Autom Softw Eng (2013) 20:265–296
DOI 10.1007/s10515-012-0117-4

C2O configurator: a tool for guided decision-making

Alexander Nöhrer · Alexander Egyed

Received: 29 March 2012 / Accepted: 28 December 2012 / Published online: 19 January 2013

Abstract 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, it is often neither obvious which decisions have the most significant impact nor which decisions might ultimately conflict. Unfortunately, the current state-of-the-art provides little support for dealing with such situations. On the one hand, some conflicts can be avoided by providing more freedom in which order 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 as more is known about a user's intentions. This paper introduces the C2O (Configurator 2.0) tool for guided decision-making. The tool allows the user to answer questions in an arbitrary order—with and without the presence of inconsistencies. While giving users those freedoms, it still supports and guides them by (i) rearranging the order 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.

1 Introduction

In software engineering, many tasks exist that involve making decisions (Dhungana et al. 2007; Hayes and Dekhtyar 2005). For example, software installation wizards guide users through a set of predefined questions, each question with predefined choices on how to deploy a software system. Or, configuration wizards in product lines let users select from a set of features and their choices for instantiating a prod-

A. Nöhrer · A. Egyed (✉)
Institute for Systems Engineering and Automation, Johannes Kepler University, Linz, Austria
e-mail: alexander.egyed@jku.at

uct from a product line. Decision-making is also used to handle variations in user and target-platform requirements as it is the case with, for example, the Linux kernel (She et al. 2011), since it has to run on different hardware and users have different requirements onto an operating system. Many decisions have to be made, although many default values exist which is an indication for the sheer amount of decisions required to configure the kernel. The filling in of trace matrices involves also decision-making, where users are required to decide for each requirement and piece of code whether there is a trace or no trace (Hayes and Dekhtyar 2005). Even the process of resolving inconsistencies in design models could be seen as decision-making, where the questions would be inconsistencies, the choices the possible fixing actions to choose from and the relations the side effects of those fixing actions.

This paper introduces the C2O Configurator 2.0 (available for download at the C2O website[1]), a tool for guided decision-making—focusing on the product line domain. From a decision oriented view, a decision model consists of questions and choices for these questions, which are typically interrelated. For example, an online car configurator is such a decision model where the car manufacturer offers a predefined set of car types with predefined features. Relations typically impose or restrict choices. For example, choosing a convertible makes asking about roof racks irrelevant for obvious reasons. Not all relations are that obvious.

During decision-making, decisions may thus lead to inconsistencies if they violate such relations. It is quite easy to detect such inconsistencies [through SAT-Solvers (Davis et al. 1962) or consistency checkers (Egyed 2006)]. It is also quite easy to prevent inconsistencies by disabling choices that would lead to inconsistencies. Doing so is state-of-the-practice, however this is problematic whenever the decision maker reaches a point where a desired choice is no longer available—a dead end. Typically, tool support to avoid and deal with dead ends is rarely existent and when a dead end is reached, decision makers often have no choice but to backtrack and start over. We identified the following main reasons for dead ends:

1. Decision makers are required to answer questions first that may not be important to them. This unnecessarily restricts the answers to follow-on questions.
2. Decision makers are unaware of the consequences of their choices because they do not understand all relations, which can result in favored choices for unanswered questions being excluded.

The first issue stems from the fact that a predefined order for answering questions restricts the user's freedom in answering questions (van Nimwegen et al. 2006). This issue is largely ignored in literature but of essential importance because users' needs vary and users prefer to answer questions first that are important to them or easy to answer—before being led through the remaining questions. Since the user may not know how to answer the remaining questions first, such that favored choices are not eliminated, such a freedom could help avoid possible dead ends significantly. For example, most online car configurators "force" the user to first select the type of car before selecting other properties (e.g., engine, color). However, what if the user cares more about mileage than make? Any pre-wired solution, even if optimal, that

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

deviates from the user's preferred way of answering the questions then forces the user to exhaustively explore all choices to find the ones that satisfy the desired properties. While it would be simple to allow users to answer questions arbitrarily, there are good reasons to restrict the order. A pre-defined order can be optimized to avoid unnecessary questions to be answered. The second issue is a matter of visualization which some tools already address.

Naturally, even by providing the freedom to answer questions arbitrarily, doing so may not prevent dead ends altogether. Existing tool support for decision-making does not allow inconsistencies and as such it forces the user to backtrack decisions to resolve the dead end before continuing. Unfortunately, such backtracking is quite complex and users may prefer to continue making decisions, even accepting the resulting inconsistencies [i.e., tolerating inconsistencies (Balzer 1991)]. The freedom to select any choice during decision-making, even a inconsistent one, is a legitimate way forward without forcing the user to deal with a problem that may not be important from a user's perspective at that time.

This paper introduces a product configurator tool that allows users to answer questions and introduce inconsistencies arbitrarily—but provides the necessary capabilities to support the user during decision-making: Firstly users can make decisions in any arbitrary order. However guidance is still provided: the questions are presented in an optimal (with regard to minimizing user input) order, which the user can choose to ignore at any time. Secondly consequences of decisions can be visualized beforehand if the user wishes. And finally our tool allows the configuration process at a dead end to be continued by entering a inconsistent state. However users will be guided through the remