also be applied to isolate low-level facts [63, 64].

5.3.3.1 Disregard All Strategy

A trivial way to ensure a correct state in the presence of inconsistencies is to ignore everything that happened so far, meaning that every single assumption before the inconsistency was encountered is isolated from future reasoning. This solution may seem mundane; however, we can think of it as the worst-case strategy against which others can be compared. This strategy is also specific to high-level isolation since isolating all clauses hardly makes sense. The result of this isolation strategy can be seen in Table 5.1: All user assumptions are isolated resulting in no derived assumptions for future questions, but also for already decided questions – basically the initial state is restored.

Table 5.1: Isolation strategies based on the example in Table 3.2.

Decision	$4^{th}q$	$5^{th}q$				
	s	u	*Disregard all* s	*Skip* s	*MaxSAT* s	*HUMUS* s
$Laptop_{yes}$	**1**			**1**	**1**	**1**
$Laptop_{no}$	0			0	0	0
$Screen\,Size_{12.1''}$	**1**			**1**	**1**	
$Screen\,Size_{13.3''}$	0			0	0	
$Screen\,Size_{15.4''}$	0			0	0	
$Memory_{2GB}$	0			0	0	0
$Memory_{8GB}$	**1**			**1**	**1**	**1**
$Screen\,Resolution_{XGA}$	**1**			**1**	0	
$Screen\,Resolution_{WXGA}$	0			0	1	
$Screen\,Resolution_{WUXGA}$	0			0	0	
$Webcam_{yes}$	0	**1?**		0	**1**	
$Webcam_{no}$	1			1	0	
$Operating\,System_{32bit}$	0			0	0	0
$Operating\,System_{64bit}$	1			1	1	1
$Laptop\,Type_{Inspirion}$	0			0	1	
$Laptop\,Type_{Latitude}$	1			1	0	
$Laptop\,Type_{Vostro}$	0			0	0	

q...question, u...user assumption, s...derived state, 0...false, 1...true

5.3.3.2 Skip Strategy

In order to be consistent again, this strategy skips the user assumption immediately preceding the detection of the inconsistency. Unless clauses are added iteratively this also makes no sense for a low-level isolation strategy (hence it applies to high-level facts only). The result of this isolation strategy can be seen in Table 5.1: The conflicting user assumption $Webcam_{yes}$ is isolated (skipped) – basically the state before the inconsistency detection is kept, requiring a simple history of user assumptions to implement it.

5.3.3.3 MaxSAT Strategy

MaxSAT stands for maximum satisfiability of the highest cardinality [64]. The basic concept is to identify a set of clauses that can be satisfied with a maximum cardinality. In our case, since we do not care about clauses but assumptions, this idea can be translated to keeping as many user assumptions as possible that do not contradict each other, or in other words find the "closest" solution. After a MaxSAT solution is calculated, every assumption not contained in the solution is isolated. While Disregard All and Skip are isolation strategies with a single solution, MaxSAT is different in that there could be multiple, alternative "closest" solutions with an equal number of user assumptions kept. For the given illustration there are three possible solutions of the same cardinality. Since MaxSAT is non-deterministic any of those alternatives can be the result. The one solution presented in Table 5.1 isolates the user assumption $Screen\,Resolution_{XGA}$, the most noticeable effects of this isolation are that the derived positive assumption $Screen\,Resolution_{WXGA}$ and that $Laptop\,Type_{Inspirion}$ is derived instead of $Laptop\,Type_{Latitude}$. The isolation solution from Skip is a special case of a MaxSAT isolation; however, it may not always be desired to isolate the last decision made by the user.

The implementation of the MaxSAT strategy is realized by searching for a MSS with maximum cardinality, by iterating over user assumptions subsets starting with the biggest down to the smallest ones. As soon as a subset is found that is satisfiable, every assumption that is not contained in this set is isolated. MaxSAT typically returns the first MSS it finds, ignoring potential other MSS of the same size.

5.3.3.4 HUMUS Strategy

HUMUS stands for High-level Union of Minimal Unsatisfiable Sets; it is a concept based on the calculation of all Minimal Unsatisfiable Sets (MUSes) [63], which again targets more at the low-level. The basic concept behind it is to isolate all contributors (directly and indirectly) of the inconsistency and only keep assumptions that have no relation to the inconsistency. The result of this isolation strategy is depicted in Table 5.1, by isolating all contributors the only impartial assumption $Memory_{8GB}$ is kept. Note that the HUMUS calculation only returns a single result like Skip or Disregard All because an user assumption either contributes to the inconsistency or it does not; if it contributes it is per definition an element of HUMUS.

The implementation of the HUMUS strategy takes a shortcut in comparison to the approach of Liffiton to compute all MUSes [63], since we only care about the union of the MUSes and not the individual MUSes themselves. Our implementation uses a variant of Liffiton's [63] approach to calculate MSSes using assumptions over clause selector variables. If a satisfiable subset of clauses has been found, which cannot be increased in size without making the resulting formula unsatisfiable, the complement of this set is an MCS. This particular satisfiable subset is then blocked with a blocking clause. We do not use at-most constraints, this avoids having to reset the SAT solver as soon as an MCS of a different size is found. After having calculated all the MCSes we simply calculate the union of the MCSes, since the resulting set is the same as the union of all MUSes due to the relation via hitting sets.

5.3.4 Discussion of Isolation Strategies

As can be observed in Table 5.1 the different isolation strategies, described in the last section, result in different states after the inconsistency is encountered. Thus they do not achieve the same result but rather provide alternatives on how to proceed after an inconsistency is encountered. All strategies have in common that the SAT-based automations appear functional again (SAT is returned instead of UNSAT). But how can one tell which resulting state is the most complete or correct one, without knowing how the user eventually will fix the inconsistency? Obviously there must be qualitative differences once the user fix is known and the reasoning so far can be analyzed with respect to the now known fix. These qualitative differences can be divided into three categories and are discussed next.

5.3.4.1 Incomplete Reasoning

If the isolation strategy removes correct user assumptions then the reasoning gets incomplete due to missing information in the reasoning process (not as much is inferred as could be). As a result user guidance (see Section 2.2.3) would potentially offer fewer derived assumptions. For decision-making scenarios, fewer derived assumptions are not problematic except that the degree of automation decreases (hence, more isolation implies less automation). However, do note that completeness during user guidance is a loose concept. For example, at the beginning of the configuration process no user guidance is available, because no user assumptions are available to reason with. During the configuration process, depending on how many relations the given answers are in, the number of derived assumptions increases steadily. And at the end of the configuration process, when all is known, naturally guidance would be best since the reasoning is complete, but that is also the point where guidance is not needed anymore.

Generally speaking, as can already be observed in Section 5.3.3, the Disregard All isolation strategy results in the most incomplete reasoning possible (worst case), while the MaxSAT and Skip strategies potentially suffer the least incomplete reasoning (best case). For the HUMUS strategy the degree of incomplete reasoning could vary between the best and the worst case depending on the number of constraints in the model and therefore the number of involved assumptions. However, many constraints in a model could indicate overlapping constraints and redundant information. For example, revisiting the illustration in Figure 3.1, both $Screen\,Size_{13.3''}$ and $Screen\,Resolution_{WUXGA}$ result in the elimination of $Laptop\,Type_{Inspirion}$. So isolating only one of those two assumptions would not result in incomplete reasoning (hence, isolation results in a potential incompleteness only). Even if both assumptions would be isolated and cause incompleteness due to something correct being isolated, new answers like $Webcam_{no}$ would re-provide this lost piece of information again (hence, isolation may lead to temporary incompleteness only).

5.3.4.2 Incorrect Reasoning

Incorrect reasoning is the result of reasoning with defects. Knowing how to fix an inconsistency is crucial for determining incorrect reasoning because it identifies defects that caused inconsistencies. Incorrect reasoning can be determined by analyzing the effects that defects have on the reasoning process, if not isolated. Reasoning with defects would mean that the guidance might leave out correct choices or even suggest incorrect choices to follow-up questions. Since constraints are somewhat redundant as was discussed above, a non-isolated defect might also lead to another inconsistency later. That is, it may conflict with new user assumptions while tolerating inconsistencies which seems more equivalent to postponing inconsistencies rather than tolerating inconsistencies. Related to this problem is the detection of another inconsistency while already tolerating an inconsistency. In this case it cannot be determined if it is an inconsistency related to the inconsistency that was tolerated earlier or if it is in fact a new, unrelated inconsistency.

When using the Disregard All and HUMUS isolation strategies one can be sure to eliminate the defect, because Disregard All isolates all assumptions made by the user prior to the inconsistency (and is as such conservative) and HUMUS computes all assumptions involved in the inconsistency (directly and indirectly). HUMUS, in the worst case, could isolate everything like Disregard All if all assumptions are contributors to the inconsistency. On the other hand with the MaxSAT and Skip strategies it is the inconsistency that is eliminated and not necessarily the assumption(s) that the user will change later when fixing (though by random chance these strategies may also isolate these assumptions). It follows that one cannot be sure if the configuration process is being continued with incorrect derived assumptions based on something that will be fixed and hence the user should not fully trust the results derived from these kinds of automations. In other words MaxSAT and Skip are maximizing what assumptions to keep (they isolate less) which likely leads to less incomplete reasoning, though at the expense of incorrect reasoning. Disregard All and HUMUS likely lead to more incomplete reasoning, though they are guaranteed not to lead to incorrect reasoning – in the presence of inconsistencies.

So in regard to goal C1 from Table 4.1, it is possible to always identify the defective decision. However, it depends on the isolation strategy when it is known for sure that the defective decision was identified. And in reference to goal C2 from Table 4.1, it is possible to reason correctly in the presence of inconsistencies when the Disregard All or the HUMUS isolation strategy is used.

5.3.4.3 Revisitation

Depending on the use of the SAT solver this could be more or less important. In decision-making scenarios this means that the more user assumptions are isolated the more questions have to be potentially revisited at a later point in time (less automation for the end user). As with incomplete reasoning this issue is the biggest for the Disregard All approach, the smallest for the MaxSAT and Skip approaches, and highly depends on the model and situation for the HUMUS approach.

Chapter 6

Proof of Concept

“ Design is a funny word. Some people think design means how it looks. But of course, if you dig deeper, it’s really how it works. ”
– *Steve Jobs*

One example of applying our “behind the curtains” approaches is our proof of concept tool. In this chapter we will introduce the C2O – Configurator 2.0 – tool, one possibility of how to visualize our guidance calculations and integrate them with a tool. First we will give a short summary of the specific goals of the tool in Section 6.1. Next we provide an overview over the tool architecture in Section 6.2, after that we will explain visualization aspects in Section 6.3.

6.1 Goals

The main goal for the C2O configurator tool was to develop one possibility of how to present our reasoning approaches. We wanted to provide a clean and intuitive user interface that diversifies on existing configurators, not only by providing new guidance aspects to the decision maker, but also by changing the typical look of configurators.

It should not restrict the decision maker in any way on what questions she wants to answer first and of course also should not hinder her when she wants to make conflicting decisions. On the other hand we wanted to provide as much guidance as possible without being too intrusive. Thus realizing our main overall goals of preventing and managing inconsistencies during configuration.

6.2 Tool Architecture

The C2O configurator tool is built on the reasoning architecture described in Section 5.1 and communicates with it via events as depicted in Figure 6.1. Internally it keeps a

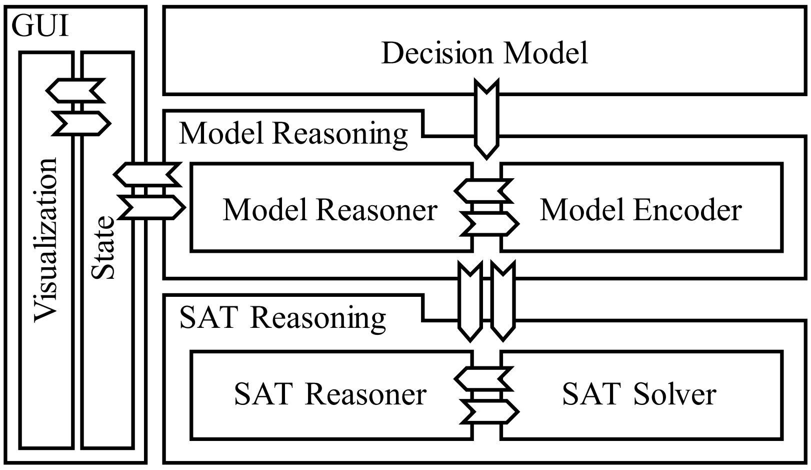


Figure 6.1: Overview of the C2O Configurator architecture.

state of user decisions that is separated from that of the reasoning engine. This is necessary because in the case of inconsistencies user decisions get isolated from reasoning. However, in the user interface we want to retain those decisions and visualize this aspect to the decision maker. Furthermore, because our preferred isolation strategy is HUMUS, we know that after an inconsistency is resolved, decisions that were isolated might be viable again and should not have to be remade by the decision maker.

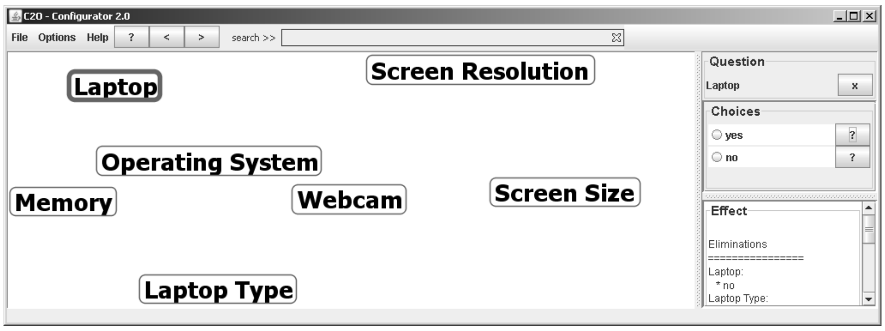
Currently, only the interaction between the decision maker and the system has a graphical user interface, defining questions, choices and relations is done programmatically.

6.3 Visualization Aspects

The visualization aspects can be categorized into three type of aspects, *i)* the general visualization of decision models, *ii)* the visualization of the guidance calculation and *iii)* the management of inconsistencies as described in the next sections.

6.3.1 Decision Models

Decision models are visualized in the C2O configurator tool as shown in Figure 6.2 for our illustration. In comparison to state-of-the-art configurator tools we turned away from the usual hierarchical way of visualizing the questions and embrace the chaos. The choices of one question are presented on the left as soon as the decision maker clicks on a question. When it comes to relations the visualization is rather crude, constraint relations are not visualized at all. However, by clicking on the "?" besides one choice one can find out what the effect of selecting the choice would be, as shown for $Laptop_{yes}$ in Figure 6.2. While some configurators have the ability to visualize constraints (e. g. the S2T2 [35]), usually constraints such as cross-tree constraints are



```
Eliminations
===============
Laptop:
  * no
Laptop Type:
```

Figure 6.2: Visualization of the illustrative example in the C2O Configurator.

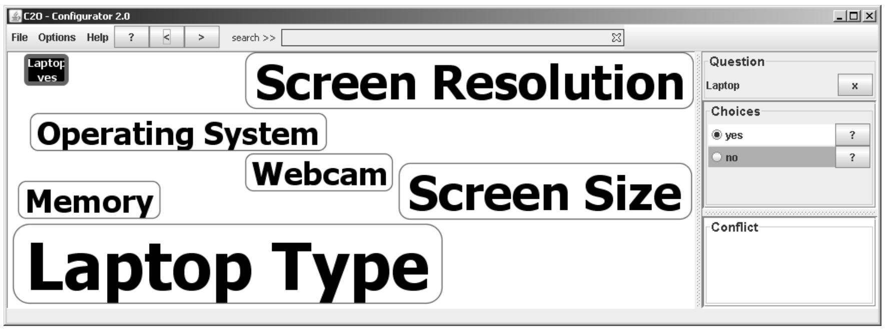


Figure 6.3: Visualization of the guidance calculation.

not visualized in configurators. Although constraint relations are not visualized by our tool, there is the ability to visualize the relevancy relations by hiding irrelevant questions. When this feature is activated all questions but the *Laptop* question are hidden from the decision maker at first, since the other questions only become relevant after the decision $Laptop_{yes}$ was made. As soon as they become relevant they will also be shown again to the decision maker.

6.3.2 Guidance

The screenshot shown in Figure 6.2 is without our guidance enabled. Once it is enabled and the potential to be the next question in order to minimize input is calculated for each question, it is visualized by making the font bigger, the higher the potential to lead to the shortest path for any configuration, the bigger the font. As a result, as shown in Figure 6.3, after the decision $Laptop_{yes}$ is made by the decision maker, visualized by making it black and show the selection *yes* with the question, the remaining questions are shown again with different font sizes. At this point the decision maker has a clear feedback on what questions at this point in the configuration process would lead to a configuration faster. However, she can still choose to ignore these suggestions and make, for example, the decisions $Screen\,Size_{12.1''}$ and $Screen\,Resolution_{XGA}$.

The incremental calculation and as a result the dynamic adaption to the current

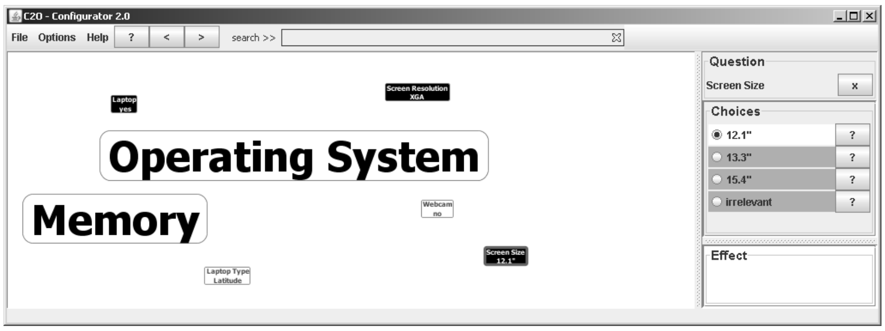
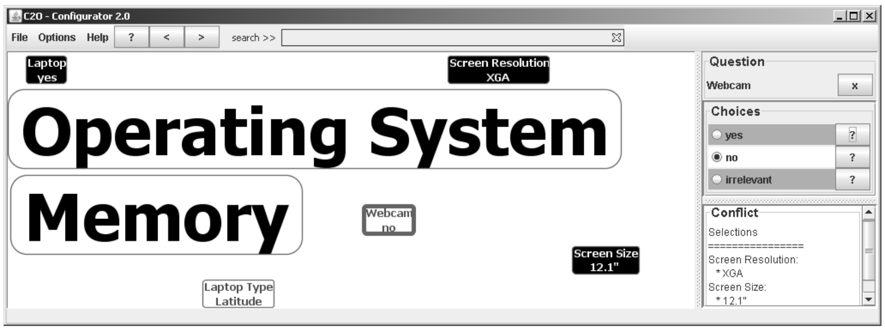


Figure 6.4: Visualization of derived decisions.



```
Selections
================
Screen Resolution:
  * XGA
Screen Size:
  * 12.1"
```

Figure 6.5: Explaining an inconsistency.

state during the configuration process is evident in Figure 6.4, since the importance of the questions has changed. Furthermore, Figure 6.4 also depicts that decisions can be derived automatically by eliminating choices based on relations, visualized by the grayed question captions with the selected choice shown beneath.

6.3.3 Inconsistencies

Because of automatically derived decisions, but also eliminated choices due to relations, inconsistencies might be unavoidable. For instance, after the decision maker made the three decisions $Laptop_{yes}$, $Screen\,Size_{12.1''}$ and $Screen\,Resolution_{XGA}$ the decision $Webcam_{yes}$ is no longer available. As a result $Webcam_{no}$ is automatically derived and clicking on the $Webcam$ question shows that yes is no longer available. With the help of HUMUS the tool can now provide an explanation to the decision maker why $Webcam_{yes}$ is no longer available by clicking onto the "?" besides the choice yes, as depicted in Figure 6.5. However, the tool also allows the decision maker to ignore that this choice is disabled and lets her select it anyway, causing the inconsistency. At this point the tool supports two alternative strategies shown in Figures 6.6 and 6.7. In Figure 6.6 all contributing decisions are isolated, as visualized by the grayed out questions with their selected choice beneath including $Webcam_{yes}$ (in this case the decisions are light gray meaning that each individually is still viable). However, one could argue that explicitly

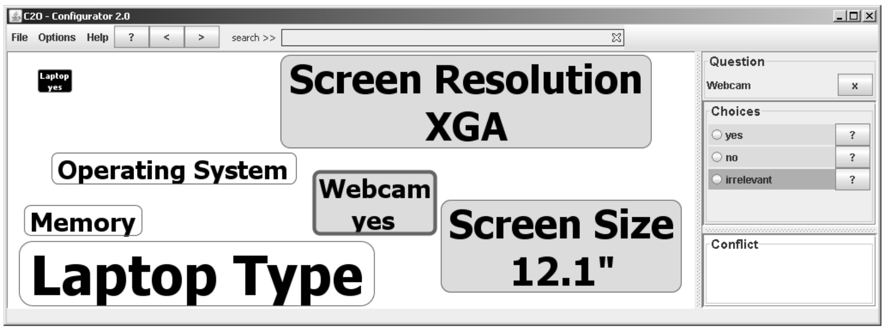


Figure 6.6: Causing an inconsistency.

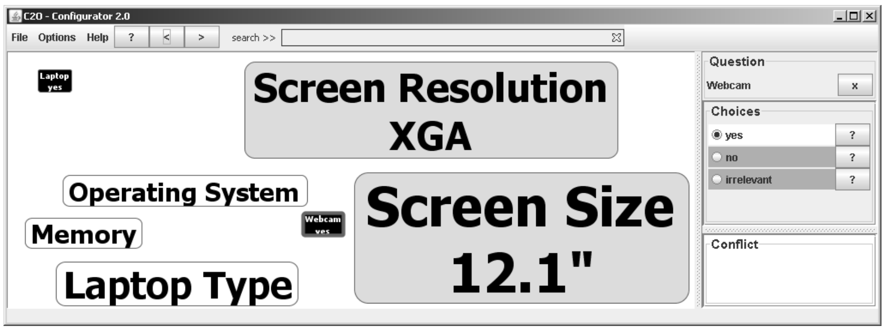


Figure 6.7: Causing an inconsistency with trust.

causing an inconsistency means that the decision that caused the inconsistency must be very important to the decision maker, so as an alternative strategy one can decide to trust this decision to be correct, as shown in Figure 6.7.

In either case decisions that were isolated from reasoning are kept in GUI and selecting questions to answer, e. g., *Laptop Type* now reveals a third state for choice visualization. In addition to a white background for choices that can be selected without causing an inconsistency (e. g. see Figure 6.2 both *yes* and *no* for the question *Laptop*) and to a dark gray background for disabled choices that would lead to a new inconsistency (e. g. see Figure 6.8 the choice *Latitude*) we have a light gray background. Choices with a light gray background can be selected without causing a new inconsistency. However, they have an impact on existing inconsistencies by either reducing the number of possibilities of how to fix them, or by resolving them automatically. To find out what the exact impact is the decision maker has to either click on the "?" besides the choice or simply select the choice. One example where a choice reduces the number of possibilities is shown in Figure 6.9, by selecting $Laptop\,Type_{Vostro}$, the choice *XGA* is disabled for the *Screen Size* question, as a result it is then visualized with dark gray background. Another example is shown in Figure 6.10, by selecting $Laptop\,Type_{Inspirion}$, only the choice *WXGA* is viable for the *Screen Resolution* and $Screen\,Size_{12.1''}$ becomes compatible again with the other user decisions. As a result the answer for the *Screen Resolution* is changed automatically, which is indicated to

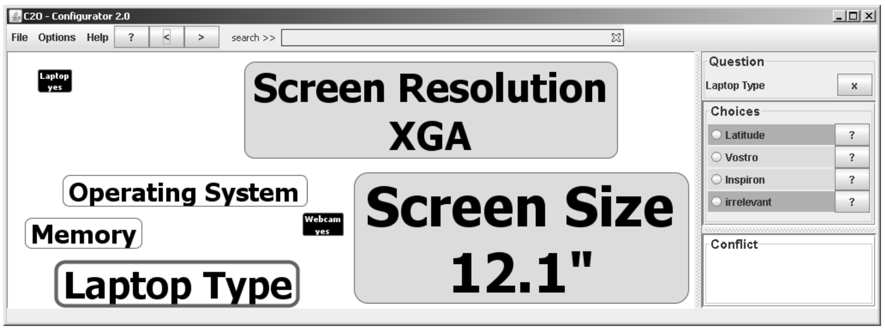


Figure 6.8: Choices impacting the inconsistency.

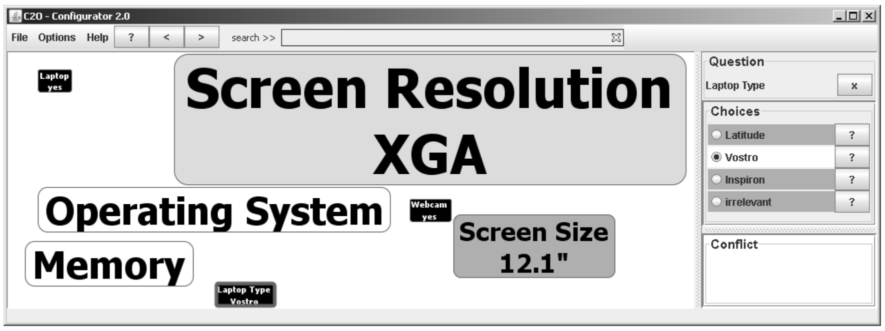


Figure 6.9: Reducing the number of possible fixes for an inconsistency.

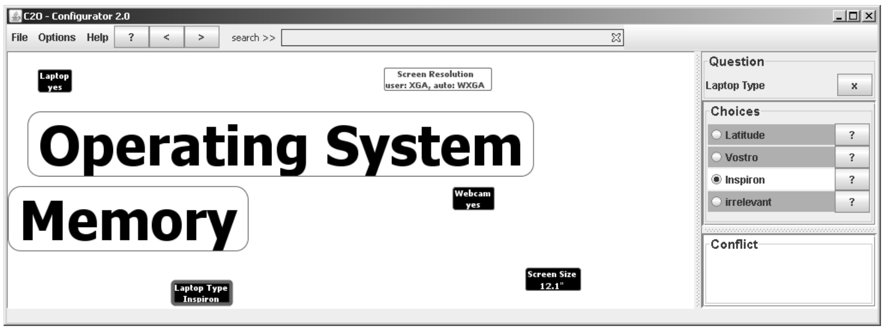


Figure 6.10: Automatically resolving an inconsistency.

the decision maker and the decision $Screen\,Size_{12.1''}$ is again displayed as a consistent user decision and the inconsistency is resolved.

Chapter 7

Evaluation

“ The true delight is in the finding out rather than in the knowing. ”
– *Isaac Asimov*

Since user interaction scenarios are hard to evaluate and are often subjective, we focused our evaluation efforts on the reasoning techniques behind so far. Those can be tested and measured with the usage of random data, covering all possible scenarios from best cases to worst cases.

This chapter will be separated into several sections, each dealing with the evaluation of an independent approach. Section 7.1 will present our evaluation for the guidance calculation, followed by the evaluation of living with inconsistencies in Section 7.2. After that results on the potential of living with inconsistencies for fixing purposes will be presented in Section 7.3, followed by a discussion about the implications of those results for decision-making in Section 7.4. The chapter will be round of with preliminary results from the UML domain in Section 7.5.

7.1 Guidance Calculation Results

As explained in Section 5.2, our approach is incremental as it computes the next question in the sequence on demand, depending on the current stage of the configuration process. Our approach is therefore always in compliance with *our vision to allow the decision makers the freedom to answer those questions first that are most important to them.* The following evaluations thus focuses on the quality of *of automatically arranging the sequence of questions to minimize user input* (see Table 4.1, goal O1).

7.1.1 Case Studies

We evaluated our approach on seven case studies with a different number of questions (#q), choices (#c) and relations (#r), the details are shown in Table 7.1. The *Dopler*

Table 7.1: Case studies overview

	#q	*#c*	*#r*	*#configurations*	
				actual	*theoretic*
Dopler	14	48	8	105	3,870,720
Graph	29	70	24	192	5.2E+10
Dell1	28	147	103	not computable	7.3E+17
Dell2	24	147	23		2.9E+15
Web	42	113	31		4.2E+17
CC-L2	59	137	20		2.4E+20
EShop	286	703	147		6E+115

model of the DOPLER product line tool suite [65], the complete laptop configurator reverse engineered from the DELL homepage (during the period of February 9th till February 12th 2009) in two versions (*Dell1* and *Dell2*), the *CC-L2*, a steel plant manufacturing product line model used by an Industry partner [32], and feature models published on the online feature model repository of S.P.L.O.T. (Software Product Lines Online Tools website [1]). An Electronic Shopping system (*EShop*) by Sean Quan Lau, a web portal (*Web*) by Marcilio Mendonca and a graph feature model (*Graph*) by Hong Mei. The *EShop* model was the biggest one with 286 questions. Note that the feature models were automatically converted into our own decision models (basically features are represented by questions with up to three answers: yes, no, irrelevant) to be used with our tool, hence the characteristics differ from those given on the S.P.L.O.T. website. The reason for there being two DELL models is to show that significant differences in how the product lines are modeled (while enabling the same set of products) do not appear to have a large impact on our approach. As can be seen, the *Dell1* makes extensive use of relations as opposed to the *Dell2*.

To evaluate our approach, we compared it with the theoretic worst case and best case. In the worst case, one needs to answer all questions (equal to the number of questions in the model and thus the same for every configuration). The best case is different for every configuration. To evaluate our approach and gauge its optimality, we thus randomly selected configurations, reverse computed their best and worst cases and then evaluated how our approach compared to these. Our approach's optimality is thus measured in terms of its position relative to the theoretic best case and worst case as depicted in Figure 7.1. For validation purposes, all configurations for the smaller models and 10,000 random configurations for the bigger models have been analyzed for high statistical significance. The figure also shows the confidence intervals (95%) but they are often so small that they are no longer visible due to the large sampling. The results show that our approach is on average no more than two questions away from the theoretical minimum for the small models and six questions for the *EShop* and *Web* model. Our approach is thus 78-99% optimal and *we achieved our goal of minimizing user input.*

[1] http://www.splot-research.org

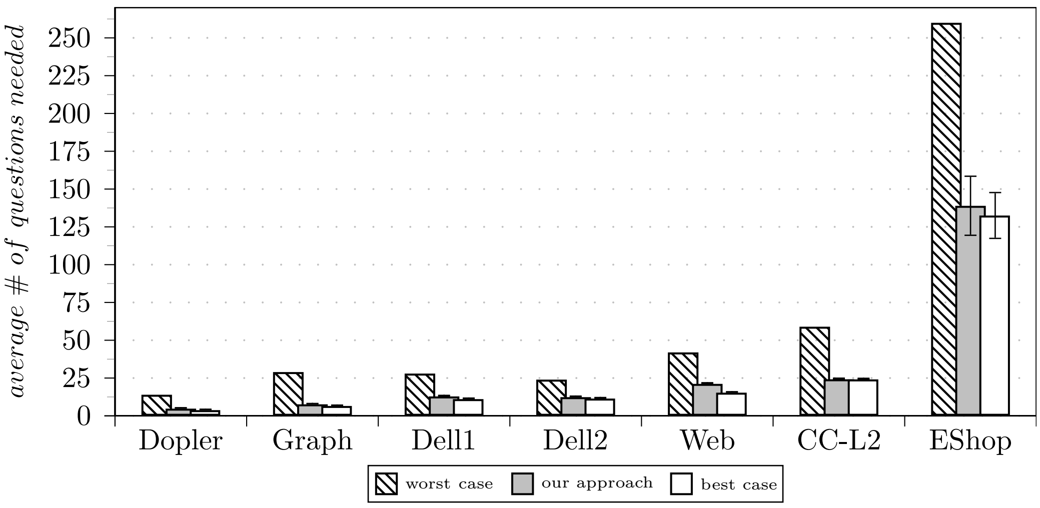


Figure 7.1: Comparison optimality of the guidance calculation.

However the decision maker could ignore the proposed sequence and answer in any sequence (highest degree of freedom possible), while still benefiting from the reasoning about choices being eliminated and questions being answered. The question is, whether following the proposed sequence is superior to such a more or less random sequence. The results for each model are shown in Figure 7.2 as a percentage proportionally to all configurations investigated. For each configuration an independent two-sample t-test for unequal variance and sample size was performed (with a p-value of 0.05), to test if there is a significant difference between our proposed and a random sequence. These individual results were categorized into four classes: our approach is significantly better, better, worse and significantly worse. This comparison was based on 1,000 randomly sequenced runs per configuration. A Kolmogorov-Smirnov comparison of all the values for each model showed a significant difference with a p-value < 0.0001 for each model. As can be seen, there were cases where a random sequence performed better or even significantly better than our approach. This happens particularly with the first question answered. Since at that time, nothing is known about the decision maker's intention (no question was answered), our approach always asks the same question first. In some cases, this first question may not be an optimal question to ask for a certain target configuration and the random approach is more likely to pick this optimal, first question – hence, the random approach may out-compete our approach occasionally. However, we do see that our approach is almost always significantly better. Our approach to guided decision-making is thus an improvement over any random approach (likely representing novice users and perhaps even expert users for very complex configuration problems).

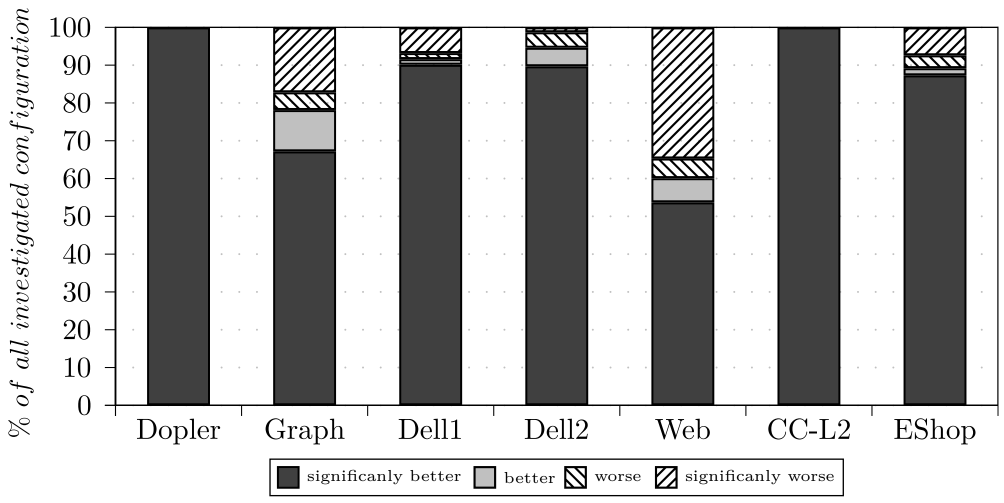


Figure 7.2: Overview of t-test results comparing our guidance calculation approach to a random selection.

7.1.2 Computational Complexity

Our proposed approach is qualitative superior but does its computation scale? The computational complexity of our approach depends on three variables: the number of choices, the number of relations, and the number of questions.

To assess response time, we determined the time needed for proposing a sequence. We evaluated every model and assessed the performance needed at varying degrees of questions answered. The average times needed (each model and percentage point was evaluated by 100 different configurations for high statistical significance) are shown in Figure 7.3, according to how many questions were answered as a percentage of the total number of questions. Their respective confidence intervals (95%) are not shown to avoid further clutter and are in the range of 1% up to 12% (only in a few cases) of the average values. The performance tests were conducted on an Intel® Core™ 2 Quad Q9550 @ 2.83 GHz with 4GB RAM (although at the moment only one core is used).

Our finding is that there exists an initial time cost in calculating the sequence for the first time (0 questions answered, especially high for the *EShop* model 56 seconds and therefore cut off in the graph). After this initial cost (which can be pre-calculated and cached), simulation time decreases with questions answered because eliminated choices do not need to be considered any longer. The fluctuations in calculation time can be explained by the different number of questions being affected by the elimination effect of the last question answered before the measurement. As can be seen, the simulation time is acceptable and grows linearly only with product line size. Given that the largest evaluated product line contains over 280 questions and our approach's response time was almost always <2 seconds, we thus believe that this approach is quite scalable and applicable to many product lines today, which answers the question if our approach can be used in real-time (see goal S1 from Table 4.1). Nonetheless, we

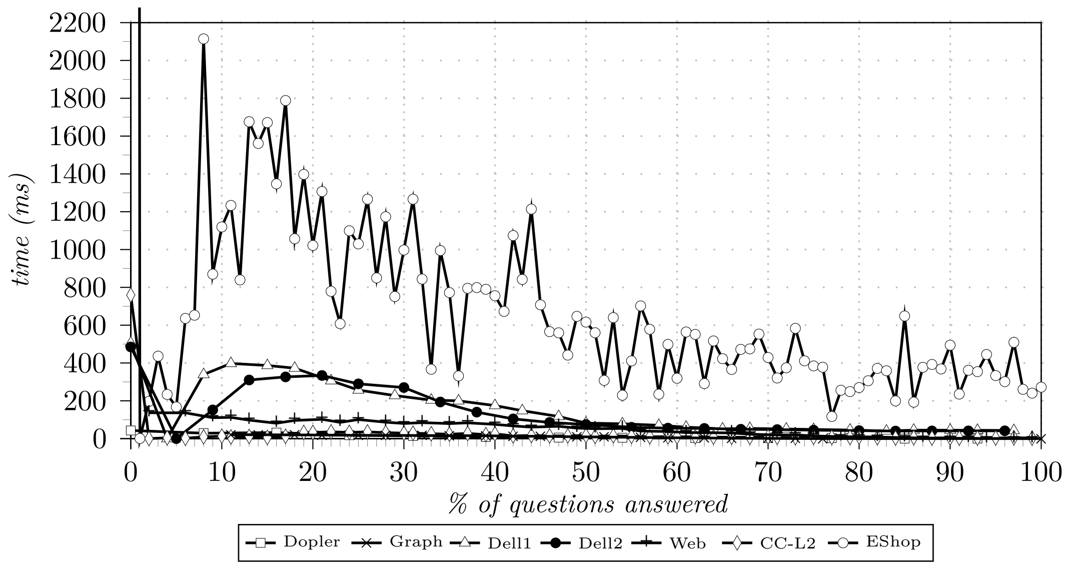


Figure 7.3: Response time measurements.

Table 7.2: Decision models used for evaluation.

Model	*#q*	*#c*	*#r*	*#literals*	*#clauses*
Dopler	14	48	8	51	274
Graph	29	70	24	70	163
Dell1	28	147	103	137	2,127
Dell2	24	147	23	142	2,540
Web	42	113	31	113	253
CC-L2	59	137	20	135	257
EShop	286	703	147	703	1,440

see ample opportunities for improving even this performance which will be explored in future work.

7.1.3 Memory Consumption

As mentioned in section 7.1.2, simulation results are cached. In particular, each result of a `SelectionGain` simulation for each choice is cached. Therefore the cache grows linearly with the number of choices in the model, with one reference to the choice and one long containing the gain value.

7.2 Living with Inconsistencies Results

To assess the differences of isolation strategies, we evaluated them on our case studies given in Table 7.1 Key characteristics of those models are repeated in Table 7.2 like the number of questions ($\#q$), the number of choices ($\#c$) in the model, and the number of relations ($\#r$) between questions in the model. In addition the number of literals and clauses needed after the transformation into CNF are stated.

7.2.1 Objectives and Questions

The objectives of the evaluation of living with inconsistencies are to investigate the effects of different isolation strategies on user guidance in decision-making scenarios. Specific questions we answer, in accordance with our goals from Table 4.1 regarding goals O2, C3, S2, G1 and G2, are:

1. What is the difference among the isolation approaches in terms of incomplete reasoning (missing guidance)?
2. What is the difference among the isolation approaches in terms of incorrect reasoning (faulty guidance)?
3. How many questions have to be revisited using the different approaches?
4. How are the approaches handling multiple inconsistencies at the same time?
5. How do the different isolation approaches for tolerating inconsistencies in SAT-based reasoning scale?

7.2.2 Execution

As mentioned in section 5.3.3 in order to be able to evaluate the isolation approaches with respect to our objectives, we need to know how an inconsistency is going to be fixed. For that purpose we generated one thousand valid configurations for each model (without any inconsistencies), to have a statistical significant sample size. In each configuration we injected up to three defects, for the purpose of getting an idea how the isolation strategies are effected when multiple defects are present three were sufficient because more defects are treated in the same manner. We seeded the defects by randomly changing decisions of each configuration to cause inconsistencies and treating the original decisions as the fixes. We then simulated the decision-making involved. Since each configuration contained defects, the decision-making eventually encountered an inconsistency. Starting at this point the simulation was continued using the different isolation approaches described in Section 5.3.3, while the reasoning data was collected for each simulation. The approach works with defects injected at any stage from the very beginning to the end. However, for meaningful observations on completeness and correctness it is not useful to inject defects at the beginning or end due to the following reasons: *i)* assuring that uninvolved assumptions are made before the inconsistency detection, in order to reveal differences between the isolation approaches and *ii)* potentially leaving assumptions to be made left after the isolation, in order to be able to measure the impact of the different isolation approaches onto the reasoning with sufficient data points. This is why we injected defects in the range $[0.2 * \#q,\ 0.8 * \#q]$.

To put our results concerning incomplete and incorrect reasoning into perspective, we also calculated the hypothetical best and worst cases. The worst case is that no SAT-based automation is available while tolerating inconsistencies (i. e., because it fails due to UNSAT and no strategy is in place for isolation). This implies that the worst case is about answering each question without any reasoning in place that helps guide the user. The ideal case is to isolate the defects only. We, of course, knew for each configuration where the defects were, since we seeded them. Thus, we can think of our knowledge as the optimal isolation strategy (although it should be clear that in a non-experimental setting this extra information would not be available and the ideal is not computable). Having the ideal available is useful for understanding how far away the various isolation strategies are from the optimum. As was mentioned above, Disregard All, Skip, and HUMUS compute unique isolation solutions while MaxSAT typically computes a non-deterministic solution out of several alternatives. To account for the randomness of the MaxSAT isolation we thus investigated it from its normal case (the random selection of a solution) as well as from its worst case (the solution does not isolate the defects and thus causes the maximum harm in terms of incorrect reasoning).

The results present the comparison of the different isolation strategies with the ideal isolation simulation data starting at the point during the decision-making process the first inconsistency was discovered. Looking at our example from Table 5.1 this would mean starting the comparison at the fifth question and ignoring literals belonging to questions already answered (the grayed out areas in the table). For instance to compute the incomplete reasoning data, we counted the number of literals remaining (no assumptions were derived for them) after each question answered, excluding literals belonging to already answered questions. To compute the incorrect reasoning data we counted the number of assumptions made different from the ones of the ideal simulation data, again excluding literals belonging to already answered questions. To compute the revisitation sizes the number of assumptions isolated were counted for each approach. And last but not least the needed isolation times were measured during the simulations.

7.2.3 Results

To get an idea what the raw evaluation data looks like, Figure 7.4 shows absolute results of three individual runs to assess incomplete reasoning with the HUMUS isolation strategy for the *Dell1* model. On the x-axis the number of questions answered is shown, while on the y-axis the number of unassigned literals remaining is depicted. While these individual runs look quite different, their characteristics are basically the same, which is reflected in the overall results. The meaning of the top curve for example is that the inconsistency was detected after question 16 was answered, at this point after the HUMUS isolation took place there were 103 unassigned literals left either belonging to the remaining 12 questions that were not yet answered nor looked at. After answering question number 17 through reasoning the number of unassigned literals was reduced to 41 and so on.

For calculating the overall results covering all models, these individual runs were normalized on the x-axis between the question where the inconsistency was detected

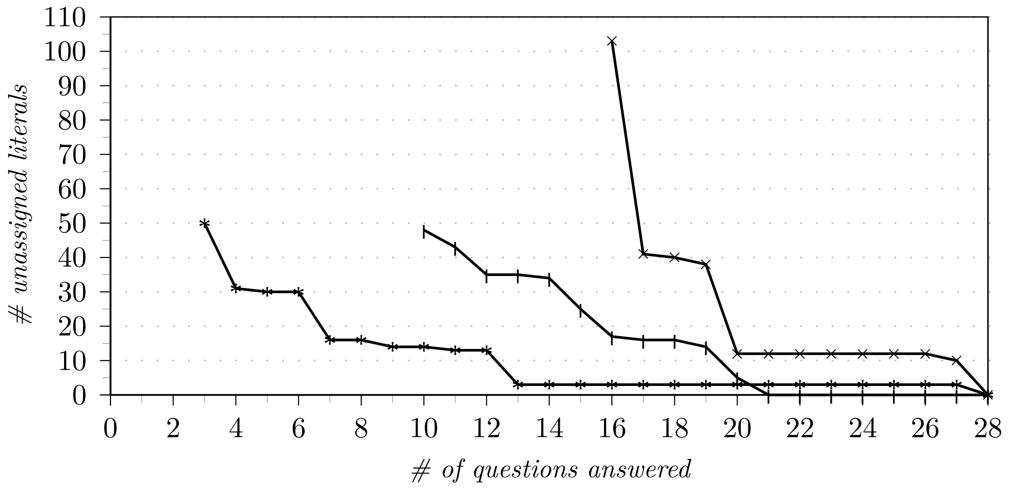


Figure 7.4: Incomplete reasoning progression runs with HUMUS isolation strategy for the *Dell1* model.

(0%) and the number of remaining questions in the model (100%) that need to tolerate the inconsistency. The y-axis values were normalized between the number of literals representing all the choices of the questions left at the time the inconsistency was detected (100% equal to no reasoning) and zero literals (0%). For example, the top curve from Figure 7.4, the point questions answered 17 and unassigned literals 41 will get normalized between 16 questions answered (0%) and 28 questions answered (100%) resulting in a x-value of $\sim$ 8.3%, the 41 unassigned literals will get normalized between 0 unassigned literals (0%) and 105 unassigned literals (100%), from the no reasoning simulation run, resulting in a y-value of $\sim$ 39.05%. Based on such normalized runs the averages shown in Figures 7.5 and 7.6 were calculated. Due to the very large number of configurations investigates, the averages have little variation and combined form a perfect line.

7.2.3.1 Incomplete Reasoning (Single Defect)

In Figure 7.5 the average results of all configurations evaluated for incomplete reasoning are shown (objective 7.2.1-1). The 95% confidence intervals are not shown to avoid further clutter, they are in the range of 0% up to 3.14% for all data points. Due to the fact that we only evaluate incomplete reasoning on questions not yet answered by the user, even with no reasoning 0% incomplete reasoning can be reached at the end when no questions are left. On the other hand even with knowledge that normally would not be available (ideal case) for about 30% of the remaining literals no assumptions can be derived at the time of the inconsistency detection. The results correspond to the general discussion beforehand in Section 5.3.4.1 but also hold some surprises. The Disregard All strategy which we expected to be the worst-case is in fact a good alternative to no reasoning at all and not just because it enables automations. The HUMUS isolation strategy on average only starts out with about 10% more incompleteness than the

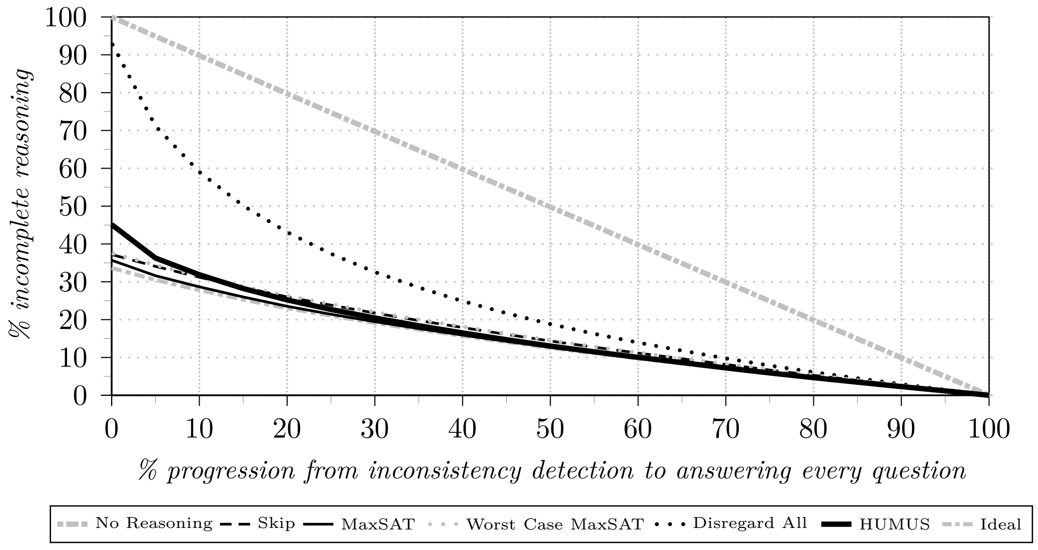


Figure 7.5: Incomplete reasoning progression results combining all case study systems.

MaxSAT strategy and quickly closes the gap to the MaxSAT strategy (to the worst case MaxSAT at ~10% and the random MaxSAT at ~45% of the remaining configuration process). Overall it seems that the investigated models contain significant overlapping constraints resulting in a more complete reasoning than expected.

7.2.3.2 Incorrect Reasoning (Single Defect)

In Figure 7.6 the average results of all configurations evaluated for incorrect reasoning are shown (objective 7.2.1-2). Again the 95% confidence intervals are not shown to avoid further clutter, they are in the range of 0% up to 0.36% for all data points. The results for incorrect reasoning are also quite interesting. Overall the amount of incorrect derived assumptions seems not very high – in the worst case only about 6%. Another interesting fact is the self correcting ability of the MaxSAT strategy since reasoning with the defect sometimes leads to its re-detection in form of new inconsistencies due to the overlapping constraints. Every time such a defect is re-detected, the MaxSAT strategy has a chance to isolate the defect, increasing its chance over time and resulting in a rapid decrease in incorrect reasoning. The worst case MaxSAT obviously does not get that benefit and the Skip strategy also has no chance to eliminate the defect once it is included in the reasoning process since the defect must be located at an earlier point in time during the configuration. MaxSAT is thus an interesting alternative to HUMUS if incorrect reasoning is acceptable temporarily. However, do note that using MaxSAT in this manner is more equivalent to postponing inconsistencies since the defect is detected multiple times in form of different, yet related inconsistencies.

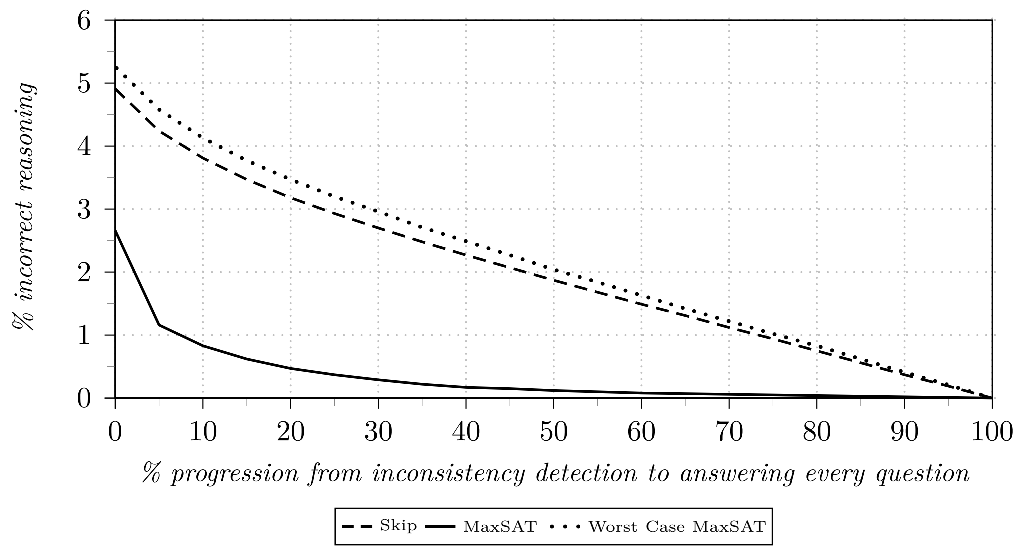


Figure 7.6: Incorrect reasoning progression results combining all case study systems.

7.2.3.3 Revisitation (Single Defect)

The results show that the Disregard All isolated about 41% of the literals on average. The HUMUS isolation strategy (3,45%) on average isolates about twice as many literals as MaxSAT (1,27%), but less than Skip (3,94%) and the worst case MaxSAT (4,19%). This can be explained by additional isolations needed (increasing the number of literals in isolation) in case of additional inconsistencies, which occurred more frequent in the worst case (objective 7.2.1-3).

7.2.3.4 Multiple Defects

We also conducted simulations with up to three defects (objective 7.2.1-4). To avoid additional clutter only the HUMUS and MaxSAT strategies are shown in the figures. As evident in Figure 7.7 (95% confidence intervals are in the range of 0% up to 1.33% for all data points) the results for incomplete reasoning look quite the same except that naturally it increases for all strategies, since more defects mean more isolations, the same is true for revisitation results. However, the results for incorrect reasoning in Figure 7.8 (95% confidence intervals are in the range of 0% up to 3,01% for all data points), in addition to a slight overall increase, clearly indicate a slower decrease in incorrect reasoning over time. This effect can be explained by the nature of MaxSAT to only isolate as little as possible and inconsistencies that involve common correct assumptions. Given our example of the three incompatible decisions $Screen\,Size_{12.1''}$, $Screen\,Resolution_{XGA}$ and $Webcam_{yes}$, if two of them were injected defects then MaxSAT would always isolate the third correct one, until additional decisions conflict with the defects and as a result solutions with one decision isolated would not work anymore.

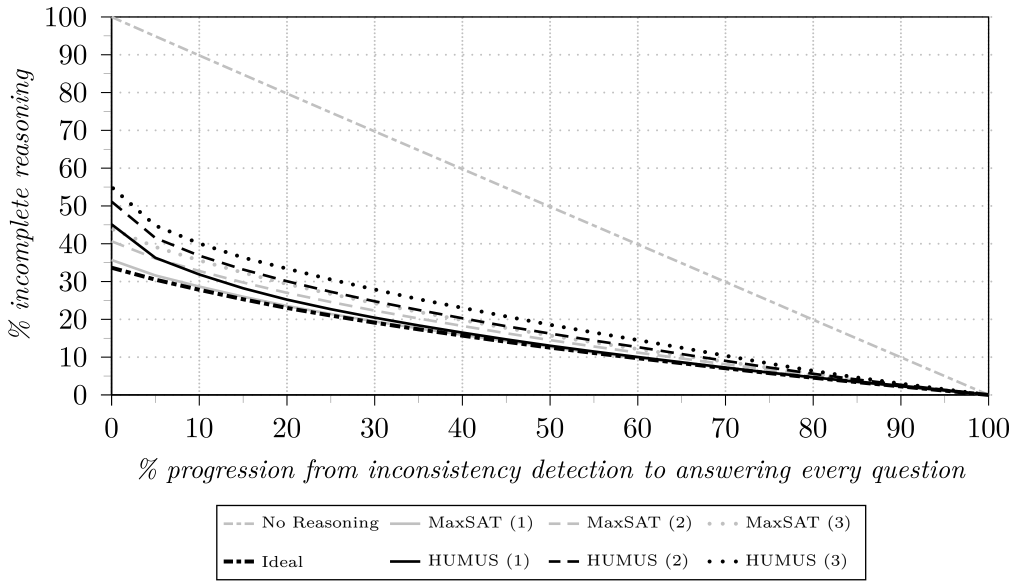


Figure 7.7: Incomplete reasoning progression results combining all case study systems with multiple defects.

Table 7.3: Scalability test results on artificial SAT problems.

#Contributors	*10*	*100*	*1,000*	*10,000*	*100,000*
MaxSAT	1ms	1ms	3ms	86ms	∼6s
HUMUS	1ms	3ms	241ms	∼28s	∼1h

7.2.3.5 Scalability

We also conducted performance tests on an Intel Core 2 Quad Q9550 @2.83 GHz with 4GB RAM, although only one core was used for the time being. The computation time needed for all models and the different isolation approaches was between 0ms and 1ms per computation. The evaluated models are not the largest SAT models around, however in context of this domain they are quite large and we have shown that the approach scales for our case studies (objective 7.2.1-5). However, further evaluations on artificial SAT models (Table 7.3) show an exponential growth but acceptable performance for inconsistencies involving up to 10,000 assumptions. While those artificial SAT models may not represent the structure of typical decision models well, they represent the worst case structure for our implementation. Those artificial SAT models consist of a single clause containing n literals $(l_1 \vee l_2 \vee \ldots \vee l_n)$ and then all literals are assumed to be set to *false* resulting in an inconsistency because at least one literal has to be set to *true*. While this kind of SAT model may seem trivial, our HUMUS implementation determines all MCSes by searching for all MSSes and taking the complementary set of each one. As a result the number of MSSes grows linearly ($n - 1$, where n is the cardinality of the elements in the clause), but so does the number of elements in each

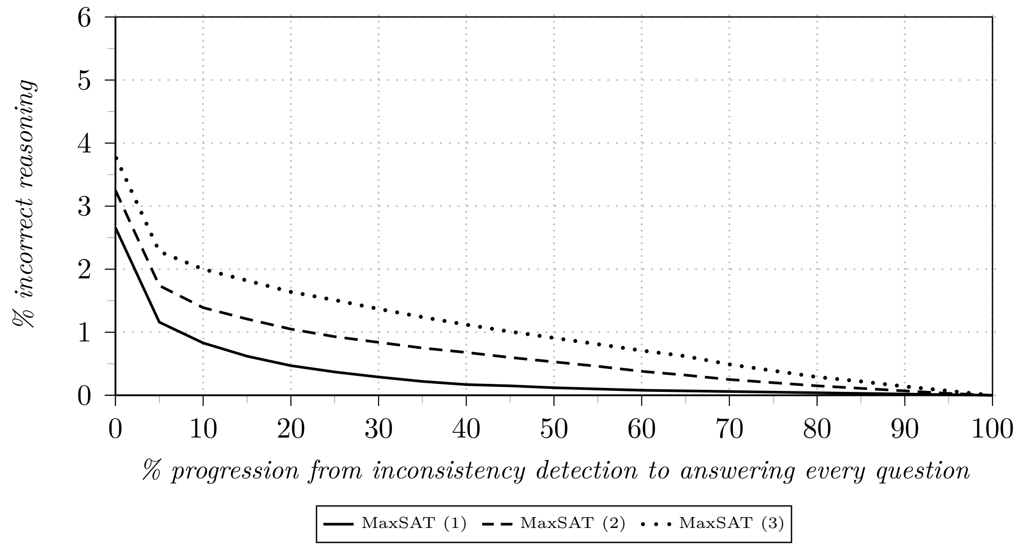


Figure 7.8: Incorrect reasoning progression results combining all case study systems with multiple defects.

MSS. Our current implementation builds a single MSS bottom-up which means there are $n!$ SAT calls necessary, to determine one MSS.

7.3 Potential of Living with Inconsistencies to Fix Inconsistencies

As demonstrated earlier HUMUS is useful for living with inconsistencies. However, there can be an additional benefit to living with inconsistencies, namely the ability to automatically fix inconsistencies or reduce the number of fixing possibilities. For the evaluation we used the models from Table 7.2 and in addition to simulating the different isolation strategies we calculated the number of possible fixes at the point of the inconsistency detection and after answering each following question.

7.3.1 Automatically Fixing Inconsistencies

At the time of the failure, three situations are possible:

1. A single fix is already computable when the inconsistency is detected and the decision that caused the inconsistency is trusted to be correct (*fixable at failure*).
2. A single fix is not computable when the inconsistency is detected but becomes computable at some point if follow-on decisions can be trusted (*fixable later*).
3. A single fix is not computable even at the end, with follow-on decisions trusted. However, the number of choices are reduced making it easier to fix the defect (*fixable with user input*).

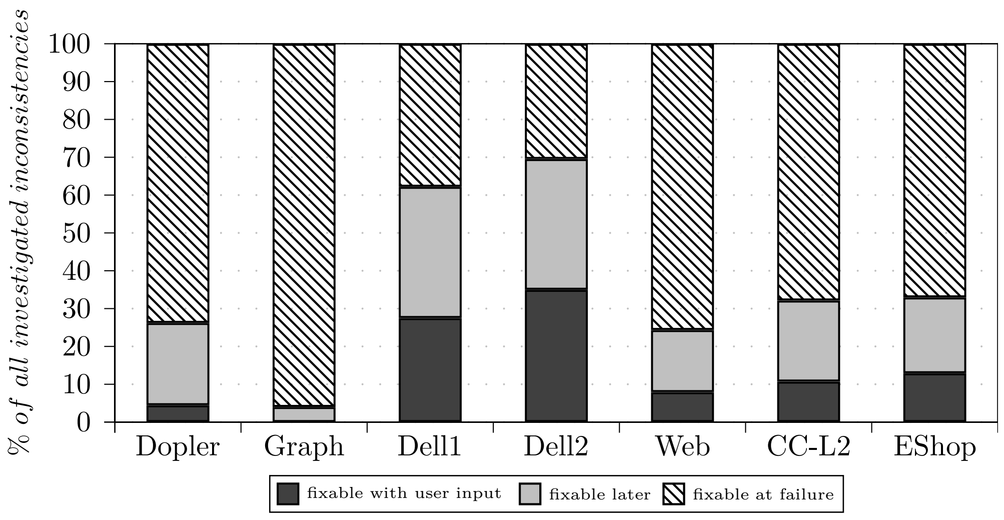


Figure 7.9: Distribution of fixable situations without additional user interaction.

We can see in Figure 7.9, that 30-96% of defects were fixable at the time the inconsistency was detected (i. e., there is only one option available and fixing it is trivial). The remaining defects required more user input. However, 4-35% of the remaining defects were fixable simply by letting the decision-making continue (while the inconsistency was tolerated) without requiring additional information. Tolerating inconsistencies and making new decisions, independent of the inconsistency, thus fixes defects automatically in many situations. Even in the cases where defects were not fixed automatically, the choices for fixing them got reduced considerably during tolerating. This benefit is discussed next. This demonstrates that the fixing of defect is not trivial in many cases.

7.3.2 Automatically Reducing Choices for Fixing

For the 4-70% of defects in Figure 7.9 that were not fixable at the time the inconsistency was detected, Figure 7.10 presents the actual number of possible fixes at the time of the detection and at the end (with their respective confidence intervals of 95%). Three situations are distinguished here:

1. The number of possible fixes if the HUMUS is computed at the end after all decisions have been answered (*worst case*).
2. The number of possible fixes if the HUMUS is computed when the inconsistency is detected (*failure*).
3. The number of possible fixes if the HUMUS is computed when the inconsistency is detected, but further reduced by using follow-on decisions with the assumption that they can be trusted (*with tolerating inconsistencies*).

The first situation does not distinguish between decisions made prior to and after the detection of the inconsistency. Follow-on decisions are not trusted to be correct and

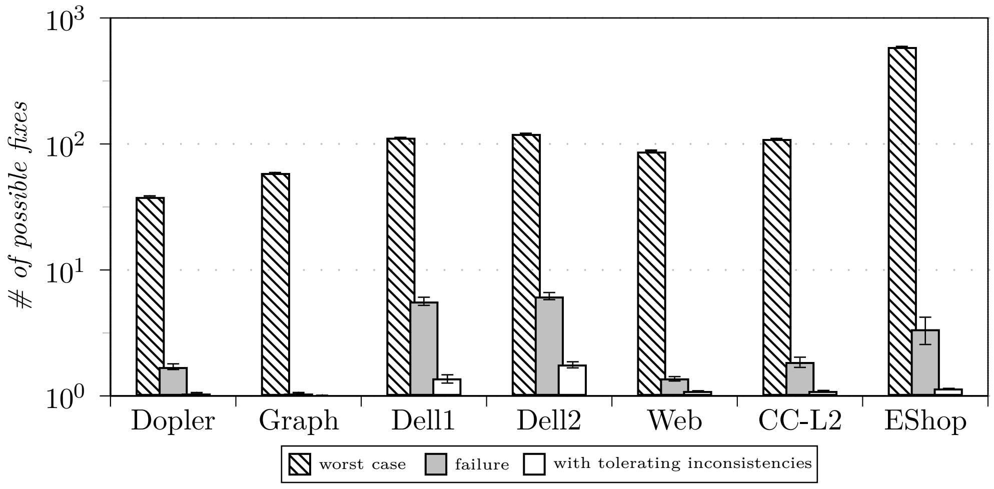


Figure 7.10: Overview over the number of possible fixes.

thus there are many choices for fixing the inconsistency. The second situation recognizes that the defect must be embedded among the decisions made prior to the detection. This simple knowledge vastly reduces the number of possible fixes. Tolerating the inconsistency then further improves on this by considering the effect of decisions made after the detection (i. e., while tolerating inconsistencies) onto the decisions made prior (the optimal is '1'). Tolerating inconsistencies thus makes it easier to fix defects. The improvements observed in Figure 7.10 are more substantial in decision models where there are the more relations (e. g., *Dell1* model). This is easily explained. The more relations (constraints) there are in a model, the more knowledge can be inferred and thus the more restricted are the number of possible fixes.

In Figure 7.11 the progression of this improvement is shown relative to the percentage of decisions remaining until the end. We see that it is not always necessary to continue decision-making until the end to get the most out of tolerating inconsistencies. We observed that at about 50% of the remaining questions answered after the detection (while tolerating inconsistencies) are enough to achieve 80-95% optimal reasoning and that every decision made after the detection simplifies the fixing of the defect. Note that the 0% marker on the x-axis corresponds to the point of the detection and the 100% marker to the end of decision-making. The y-axis denotes the percentage of choices reduced for fixing defects compared to the optimum which is equal the number of choices reduced at the end (once all user input is known). Note that we excluded all those cases where no more reduction was possible after the detection (this data would be always at 100% and therefore distort the other results).

7.3.3 Fixing Multiple Defects

We realize that for fixing inconsistencies through tolerating, our assumption that new decisions can be trusted is not always realistic, especially considering the fact that an inconsistency may be the result of multiple defects. However, it provides us with results

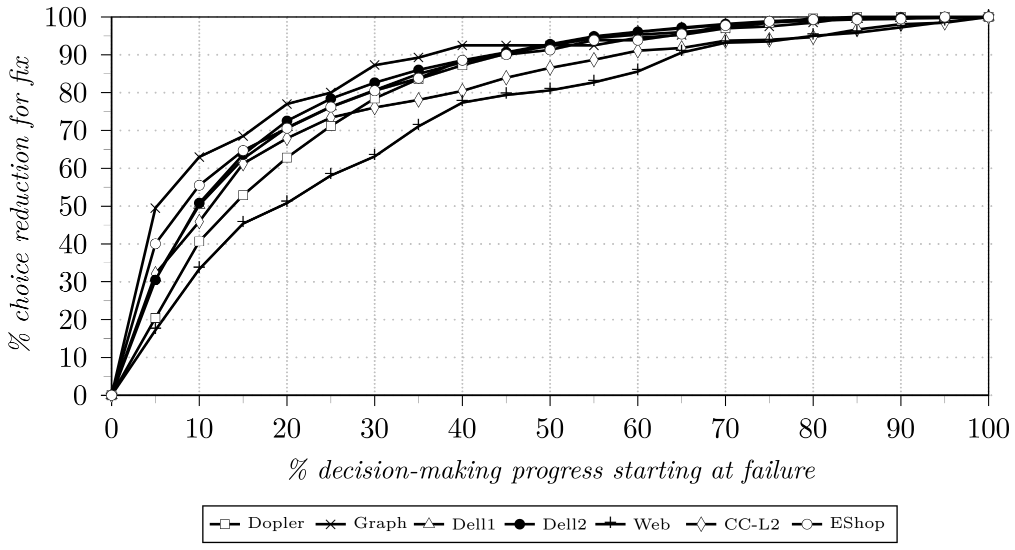


Figure 7.11: Normalized progression of fix reduction.

under optimal conditions and just by remembering the point of failure in combination with HUMUS, the number of possible fixes can be significantly reduced compared to searching for a fix at the end of the configuration as shown in Figure 7.10. This always works even if new decisions cannot be trusted and need additional fixing.

In other words, what this means is that as soon as a second inconsistency is detected during configuration all assumptions made on how to fix the first inconsistency based on decisions made between the detection of the first and the second inconsistency need to be isolated with their decisions, otherwise incorrect reasoning might be used to reduce the number of fixes. However, as long as not all new decisions are involved in the new inconsistency, the number of fixes can be reduced through reasoning. We presume that the effect on incomplete reasoning is bigger than just with tolerating, investigating this will be part of our future work.

7.4 Implications for Decision-Making

Automatically determining the sequence of questions to reach each possible configuration with answering a minimal amount of questions, unburdens the designer of decision models from having to think about the sequence. In combination with our vision to give decision makers the freedom to answer any question at any given time, this allows decision makers to answer questions in the sequence they want or in the sequence our approach suggests or even combine both ways.

Concerning the living with inconsistencies, all four strategies are useful to allow automations to continue working in the presence of inconsistencies. In essence both MaxSAT and HUMUS are viable options for tolerating inconsistencies although, at least for this domain, HUMUS appears superior to MaxSAT given that it avoids incorrect reasoning altogether (which we believe often to be worse than incomplete reasoning) and that its incomplete reasoning and revisitation effort is only briefly worse than that of MaxSAT. It also allows decision-making tools to visualize all contributing decisions of an inconsistency, which could help users to make an informed decision on how to fix an inconsistency, which is in our opinion superior to more or less random suggestions. It also shows that living with inconsistencies is not only advantageous from a usability perspective, but can also be helpful in resolving inconsistencies. The approaches overall also scale well and are suited for usage in an interactive tool.

7.5 First Results on the Feasibility of our Approach in UML Modeling

One possibility of how to use our approach of minimizing user input in the more general UML modeling scenario is to treat the resolving of inconsistencies in UML as decision-making. In which case the fixing of inconsistencies can be seen as answering a question about how to fix it, with all fixing possibilities as choices to choose from. In order for our approach to be effective or even matter, we need to know that fixing one inconsistency has an impact, either by causing new inconsistencies or resolving / reducing the number of choices for resolving other inconsistencies. Another aspect was to find out if inconsistencies in UML models are structured similar to inconsistencies in decision-making. Due to the fact that UML constraints are typically evaluated individually, in contrast to checking the whole system as with SAT solvers, it is our presumption that an inconsistency detected in one constraint identified by all accessed model elements is rather more equivalent to a MUS than the HUMUS. So if a model change causes several inconsistencies at a time we can argue that those inconsistencies not only are structurally related but also semantically. For that reason we performed a basic experiment to explore if inconsistencies have overlaps in their fixing locations – in other word are structurally related – so that minimizing user input is an option and in addition explore the benefits (reduction in fixing locations) of assuming that there exist a semantic relation between inconsistencies.

Our results are based on a set of four models from industrial partners. Due to proprietary information we are not allowed to present any specific details of the models. The model sizes range from 1,200 to 33,000 model elements containing several types of diagrams like class diagrams, sequence diagrams and state-charts as well as use-case diagrams. The used design rules check generic aspects of the underlying meta model such as the definition of messages and transitions as operations in the corresponding class of the class diagram, as well as the direction of the associations in the class diagram regarding the messages of the sequence diagram, and if the connected classifiers of the association ends are included in the namespace of the association. These rules are derived from larger rules used by these industrial partners (see [66] for a complete list).

The results are based on the analysis of the accessed model elements and their properties, using Egyed's approach [5]. Each evaluation of a design rule inspects various properties of different model elements and we call a single element of this set a fixing

Table 7.4: Overview of analyzed UML models.

Model	*#Elements*	*#Inconsistencies*	*#Overlapping*	*%*
A	1,282	36	29	80.5
B	2,809	365	363	99.5
C	16,255	489	487	99.6
D	33,347	1,271	1,246	98.0

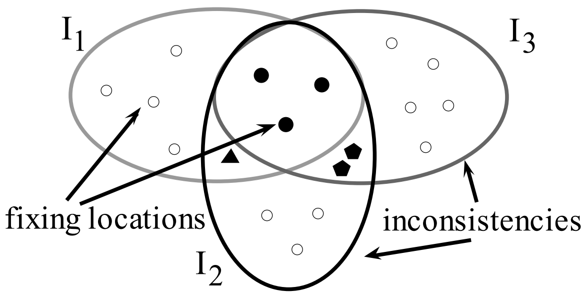


Figure 7.12: Fixing locations for three UML Inconsistencies.

location (Egyed refers to them as scope elements in [5]). An evaluated design rule is either consistent or inconsistent. In our results only inconsistent design rules are considered. The explicit side effects to other design rules, consistent ones included, are disregarded at this time because no fixing actions were calculated, although of importance for future work. Our focus at this time was solely on overlapping fixing locations, the basis for all possible fixing actions.

Table 7.4 provides some background about the models used for our evaluation. It shows the number of elements, how many inconsistencies were detected, how many of these inconsistencies have at least one fixing location in common with another inconsistency, and the percentage of such overlaps in inconsistencies. These results in Table 7.4 indicate that most of the inconsistencies found in these real world models have common fixing locations and are at least structurally related and that our minimizing user input approach could work also in this domain.

To measure the impact of knowing that inconsistencies are also semantically related, a short illustration is given next. Figure 7.12 shows fixing locations (dots, triangle, and pentagons) belonging to three different UML inconsistencies (ellipses marked I_1, I_2, and I_3). Each inconsistency has a set of fixing locations and their overlap is defined by the intersection of those sets. Applying this simple set operator, depending on the knowledge which inconsistencies should be fixed or not, the fixing locations to start out with can be derived easily. For instance if inconsistencies I_2 and I_3 should be resolved but not I_1, the fixing locations to start out with are represented by the pentagons in Figure 7.12.

We conducted a detailed analysis of the overlapping inconsistencies for each model, where we counted the number of occurrences of different sized overlaps between in-

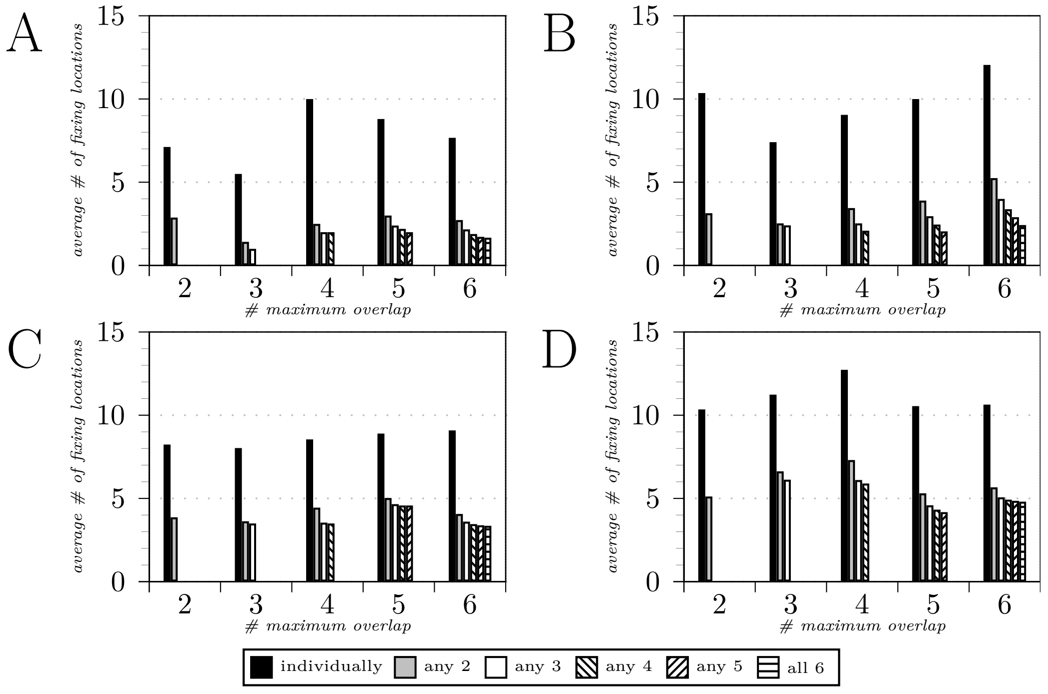


Figure 7.13: Reduction of fixing locations based on overlap size.

consistencies. For instance, counting these areas in Figure 7.12 would result in two occurrences of a size two overlap ($\{(I_1 \cap I_2) \setminus (I_1 \cap I_2 \cap I_3), (I_2 \cap I_3) \setminus (I_1 \cap I_2 \cap I_3)\}$) and one occurrence of a size three overlap ($\{I_1 \cap I_2 \cap I_3\}$). The size of an overlap is determined by the number of inconsistencies contributing to an overlapping area. Although the percentage of overlaps was very high in the analyzed models (see Table 7.4), the composition clearly showed that most of the overlaps were of size two, especially with bigger models. These results thus answer our first question of how often interrelated inconsistencies in real world examples exist.

Having established that overlaps among inconsistencies are common, in the following we will try to answer the second question of how many choices for fixing an inconsistency can be eliminated by considering the effects of interrelated inconsistencies. We will try to answer this question indirectly, again with the help of fixing locations since choices for fixing an inconsistency are based on fixing locations. By focusing on fixing locations, we try to avoid any bias from different fix generation methods – particularly because at present state-of-the-art is only able to compute fixing locations that cause inconsistencies completely but not necessary all locations affected by fixes as this requires human intervention [17].

Figure 7.13 shows the average number of common fixing locations for the different overlap groups. Each group has several bars indicating the impact on the number of fixing locations by considering additional inconsistencies of the overlap. For overlaps of size two, there are two bars: the first accounting for the average number of fixing locations if both inconsistencies are considered separately (no overlap effect). The second bar accounting for the average number of fixing locations if only the overlap among these two inconsistencies is considered. We thus see the effect of no overlap vs. pairwise overlap next to each other. The same is valid for overlaps of bigger sizes though additional bars are used to indicate additional overlaps. For example, the fixing locations of three overlapping inconsistencies can be viewed individually (first bar), pairwise as in any two overlapping inconsistencies out of the three (second bar), and altogether (third bar). This is simple to calculate. For example, if we wanted to calculate these results for Figure 7.12, they would be the following:

- *Overlaps size two:* the first bar would be the result of $Average(\mid I_1 \mid, \mid I_2 \mid, \mid I_3 \mid)$, the second bar the result of $Average(\mid I_1 \cap I_2 \mid, \mid I_1 \cap I_3 \mid, \mid I_2 \cap I_3 \mid)$.
- *Overlaps size three:* since the same inconsistencies are involved in the overlap size three the first two bars would be the same, the third however would be the result of $Average(\mid I_1 \cap I_2 \cap I_3 \mid)$.

These diagrams show consistently that the number of fixing locations is approximately reduced by half considering overlaps among two inconsistencies, but it also shows that this effect diminishes in larger groups – the limited subsequent effect may be due to the small group of inconsistencies considered.

In summary, we observe that assuming a semantic relation between inconsistencies significantly reduces the number of fixing locations to consider. In all four models, we found that a assuming a semantic relation between two inconsistencies already reduces the set of possible fixing locations by half. By reducing the basis (the locations) on which fixing actions are being calculated, the number of actual fixing actions may even be reduced more drastically. For further evaluation it is necessary to include more design rules, perhaps also considering application or domain specific design rules as they are likely to further reduce the set of possible fixing locations. What we have shown here is that there is a strong benefit in understanding the semantic relations to reduce the complexity of fixing inconsistencies.

Chapter 8

Related Work

“ A man should look for what is, and not for what he thinks should be. ”

– *Albert Einstein*

In general, there are many areas of research that aim at optimizing problems. Nonetheless, the problem we are trying to solve (i. e., optimizing the sequence of questions, living with inconsistencies), bears familiarities with a range of other areas of research. Such areas are, for example, decision support systems, dialog design, model checking, expert systems, other reasoning techniques, operations research or even Bayesian networks. The similarities and differences between our problem and approach and these areas of research are discussed next.

8.1 Other Reasoning Techniques

While our work primarily focused on Boolean satisfiability problems, there of course exist a range of other reasoning techniques with their own advantages and disadvantages. We will only discuss a few of these techniques in detail because either some aspects are similar to what we did or it would be possible to do them at all.

8.1.1 Constraint Satisfaction Problems

A decision model could be easily formulated as a constraint satisfaction problem [22] (CSP), in essence they are already constraint satisfaction problems that we transformed into SAT problems, which are a subset of constraint satisfaction problems. Generally speaking CSP is often used for model checking. However, the purpose of model checking is to find out whether there exists at least one valid configuration that satisfies the given questions, choices, and relations. In other words, a model checker would ensure that the model is valid. Our work presumes a valid model. A creator certainly validates such models and may choose to use a model checker for that purpose. One might argue that, internally, a model checker explores the many different question and choices (much like a human would) and model checkers are smart enough to optimize this process. For example, the sequence of variables plays an important role in such approaches. Heuristics like for instance minimum remaining values (MRV) [22] or constraint ordering heuristic (COH) [67] are used to speed up finding or rejecting variable assignments. Since finding valid configurations is not the focus of our work and those heuristics are designed specifically for this purpose they are not useful to us. It is also important to point out that model checkers are used to analyze the model until they find an example or counter example. Our approach is not interested in a just single example (and / or solution).

Of course CSP solvers are not only used for model checking, they are already being used to certain degrees in configuration scenarios. For example, to ensure the validity of models and support users in configuring product lines [41, 68]. This usage is reflected by users in our simulations ignoring the proposed sequence. However, to our knowledge no reasoning concerning determining the sequence of questions was used until now, also in addition the focus on living with inconsistencies is new.

8.1.2 Binary Decision Diagrams

Binary Decision Diagrams [69, 70] (BDD) are data structures used to represent Boolean functions. Directed acyclic graph structures are used to represent the relations between Boolean variables in a compressed way and can be used very efficiently for solving Boolean functions because no decompression is needed. For that purpose their solving speed largely depends on the variable ordering, not that different from model checking in general. Although there exist heuristics to determine a good variable order, the problem of doing so is NP-hard and not finding a good order might result in exponentially large graphs. SAT and BDDs represent the same entity [71], due to the fact that decision models can evolve incrementally and are rather complicated Boolean functions (see Table 7.2), we chose to favor SAT solvers over BDDs.

8.1.3 Satisfiability Modulo Theories

In addition to SAT problems being a subset of CSP problems the Satisfiability Modulo Theories [72] (SMT) are also a subset. SMT problems can be seen as an extension to SAT problems where some of the Boolean literals are the replaced by predicates, essentially functions, that have a Boolean result. Until we did not see the need for such increments in reasoning in the domain of decision-making. However, when moving to the general modeling scenario SMT solvers might be necessary.

8.1.4 Paraconsistent Logic

We only just recently became aware of paraconsistent logic [73], and its possible application for living with inconsistencies during decision-making. Basically paraconsistent logic is designed to reason with inconsistencies by dealing with contradictions in a discriminating way. Assuming that it could work the biggest disadvantage would be that existing automations for decision-making that are mostly SAT-based would have to be adapted to a different reasoning technique, with our approach of HUMUS we adapted SAT-based reasoning to be able to live with inconsistencies without affecting existing automations.

8.2 Decision-support Systems

Decision support systems [74] assist users during decision making – much like our approaches aim at assisting the user. The difference is that decision support systems are domain-specific and optimized to solve specific problems. The purpose of such systems is to improve the quality of decisions by providing essential pieces of information that help users to make those decisions. A more general discussion can be found at [58]. In other words they try to improve the quality whereas we focus on reducing the quantity or rather living with inconsistencies. Our approach does not contradict the approach of decision support systems and in fact should they be combined. While our approach reduces the number of questions a user may have to answer, a decision support system would then help the user answer these questions more easily. It is conceivable that a decision support system could be beneficial in improving our approach – i. e., by also considering the difficulty of answering questions. We have not explored this yet.

8.3 Operational Research

The interdisciplinary field of Operations research shares its goals with ours of finding optimal sequences [75]. Different algorithms and methods are used to arrive at optimal or near optimal solutions to complex problems. Operations research is used in many different domains and uses special algorithms, heuristics models and so on for many different purposes. Nevertheless to our knowledge no solutions to the specific problem of ordering questions exist.

8.4 Dialog Design

There also exists a great body of work on dialog design, even work on minimizing the length of dialogs [76]. The difference being that different unrelated questions can lead to the dialog goal and that the probabilities of users being able to answer them are assumed to be known. In this case the optimization is about selecting the smallest set of questions with the highest probability of success that need to be answered.

8.5 Expert Systems

Expert Systems [77] are used to represent experts in a certain area. The basic expert system consists of a knowledge base, normally for one problem domain, and an inference engine. The user provides facts and the expert system provides its expertise based on the given facts and rules contained in the knowledge base. On the surface there are similarities to determining an optimal sequence, the creator provides us with facts (in our case questions and their relations) and we provide the expertise, what decision to make next to minimize the overall needed user input. However reproducing our reasoning, for determining the sequence, with rules is not feasible because of the high amount of rules that would be needed, also inconsistencies would mess with the firing of rules.

8.6 Bayesian networks

Bayesian networks are used to represent uncertain knowledge in a natural way. With their help questions about the probabilities of variable assignments in a current environment state can be answered [22]. The reason why using Bayesian networks seem like overkill for our problem is because there are no uncertainties in our scenarios. Every decision has an exact known effect. Though it would be interesting to approximate the probability of a choice, it is not a trivial task due to computational scalability problems when creating such networks [22].

Chapter 9

Conclusions and Future Work

"Never put off till tomorrow what you can do the day after tomorrow."

– *Mark Twain*

In this final chapter we will summarize our work in Section 9.1, recap the contributions in Section 9.2, discuss the threats to validity of our approaches in detail in Section 9.3, and finally give an outlook on future work in Section 9.4.

9.1 Summary

In this thesis we presented two distinct approaches for managing and dealing with inconsistencies during decision-making. One that helps preventing inconsistencies by giving decision makers the maximum amount of freedom possible, while still providing meaningful guidance to reach one's goals faster. The other approach using different reasoning strategies for living with inconsistencies during decision-making, where one strategy in particular seems very promising by ensuring that reasoning after an inconsistency is encountered remains correct at the cost of completeness.

While the focus of this thesis was primarily on the technologies "behind the curtains", it also presented one possibility of how to use and visualize these technologies in a prototype tool. Several evaluation scenarios provided insight into the effectiveness of our approaches and their respective costs. It was shown that our guidance calculation with respect to minimizing user input is nearly optimal and fast enough to be used interactively. Additionally it was also shown that the cost of living with inconsistencies scales and the level of incompleteness is reasonable and that it can even have a positive impact onto resolving inconsistencies.

9.2 Contributions

In general, this thesis contributed to the research area of decision-making and areas that involve decision-making. The key contributions of this thesis are:

(i) A unique perspective on the problems currently encountered during decision-making combined with a vision and approach, that may inspire other researchers to think about the needs of the most important resource in software engineering namely the software engineer. The most important need being the freedom to make decisions and resolve inconsistencies the way the engineer prefers to. Not the other way around, where the tools need them to make decisions in a certain sequence and need them to fix inconsistencies right away (cf. Chapters 3, 4, and 5).

(ii) An approach that allows correct reasoning with SAT-Solvers in the presence of inconsistencies at the expense of marginally more incomplete reasoning, thus also allowing any existing automations to work correctly in the presence of inconsistencies without adaptions needed (cf. Chapter 5).

(iii) An approach on how to determine the shortest path through a series of related questions to any possible configuration, without knowing in advance which configuration it is going to be (cf. Chapter 5).

(iv) A basis for further research in general modeling scenarios (cf. Chapter 7). We provided an assessment on the feasibility, based on preliminary results, of applying these techniques to general modeling scenarios in Section 7.5.

(v) An extensive evaluation of the different aspects and techniques used in our approach (cf. Chapter 7). The results of the evaluation are valuable for researchers and practitioners alike, as they show how effective these techniques are and also provide insights in what is possible.

9.3 Threats to Validity

Similar to the approach and evaluation chapter, this section will be separated into two parts, each dealing with the threats to validity of an independent approach and its respective evaluation.

9.3.1 Guidance Calculation

Threats to construct validity imply whether we are optimizing what we want to optimize. The question is whether reducing the number of questions really reduces effort. Or, do we answer the easy questions only, leaving the hard questions for the decision maker? At the moment our approach tries to automatically answer as many questions as possible. Our premise is that any automation is good. Even if our approach were only to answer the easy, less effort questions, it still benefits the decision maker by not having to answer questions that can be inferred automatically. It is also our opinion that each decision maker feels differently about how challenging a question is. Thus, the effort saved depends on the decision maker.

The threat to internal validity is that we do not know the distribution of configurations. We assume configurations to be equally likely, but we know that this is not true. Still, we also know that data on the likelihood of configurations is not readily available. Foremost, it takes a large number of configurations to elicit information on statistical likelihood. With larger models (questions and relations), it is thus less likely that such data is available. Moreover, even if the data exists, a feedback loop to integrate it with the model does not necessarily exist. This could be another direction for future work.

Regarding external validity the question is: Are the case studies used for validating our approach representative? Due to the fact that the case studies are from very different domains and include real world examples, we can assume them to be representative. Moreover, we calibrated our approach only on one of the used case studies and tested it on all six of them. This implies that our approach was not biased by the calibration and no overfitting occurred.

Concerning conclusion validity we relied on extensive simulations to show that our approach is significantly better than a random approach.

9.3.2 Living with Inconsistencies

Threats to construct validity imply whether we are evaluating the different isolation strategies with the proper criteria. This thesis evaluated the different trade-offs of common SAT-based strategies to tolerating inconsistencies in the domain of decision-making based on qualitative criteria (incomplete, incorrect reasoning, and revisitation) and other criteria (performance). While we acknowledge that these criteria may not be all there are, they seemed reasonable enough for our needs.

We made no assumption that would invalidate the internal validity of our findings. As was discussed, our approach applies to guided decision-making and was evaluated on pre-definable decision models only. Fortunately, many decision models fall into this category and future work will show whether these strategies are applicable to tolerating inconsistencies in general.

Regarding external validity the question is: Are the case studies used for validating our approach representative of decision models in practice? Due to the fact that the case studies are from very different domains and real world examples, we can assume them to be representative with regards to composition and observable behavior. It seems reasonable that in other models relations will be as least as complex as in our case studies. We exhaustively evaluated the five case study models by randomly injecting defects and observing the progression with regard to the different isolation approaches. Due to this exhaustive evaluation and the conclusions, we believe the conclusion validity to be high.

However, we cannot generalize that our result will apply to other domains where SAT-based reasoning is used. This thesis thus provides a proof of concept in that we found domains where tolerating inconsistencies is indeed a viable option. We believe that many other domains would likewise benefit from observations we made here, though perhaps not all.

9.4 Future Work

Based on the threats to validity and other shortcomings mentioned throughout the thesis, we see many opportunities to improve our work further. For the guidance calculation part interesting areas of research would be to do usability tests with our configurator tool to see, how well the mix between the freedom to answer any question and the suggested questions being bigger and changing after each answer is received in comparison to state-of-the-practice configurator. It would also be interesting to get actual configuration data to draw conclusions about the actual distribution of configurations in one decision model and to see how much this data helps to further improve the optimality of our approach or if it even provides any significant improvements. As for the HUMUS isolation strategy there are a lot of research opportunities. First of all its use for product line analysis can be explored, also its potential and feasibility for fixing multiple inconsistencies while tolerating them. Besides improving the calculations behind the approaches to scale even better, the greatest challenge will be to apply our approaches to other domains and explore if and to which degree our approaches are useful in more general modeling scenarios.

Bibliography

[1] Y. Singh and M. Sood, "Model Driven Architecture: A Perspective," in *Advance Computing Conference, 2009. IACC 2009. IEEE International*, march 2009, pp. 1644 –1652. 1

[2] D. Schmidt, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25 – 31, feb. 2006. 1

[3] R. Balzer, "Tolerating Inconsistency," in *13th ICSE, Austin, Texas, USA*, 1991, pp. 158–165. 2, 10

[4] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein, "xlinkit: A Consistency Checking and Smart Link Generation Service," *ACM Trans. Internet Techn.*, vol. 2, no. 2, pp. 151–185, 2002. 2, 10

[5] A. Egyed, "Instant consistency checking for the UML," in *28th ICSE, Shanghai, China*, 2006, pp. 381–390. 2, 10, 76, 77

[6] C. W. Johnson and C. Runciman, "Semantic Errors - Diagnosis and Repair," in *SIGPLAN Symposium on Compiler Construction*, 1982, pp. 88–97. 2

[7] S. Alhir, *UML in a Nutshell: A Desktop Quick Reference*, ser. In a Nutshell. O'Reilly, 1998. 3

[8] M. Davis, G. Logemann, and D. W. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, 1962. 4, 10, 11

[9] A. Nöhrer and A. Egyed, "Conflict Resolution Strategies during Product Configuration," in *Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria*, ser. ICB Research Report, vol. 37. Universität Duisburg-Essen, 2010, pp. 107–114. 5

[10] A. Nöhrer and A. Egyed, "Utilizing the Relationships Between Inconsistencies for more Effective Inconsistency Resolution," in *3rd Workshop on Living with Inconsistencies in Software Development, Colocated with ASE, Antwerp, Belgium.* CEUR Workshop Proceedings Vol-661, 2010, pp. 39–43. 5, 6, 10

[11] A. Nöhrer and A. Egyed, “C2O: A Tool for Guided Decision-making,” in *25th International Conference on Automated Software Engineering, Antwerp, Belgium*, 2010, pp. 363–364. 6, 12

[12] A. Nöhrer, A. Reder, and A. Egyed, “Positive Effects of Utilizing Relationships between Inconsistencies for more Effective Inconsistency Resolution: NIER Track,” in *ICSE*, R. N. Taylor, H. Gall, and N. Medvidovic, Eds. ACM, 2011, pp. 864–867. 6

[13] A. Nöhrer and A. Egyed, “Optimizing User Guidance during Decision-Making,” in *Software Product Lines, 15th International Conference, Munich, Germany*, 2011. 6, 16

[14] A. Nöhrer, A. Biere, and A. Egyed, “Managing SAT inconsistencies with HUMUS,” in *VaMoS*, U. W. Eisenecker, S. Apel, and S. Gnesi, Eds. ACM, 2012, pp. 83–91. 6

[15] A. Nöhrer, A. Egyed, and A. Biere, “A Comparison of Strategies for Tolerating Inconsistencies during Decision-Making,” in *Software Product Lines, 16th International Conference, Salvador, Brazil*, 2012. 6

[16] Y. Xiong, Z. Hu, H. Zhao, H. Song, M. Takeichi, and H. Mei, “Supporting Automatic Model Inconsistency Fixing,” in *7th ESEC/FSE, Amsterdam, The Netherlands*, 2009, pp. 315–324. 10

[17] C. Nentwich, W. Emmerich, and A. Finkelstein, “Consistency Management with Repair Actions,” in *25th ICSE, Portland, Oregon, USA*, 2003, pp. 455–464. 10, 78

[18] A. Egyed, E. Letier, and A. Finkelstein, “Generating and Evaluating Choices for Fixing Inconsistencies in UML Design Models,” in *23rd ASE, L’Aquila, Italy*, 2008, pp. 99–108. 10

[19] A. Egyed, “Fixing Inconsistencies in UML Design Models,” in *29th ICSE, Minneapolis, USA*, 2007, pp. 292–301. 10

[20] L. C. Briand, Y. Labiche, and L. O’Sullivan, “Impact Analysis and Change Management of UML Models,” in *19th International Conference on Software Maintenance, Amsterdam, The Netherlands*, 2003, pp. 256–265. 10

[21] M. Sabetzadeh, S. Nejati, S. M. Easterbrook, and M. Chechik, “Global consistency checking of distributed models with TReMer+,” in *30th ICSE, Leipzig, Germany*, 2008, pp. 815–818. 10

[22] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2009. 10, 81, 82, 84

[23] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, 2009. 11, 49

[24] D. L. Parnas, "On the Design and Development of Program Families," *IEEE Trans. Software Eng.*, vol. 2, no. 1, pp. 1–9, 1976. 12

[25] E. W. Dijkstra, "Notes on Structured Programming," Technological University Eindhoven, Tech. Rep. 70-WSK-03, Second Edition, 1970. 12

[26] P. Clements and L. Northrop, *Software product lines: practices and patterns.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. 12

[27] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990. 12, 13

[28] G. Campbell, N. Burkhard, J. Facemire, and J. O'Connor, "Synthesis Guidebook," Software Productivity Consortium, Hemdon, VA, Tech. Rep. SPC-91122-MC, 1991. 12, 14

[29] F. van der Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action – The Best Industrial Practice in Product Line Engineering.* Springer, 2007. 12

[30] R. Rabiser, "A User-Centered Approach to Product Configuration in Software Product Line Engineering," Ph.D. dissertation, Institute for Systems Engineering and Automation, Christian Doppler Laboratory for Automated Software Engineering, Johannes Kepler University, Linz, February 2009. 12, 14, 15

[31] T. Asikainen, T. Männistö, and T. Soininen, "Using a Configurator for Modelling and Configuring Software Product Lines based on Feature Models, Boston, Massachusetts, USA," in *Workshop on Software Variability Management for Product Derivation in conjunction with SPLC*, 2004. 12

[32] D. Dhungana, P. Grünbacher, and R. Rabiser, "The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study," *Automated Software Engineering*, vol. 18, pp. 77–114, 2011. 12, 14, 15, 62

[33] P. Trinidad, D. Benavides, A. R. Cortés, S. Segura, and A. Jimenez, "FAMA Framework," in *Software Product Lines, 12th International Conference, Limerick, Ireland*, 2008, p. 359. 12, 16

[34] M. Mendonca, M. Branco, and D. Cowan, "S.P.L.O.T.: software product lines online tools," in *24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications.* New York, NY, USA: ACM, 2009, pp. 761–762. 12, 16

[35] G. Botterweck, M. Janota, and D. Schneeweiss, "A Design of a Configurable Feature Model Configurator," in *VaMoS*, ser. ICB Research Report, D. Benavides, A. Metzger, and U. W. Eisenecker, Eds., vol. 29. Universität Duisburg-Essen, 2009, pp. 165–168. 12, 56

[36] S. Deelstra, M. Sinnema, and J. Bosch, "Product Derivation in Software Product Families: A Case Study," *Journal of Systems and Software*, vol. 74, no. 2, pp. 173–194, 2005. 12

[37] L. Chen, M. A. Babar, and N. Ali, "Variability Management in Software Product Lines: A Systematic Review," in *SPLC*, ser. ACM International Conference Proceeding Series, D. Muthig and J. D. McGregor, Eds., vol. 446. ACM, 2009, pp. 81–90. 12

[38] M. Sinnema, J. S. van der Ven, and S. Deelstra, "Using Variability Modeling Principles to Capture Architectural Knowledge," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 5, 2006. 12

[39] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski, "Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches," in *VaMoS*, U. W. Eisenecker, S. Apel, and S. Gnesi, Eds. ACM, 2012, pp. 173–182. 12

[40] K. Czarnecki and U. W. Eisenecker, *Generative Programming – Methods, Tools and Applications*. Addison-Wesley, 2000. 13

[41] D. Benavides, P. T. Martín-Arroyo, and A. R. Cortés, "Automated Reasoning on Feature Models," in *Advanced Information Systems Engineering, 17th International Conference, CAiSE, Porto, Portugal*, 2005, pp. 491–503. 13, 82

[42] M. Mendonça, A. Wasowski, and K. Czarnecki, "SAT-based analysis of feature models is easy," in *Software Product Lines, 13th International Conference, San Francisco, California, USA*, 2009, pp. 231–240. 13, 16

[43] D. Benavides, S. Segura, and A. R. Cortés, "Automated analysis of feature models 20 years later: A literature review," *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010. 13, 14, 16

[44] K. Schmid and I. John, "A Customizable Approach to Full Lifecycle Variability Management," *Journal of the Science of Computer Programming, Special Issue on Variability Management*, vol. 53, no. 3, pp. 259–284, 2004. 14

[45] D. Dhungana, R. Rabiser, P. Grünbacher, K. Lehner, and C. Federspiel, "DOPLER: An Adaptable Tool Suite for Product Line Engineering," in *Software Product Lines, 11th International Conference, Kyoto, Japan*, 2007, pp. 151–152. 15, 32

[46] D. Dhungana and P. Grünbacher, "Understanding Decision-Oriented Variability Modelling," in *SPLC (2)*, S. Thiel and K. Pohl, Eds. Lero Int. Science Centre, University of Limerick, Ireland, 2008, pp. 233–242. 15

[47] R. Dhungana, “A Model-driven Approach to Flexible and Adaptable Software Variability Management,” Ph.D. dissertation, Institute for Systems Engineering and Automation, Christian Doppler Laboratory for Automated Software Engineering, Johannes Kepler University, Linz, February 2009. 15

[48] N. Siegmund, M. Rosenmüller, C. Kästner, P. G. Giarrusso, S. Apel, and S. S. Kolesnikov, “Scalable Prediction of Non-functional Properties in Software Product Lines,” in *SPLC*, E. S. de Almeida, T. Kishi, C. Schwanninger, I. John, and K. Schmid, Eds. IEEE, 2011, pp. 160–169. 16

[49] M. L. Rosa, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede, “Questionnaire-based variability modeling for system configuration,” *Software and System Modeling*, vol. 8, no. 2, pp. 251–274, 2009. 16

[50] G. S. Tseitin, “On the Complexity of Derivation in Propositional Calculus,” in *Automation of Reasoning 2: Classical Papers on Computational Logic 1967–1970*, J. Siekmann and G. Wrightson, Eds. Springer, 1983, pp. 466–483. 16

[51] J. White, D. C. Schmidt, D. Benavides, P. Trinidad, and A. R. Cortés, “Automated Diagnosis of Product-Line Configuration Errors in Feature Models,” in *Software Product Lines, 12th International Conference, Limerick, Ireland*, 2008, pp. 225–234. 16, 28

[52] D. Haw, C. A. Goble, and A. L. Rector, “GUIDANCE: Making it Easy for the USer to be an Expert,” in *IDS*, 1994, pp. 25–48. 16

[53] R. L. Cobleigh, G. S. Avrunin, and L. A. Clarke, “User guidance for creating precise and accessible property specifications,” in *14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Portland, Oregon, USA*, 2006, pp. 208–218. 16

[54] C. van Nimwegen, D. D. Burgos, H. van Oostendorp, and H. Schijf, “The paradox of the assisted user: guidance can be counterproductive,” in *Conference on Human Factors in Computing Systems, Montréal, Québec, Canada*, 2006, pp. 917–926. 16, 33, 38

[55] K. J. Vicente, “Crazy Clocks: Counterintuitive Consequences of "Intelligent" Automation,” *IEEE Intelligent Systems*, vol. 16, no. 6, pp. 74–76, 2001. 16, 32

[56] G. A. Miller, “The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information,” *The Psychological Review*, vol. 63, pp. 81–97, 1956. 16

[57] A. Pleuss, R. Rabiser, and G. Botterweck, “Visualization Techniques for Application in Interactive Product Configuration,” in *SPLC Workshops*, I. Schaefer, I. John, and K. Schmid, Eds. ACM, 2011, p. 22. 16

[58] G. Marakas, *Decision Support Systems in the Twenty-First Century.* Prentice Hall, 1999. 25, 83

[59] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, "Intelligent Support for Interactive Configuration of Mass-Customized Products," in *Engineering of Intelligent Systems, 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2001, Budapest, Hungary*, 2001, pp. 746–756. 28

[60] A. MacLean, R. M. Young, and T. P. Moran, "Design rationale: the argument behind the artifact," in *SIGCHI conference on Human factors in computing systems: Wings for the mind*, ser. CHI '89. New York, NY, USA: ACM, 1989, pp. 247–252. 32

[61] A. Biere, "PicoSAT Essentials," *JSAT*, vol. 4, no. 2-4, pp. 75–97, 2008. 36

[62] D. L. Berre and A. Parrain, "The Sat4j library, release 2.2," *JSAT*, vol. 7, no. 2-3, pp. 59–6, 2010. 36

[63] M. H. Liffiton and K. A. Sakallah, "Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints," *J. Autom. Reason.*, vol. 40, no. 1, pp. 1–33, 2008. 49, 51

[64] C. M. Li and F. Manyà, "MaxSAT, Hard and Soft Constraints," in *Handbook of Satisfiability*, 2009, pp. 613–631. 49, 50

[65] P. Grünbacher, R. Rabiser, and D. Dhungana, "Product Line Tools are Product Lines Too: Lessons Learned from Developing a Tool Suite," in *23rd IEEE/ACM International Conference on Automated Software Engineering, L'Aquila, Italy*, 2008, pp. 351–354. 62

[66] A. Egyed, "Automatically Detecting and Tracking Inconsistencies in Software Design Models," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, 2010. 76

[67] M. A. Salido, "A non-binary constraint ordering heuristic for constraint satisfaction problems," *Applied Mathematics and Computation*, vol. 198, no. 1, pp. 280–295, 2008. 82

[68] J. Amilhastre, H. Fargier, and P. Marquis, "Consistency restoration and explanations in dynamic CSPs – Application to configuration," *Artificial Intelligence*, vol. 135, no. 1–2, pp. 199–234, 2002. 82

[69] R. Drechsler and B. Becker, *Binary Decision Diagrams: Theory and Implementation.* Kluwer Academic Publishers, 1998. 82

[70] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. 35, pp. 677–691, 1986. 82

[71] S. Reda, R. Drechsler, and A. Orailoglu, "On the relation between SAT and BDDs for equivalence checking," in *Quality Electronic Design, 2002. Proceedings. International Symposium on*, 2002, pp. 394–399. 82

[72] C. Barrett, L. de Moura, and A. Stump, "SMT-COMP: Satisfiability Modulo Theories Competition," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, K. Etessami and S. Rajamani, Eds. Springer Berlin / Heidelberg, 2005, vol. 3576, pp. 503–516. 82

[73] A. Hunter, "Paraconsistent Logics," in *Handbook of Defeasible Reasoning and Uncertain Information.* Kluwer, 1996, pp. 11–36. 82

[74] J. Wilkenfeld, S. Kraus, K. M. Holley, and M. A. Harris, "GENIE: A decision support system for crisis negotiations," *Decis. Support Syst.*, vol. 14, pp. 369–391, August 1995. 83

[75] H. A. Taha, *Operations Research: An Introduction.* Pearson/Prentice Hall, 2006. 83

[76] R. B. Inouye, "Minimizing the length of non-mixed initiative dialogs," in *ACL workshop on Student research, Barcelona, Spain*, 2004, pp. 39–43. 83

[77] R. G. D. Giarratano Joseph C., *Expert Systems: Principles and Programming (Fourth Edition)*, 3rd ed. Course Technology, 2004. 84

[78] U. W. Eisenecker, S. Apel, and S. Gnesi, Eds., *Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25-27, 2012. Proceedings.* ACM, 2012.

Glossary

BDD	Binary Decision Diagrams
CNF	Conjunctive Normal Form
CSP	Constraint Satisfaction Problem
DOPLER	Decision-Oriented Product Line Engineering for effective Reuse
HUMUS	High-level Union of Minimal Unsatisfiable Sets
MCS	Minimal Correcting Set
MDA	Model Driven Architecture
MDD	Model Driven Development
MDE	Model Driven Engineering
MSS	Maximum Satisfiable Set
MUS	Minimal Unsatisfiable Set
SAT Problem	Boolean Satisfiability Problem
SMT	Satisfiability Modulo Theories
SPLE	Software product line engineering
UML	Unified Modeling Language (http://www.uml.org)

Appendix A

C2O – Configurator 2.0 Manual

A.1 Description

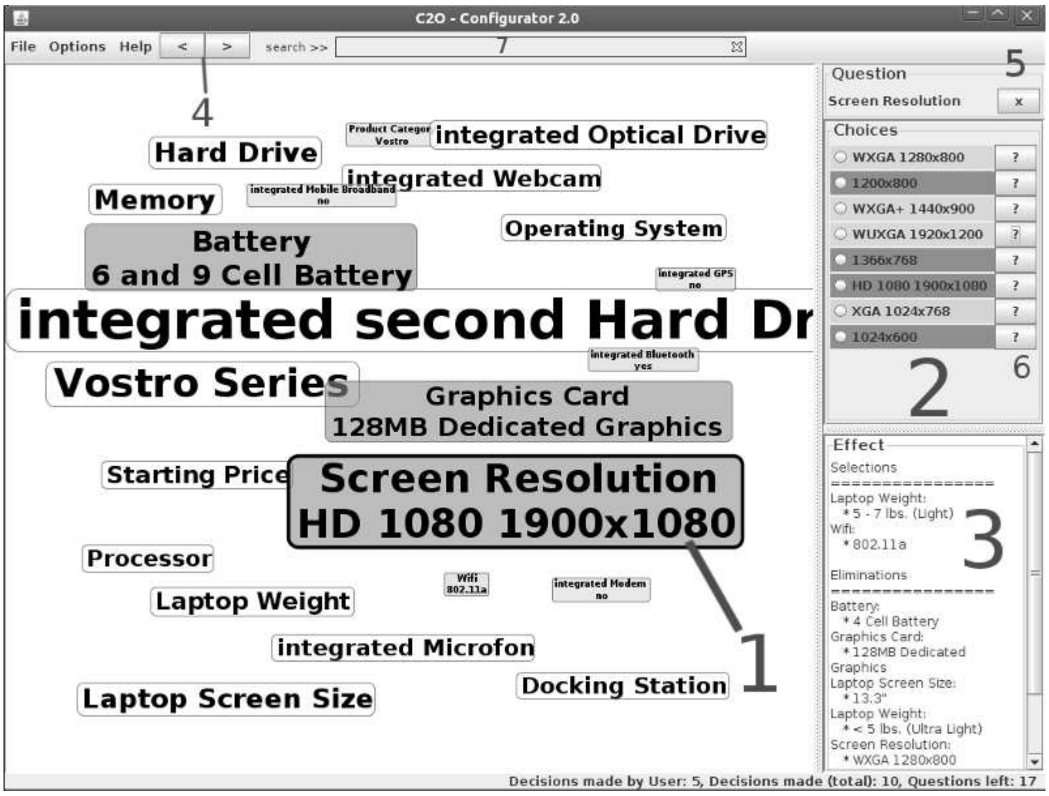
Decision models are widely used in software engineering to describe and restrict decision-making (e. g., deriving a product from a product-line). Since decisions are typically interdependent, conflicts during decision-making are inevitably reached when invalid combinations of decisions are made. Unfortunately, the current state-of-the-art provides little support for dealing with such conflicts. On the one hand, some conflicts can be avoided by providing more freedom in which sequence decisions are made (i. e., most important decisions first). On the other hand, conflicts are unavoidable at times and living with conflicts may be preferable over forcing the user to fix them right away – particularly, because fixing conflicts becomes easier the more is known about an user's intentions. The C2O (Configurator 2.0) tool is one example how guided decision-making could look like. The tool allows the user to answer questions in an arbitrary sequence – with and without the presence of conflicts. While giving users those freedoms, it still supports and guides them by *i)* rearranging the sequence of questions according to their potential to minimize user input, *ii)* providing guidance to avoid follow-on conflicts, and *iii)* supporting users in fixing conflicts at a later time.

A.2 Third Party Libraries

More information on used third party libraries can be found at the following addresses

- prefuse http://www.prefuse.org/
- PicoSAT http://fmv.jku.at/picosat
- SAT4J http://www.sat4j.org

A.3 GUI Explanation



Selections
================
Laptop Weight:
* 5 - 7 lbs. (Light)
Wifi:
* 802.11a

Eliminations
================
Battery:
* 4 Cell Battery
Graphics Card:
* 128MB Dedicated Graphics
Laptop Screen Size:
* 13.3"
Laptop Weight:
* < 5 lbs. (Ultra Light)
Screen Resolution:
* WXGA 1280x800

Decisions made by User: 5, Decisions made (total): 10, Questions left: 17

Figure A.1: Overview of the C2O Configurator GUI.

A.3.1 Questions (1)

Significance of Font Size: The bigger the font size, the higher is the potential (gain) of this question at this moment to lead to the desired configuration via the shortest path possible, if answered next. This potential is recalculated after every decision made by the user and adapts automatically.

Green (Black): Decision made by the user

Yellow (Light Grey): Decisions automatically made by the reasoning engine derived from user decisions

Red (Dark Grey): Decisions which are in conflict with at least one other decision

Orange (Grey): Highlight color for questions that match the search pattern

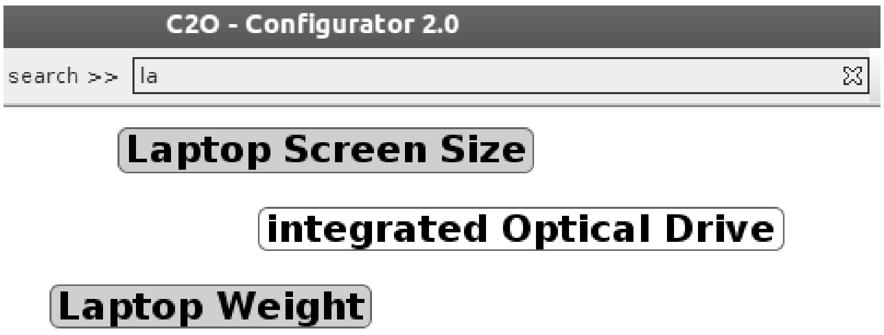


Figure A.2: Searching for questions.

A.3.2 Choice list (2)

Green (White): Choices which can be selected at this time without introducing any conflicts (or is already selected)

Red (Dark Grey): Choices which will result in a new conflict

Yellow (Light Grey): Choices which will reduce the number of possibilities of how the current conflicts can be resolved. If the number of possibilities is reduced to one a conflict can be resolved automatically, and it will be dealt with according to the settings (Option → Automatically Resolve Conflict)

A.3.3 Buttons

Explanation box (3): If you click the question mark (6) right next to a choice, you get info about the effect if you select this choice as the answer to the question

Back - Forward (4): Let's you move forwards and backwards in the history of your decisions

Undo answer (5): The currently selected choice for this question is undone

Get explanation (6): Displays which effects a selection of this choice would have.

Search (7): You can search for a specific question which will be highlighted

A.3.4 Menu - File

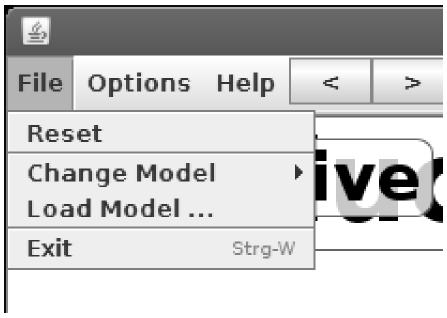


Figure A.3: File menu.

Reset: Resets the reasoning engine, and let's you start over

Change Model: List of all available models you can change too

Load Model: Loads a model in the above specified XML format

A.3.5 Menu - Options

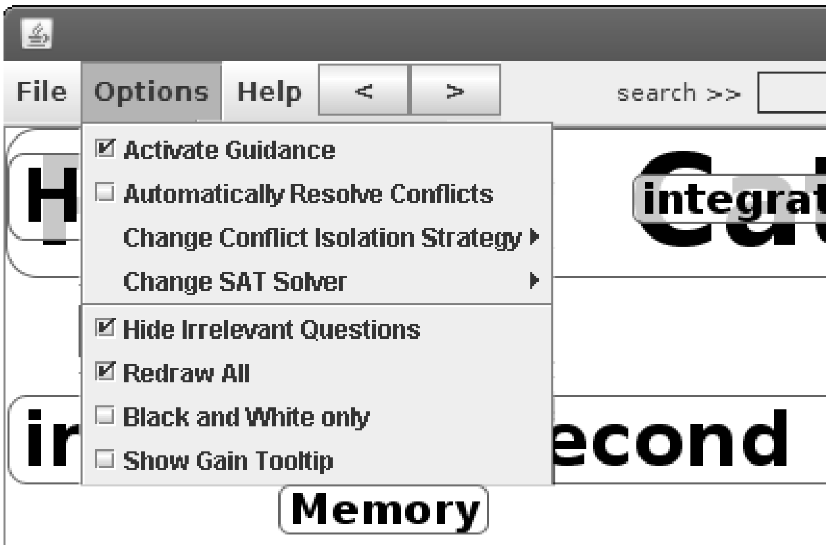


Figure A.4: Options menu.

Reasoning

Activate Guidance: Gains will be calculated. The question font size adapts according to the calculated potential for leading to the shortest path if answered next

Automatically Resolve Conflict: If activated the SAT Solver automatically solves conflicts if possible. If not activated you will be asked if you want to keep your original choice or take the proposed choice from the SAT Solver

Change Conflict Isolation Strategy: Adds the possibility to change the Isolation strategy during the Isolation of a Conflict. The choices are HUMUS, MaxSAT and Skip in- or excluding implicit trust (trust the last user decision that caused the conflict). HUMUS basically isolates all contributors and MaxSAT isolates as few as possible (random), whereas Skip explicitly does not trust the decision that caused the conflict and skips it.

Change SAT Solver: Adds the possibility to change the SAT solver from PicoSAT to SAT4j and vice versa, PicoSAT is faster (especially with the HUMUS calculation) but needs native libraries (which are provided with the download)

Appearance

Hide Irrelevant Questions: Hides (at this time) irrelevant questions, for a better overview.

Redraw all: Redraws the complete GUI after each change, instead of only changed parts which is a little bit slower. This is useful because sometimes the size changes cause artifacts.

Show Gain Tooltip: Adds a mouse-over-effect to each question which displays its current gain value - Black & White: For presentation or printing purposes the GUI can be switched to gray scale.

A.4 FAQ

A.4.1 Is the source code publicly available?

For the time being the source code is not publicly available. If you like the tool and want to use its technologies please contact us directly, so we can work something out.

A.4.2 Can I use other Data Types than String as question answers, or have multiple choice answers?

The reasoning engine can handle different data types and multiple choices answers, but this prototype cannot visualize and handle them at this time, also no support for loading such models is provided at the moment.

A.4.3 Does this technology work with other type of models?

- The conflict detection, explanation, and living with conflicts approach is built on SAT-solving technology, as a consequence it works with all models that can be translated into CNF (conjunctive normal form) and used by a SAT-Solver.

- The guidance heuristics is also implemented for models translatable into CNF. But the concepts behind it are very general so it should be easy to adapt to other models as well.

Appendix B

Model XML Format

This specifies the preliminary model XML format, used by the C2O Configurator at the moment not many validations are performed so make sure you get it right.

B.1 Overview

```
<?xml version="1.0" encoding="UTF-8" ?>
<model name="Example">
    <questions>
    ...
    </questions>
    <relations>
    ...
    </relations>
</model>
```

Each model has to have a `<model>` element as root where the name of the model is specified via the "name" attribute. Since some caching is performed and saved on the hard-drive make sure this name is unique, otherwise switching between models with the same name might cause strange side effects. This `<model>` element has two child elements `<questions>` and `<relations>` which are explained next. Questions

```
<question identifier="Question 1"
    description="Which Question do you want to answer?">
    <choice name="Question 2" />
    <choice name="Question 3" />
</question>
```

The `<questions>` element can have any number of child elements. Each must have a unique "identifier" attribute, and can have have a description containing the whole question. Each must have at least one element as a child. The `<choice>` element represents one possibility of how to answer the question and again must have a "name" attribute that has to be unique in the scope of this `<question>` element.

B.2 Relations

B.2.1 Constraint Relations

Constraint Relations in always must have one source question and at least one target question. Each `<source>` and `<target>` element must have a set attribute "question-Identifier" that has to correspond to the unique identifier of a defined question.

```
<constraintRelation>
    <source questionIdentifier="Question 2" />
    <targets>
        <target questionIdentifier="Question 4" />
    </targets>
    <rule choiceName="Choice 1">
        <allowed choiceName="Choice 1" />
    </rule>
    <rule choiceName="Choice 2">
        <disallowed choiceName="Choice 1" />
    </rule>
</constraintRelation>
```

A Constraint Relation defines a relation where selecting choices for the source question constrain the possibilities for selecting choices of the target question. If you have more than one target question keep in mind that all the specified target questions must have the same choices.

Besides the source and targets a constrained relation can have rules. At most one for each choice of the source question. Each element has to have a "choiceName" attribute identifying a choice of the source question. This element can have either or child elements but not both, again with a "choiceName" attribute identifying a choice of the target question.

The semantics of the two rules given in the example are:

- If "Choice 1" of "Question 2" is selected only "Choice 1" is allowed as answer for "Question 4". The same is true for the other direction, meaning if "Choice 1" of "Question 4" is selected only "Choice 1" is allowed as answer for "Question 2"

- If "Choice 2" of "Question 2" is selected, every choice but "Choice 1" is allowed as answer for "Question 4", again if "Choice 1" of "Question 4" is selected any choice but "Choice 2" is allowed as answer for "Question 2". (this statement would be even further constrained by the first rule)

The semantics of choices where no rule is explicitly defined is that a combination with any choice of the target is possible.

B.2.2 Relevancy Relation

Relevancy Relations in always must have one source question and at least one target question. Each `<source>` and `<target>` element must have a set attribute "question-Identifier" that has to correspond to the unique identifier of a defined question.

```
<relevancyRelation>
    <source questionIdentifier="Question 1" />
    <targets>
        <target questionIdentifier="Question 2" />
    </targets>
    <irrelevantIf choiceName="Question 3" />
</relevancyRelation>
```

A Relevancy Relation defines a relation where selecting choices for the source question renders other questions relevant or irrelevant (determines if they need to be answered by the user). This relation can also be used to impose a specific partial sequence onto the automatic guidance calculation.

Besides the source and targets a relevancy relation can have rules. Those rules are represented by either a or element, at most one for each choice of the source question. Again those elements should not be mixed. Each of those elements has to have a "choiceName" attribute identifying a choice of the source question.

The semantics are:

- If no `<relevantIf>` or `<irrelevantIf>` element is defined, every choice of the source question makes the target questions relevant. This can be used just to impose a partial sequence onto the automatic guidance calculation.

- If `<relevantIf>` elements are used each choice identified by these elements will render the target questions relevant, all other choices will render them irrelevant.

- If `<irrelevantIf>` elements are used each choice identified by these elements will render the target questions irrelevant, all other choices will render them relevant.

B.2.3 CNF Relation

A CNF Relation is intended for more complex relations that cannot be expressed with the other two kinds of relations. You basically can write any CNF (conjunctive normal form) clause and add it as a relation. No source or targets are required for this type of relation.

```
<cnfRelation>
    <literal question="Question 3" choiceName="Choice 1"
        positive="true" />
    <literal question="Question 3" choiceName="Choice 2" />
    <literal question="Question 5" choiceName="Choice 3"
        positive="false" />
</cnfRelation>
```

A CNF Relation consists of elements, where each element represents one literal in a CNF clause, the attribute "question" and "choiceName" specify the answer that is internally represented by a literal, the attribute "positive" indicates whether the literal should be added as is ($literal$, positive="true" which can be omitted) or negated ($\neg literal$, positive="false").

The semantics of this example are:
$Question3(Choice1) \vee Question3(Choice2) \vee \neg Question5(Choice3)$

B.3 Examples

B.3.1 Example.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<model name="Example_XML">
    <questions>
        <question identifier="Question 1" description="Which
           Question do you want to answer?" >
            <choice name="Question 2" />
            <choice name="Question 3" />
        </question>
        <question identifier="Question 2">
            <choice name="Choice 1" />
            <choice name="Choice 2" />
        </question>
        <question identifier="Question 3">
            <choice name="Choice 1" />
            <choice name="Choice 2" />
            <choice name="Choice 3" />
        </question>
        <question identifier="Question 4">
            <choice name="Choice 1" />
            <choice name="Choice 2" />
            <choice name="Choice 3" />
        </question>
        <question identifier="Question 5">
            <choice name="Choice 3" />
            <choice name="Choice 4" />
        </question>
    </questions>
    <relations>
    <relevancyRelation>
        <source questionIdentifier="Question 1" />
        <targets>
            <target questionIdentifier="Question 2" />
        </targets>
        <irrelevantIf choiceName="Question 3" />
    </relevancyRelation>
        <relevancyRelation>
            <source questionIdentifier="Question 1" />
            <targets>
                <target questionIdentifier="Question 3" />
            </targets>
            <relevantIf choiceName="Question 3" />
        </relevancyRelation>
    <constraintRelation>
        <source questionIdentifier="Question 2" />
        <targets>
            <target questionIdentifier="Question 4" />
        </targets>
        <rule choiceName="Choice 1">
            <allowed choiceName="Choice 1" />
        </rule>
        <rule choiceName="Choice 2">
            <disallowed choiceName="Choice 1" />
        </rule>
    </constraintRelation>
    <cnfRelation>
      <literal question="Question 3" choiceName="Choice 1" positive=
         "true" />
      <literal question="Question 3" choiceName="Choice 2" />
      <literal question="Question 5" choiceName="Choice 3" positive=
         "false" />
    </cnfRelation>
    </relations>
</model>
```

B.3.2 Car.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<model name="CarModel">
    <questions>
        <question identifier="edition" description="edition">
            <choice name="sport" />
            <choice name="none" />
            <choice name="family" />
        </question>
        <question identifier="color" description="color">
            <choice name="red" />
            <choice name="black" />
            <choice name="green" />
            <choice name="blue" />
            <choice name="orange" />
        </question>
        <question identifier="tires" description="tires">
            <choice name="low profile" />
            <choice name="normal" />
            <choice name="fuel-efficient" />
        </question>
        <question identifier="type" description="type">
            <choice name="coupe" />
            <choice name="station wagon" />
            <choice name="cabriolet" />
        </question>
        <question identifier="rims" description="rims">
            <choice name="steel" />
            <choice name="light-alloy" />
        </question>
        <question identifier="horsepower" description="horsepower">
            <choice name="50" />
            <choice name="75" />
            <choice name="100" />
            <choice name="150" />
        </question>
        <question identifier="extras" description="extras">
            <choice name="none" />
            <choice name="roof rack" />
        </question>
    </questions>
    <relations>
        <constraintRelation>
            <source questionIdentifier="type" />
            <targets>
                <target questionIdentifier="horsepower" />
            </targets>
            <rule />
            <rule choiceName="cabriolet">
                <disallowed choiceName="50" />
                <disallowed choiceName="75" />
            </rule>
            <rule choiceName="station wagon">
                <disallowed choiceName="50" />
            </rule>
        </constraintRelation>
        <constraintRelation>
            <source questionIdentifier="extras" />
            <targets>
                <target questionIdentifier="type" />
            </targets>
            <rule choiceName="roof rack">
                <disallowed choiceName="cabriolet" />
                <disallowed choiceName="coupe" />
            </rule>
        </constraintRelation>
        <constraintRelation>
            <source questionIdentifier="rims" />
            <targets>
                <target questionIdentifier="tires" />
            </targets>
            <rule choiceName="steel">
                <disallowed choiceName="low profile" />
            </rule>
            <rule choiceName="light-alloy">
                <disallowed choiceName="normal" />
            </rule>
        </constraintRelation>
        <constraintRelation>
            <source questionIdentifier="type" />
            <targets>
                <target questionIdentifier="rims" />
            </targets>
            <rule />
            <rule choiceName="cabriolet">
                <disallowed choiceName="steel" />
            </rule>
        </constraintRelation>
        <constraintRelation>
            <source questionIdentifier="type" />
            <targets>
                <target questionIdentifier="color" />
            </targets>
            <rule choiceName="cabriolet">
                <disallowed choiceName="blue" />
                <disallowed choiceName="black" />
            </rule>
            <rule choiceName="coupe">
                <disallowed choiceName="orange" />
                <disallowed choiceName="blue" />
            </rule>
            <rule choiceName="station wagon">
                <disallowed choiceName="orange" />
                <disallowed choiceName="red" />
                <disallowed choiceName="green" />
            </rule>
        </constraintRelation>
        <constraintRelation>
            <source questionIdentifier="edition" />
            <targets>
                <target questionIdentifier="type" />
            </targets>
            <rule />
            <rule choiceName="family">
                <disallowed choiceName="cabriolet" />
                <disallowed choiceName="coupe" />
            </rule>
            <rule choiceName="sport">
                <disallowed choiceName="coupe" />
                <disallowed choiceName="station wagon" />
            </rule>
        </constraintRelation>
    </relations>
</model>
```

Alexander Nöhrer

Personal Data

Place and Date of Birth:	Eisenstadt, Austria — June 19, 1982
Religion:	Catholic
Address:	Berggasse 9, 7072 Mörbisch/See
email:	alexander.noehrer@gmail.com

Work Experience

Current Mar 2009	Project Assistant at **Johannes Kepler University Linz** *Institute of Systems Engineering and Automation* `http://www.jku.at/sea/` Researching ways to improve user guidance primarily in decision-making scenarios for my PhD thesis. Teaching ”Presenation and Working techniques” course.
Oct 2008 Jul 2008	Software Engineer at **Austrian Research Center Seibersdorf** *Radio-Frequency Engineering Department* `http://rf.seibersdorf-laboratories.at/` Customer support and adding new hardware drivers for the measurement software ”Field Nose”.
Oct 2007 Jul 2007	Software Engineer at **Austrian Research Center Seibersdorf** *Radio-Frequency Engineering Department* `http://rf.seibersdorf-laboratories.at/` Customer support and minor bugs fixes for the measurement software ”Field Nose”.
Oct 2006 Jul 2005	Software Engineer at **Austrian Research Center Seibersdorf** *Radio-Frequency Engineering Department* `http://rf.seibersdorf-laboratories.at/` Redesigned the measurement logic of the measurement software ”Field Nose” as part of my Master thesis (measurement process, hardware driver interface, data storage).
Jun 2005 Oct 2004	Tutor at **University of Applied Sciences Upper Austria** *Media Technology and Design* `http://www.fh-ooe.at/` Tutor for Java programming course, providing a questions and answers tutorium for students and correcting their exercises.
Oct 2004 Jul 2004	Software Engineer at **Austrian Research Center Seibersdorf** *Radio-Frequency Engineering Department* `http://rf.seibersdorf-laboratories.at/` Improved the measurement software ”Field Nose” including several major bug fixes. Developed a noise reducing algorithm for the measurement results.
Oct 2003 Jul 2003	Internship at **Austrian Research Center Seibersdorf** *Radio-Frequency Engineering Department* `http://rf.seibersdorf-laboratories.at/` Created the GUI of the measurement software ”Field Nose” with Java Swing.

Education

Mar 2012 Oct 2006	PhD in Software Engineering *Johannes Kepler University*, Linz `http://www.jku.at/`
Jul 2006 Oct 2002	Master of Science in Software Engineering (Dipl.-Ing.(FH)) *University of Applied Sciences Upper Austria*, Hagenberg `http://www.fh-ooe.at/` Besides providing a founded education in computer science, extensive theoretical and practical training in Project Engineering (agile and classical), Social Competence (interpersonal skills, presentation techniques, teamwork, rules for successful communication, conflict and stress management), and Business Basics was supplied.
Jun 2000 Oct 1992	A Levels *BRG Eisenstadt* `http://www.gymnasium-eisenstadt.at/`

Technical Skills and Competencies

		Basics — Expert
Operating Systems	Windows (3.11, 95, 98, 2000, XP, Vista, 7)	● ● ● ● ●
	Linux (Gentoo, Debian, SUSE)	● ● ● ●
	MacOS	● ●
Programming Languages	Java	● ● ● ● ●
	C, C++, Delphi, Pascal	● ● ● ●
	C#, VB.net	● ● ●
	VB 6.0, Assembler	●
Scripting Languages	Shell-Scripts, ANT	● ● ●
	JavaScript, ActionScript, PHP, Nullsoft Installer Scripts	● ●
Frameworks	Java Swing, Java Beans, SWT, RMI, JNI	● ● ● ● ●
	Spring, JUnit, C Stdio, C++ STL	● ● ●
	JSP, ASP.net, GWT, Restlet, EJB, MPI	● ●
Concepts	Object-oriented programming	● ● ● ● ●
	Component-oriented programming	● ● ● ● ●
	Procedural programming	● ● ● ● ●
	Introspection, Concurrency, Sockets	● ● ● ●
	Design Patterns, Anti Patterns	● ● ● ●
	Functional programming	● ● ●
Databases	HSQLDB	● ● ● ●
	Oracle9i, mySQL	● ●
Other	SAT Solvers, XML(DTD, XPath, XSLT, XML Schema)	● ● ● ● ●
	SQL, PL/SQL, Product lines, Feature Models	● ● ● ●
	UML, ER-Diagrams, Regular Expressions, Cryptography	● ● ●
Tools	Eclipse, CVS, SVN	● ● ● ●
	LaTeX(MikTeX, LyX, TeXnicCenter), JavaDoc	● ● ●
	Visual Studio, MS Office, Visio, OpenOffice, DoxyGen	● ●

Languages

German:	Mothertongue
English:	Fluent (spoken and written)
Russian:	Basic Knowledge

Interests

Simplifying User Interface Design, Artificial Intelligence, Cryptography, Adapting tools to users' needs by not needing users to adapt to the tools' needs

Publications

[8] A. Nöhrer, A. Egyed, and A. Biere. A Comparison of Strategies for Tolerating Inconsistencies during Decision-Making. In *Software Product Lines, 16th International Conference, Salvador, Brazil*, 2012

[7] A. Nöhrer, A. Biere, and A. Egyed. Managing SAT inconsistencies with HUMUS. In U. W. Eisenecker, S. Apel, and S. Gnesi, editors, *VaMoS*, pages 83–91. ACM, 2012

[6] A. Nöhrer and A. Egyed. Optimizing User Guidance during Decision-Making. In *Software Product Lines, 15th International Conference, Munich, Germany*, 2011

[5] A. Egyed, A. Demuth, A. Ghabi, R. E. Lopez-Herrejon, P. Mäder, A. Nöhrer, and A. Reder. Fine-Tuning Model Transformation: Change Propagation in Context of Consistency, Completeness, and Human Guidance. In *Theory and Practice of Model Transformations - 4th International Conference, Zurich, Switzerland*, pages 1–14, 2011

[4] A. Nöhrer, A. Reder, and A. Egyed. Positive Effects of Utilizing Relationships between Inconsistencies for more Effective Inconsistency Resolution: NIER Track. In R. N. Taylor, H. Gall, and N. Medvidovic, editors, *ICSE*, pages 864–867. ACM, 2011

[3] A. Nöhrer and A. Egyed. C2O: A Tool for Guided Decision-making. In *25th International Conference on Automated Software Engineering, Antwerp, Belgium*, pages 363–364, 2010

[2] A. Nöhrer and A. Egyed. Utilizing the Relationships Between Inconsistencies for more Effective Inconsistency Resolution. In *3rd Workshop on Living with Inconsistencies in Software Development, Colocated with ASE, Antwerp, Belgium*, pages 39–43. CEUR Workshop Proceedings Vol-661, 2010

[1] A. Nöhrer and A. Egyed. Conflict Resolution Strategies during Product Configuration. In *Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria*, volume 37 of *ICB Research Report*, pages 107–114. Universität Duisburg-Essen, 2010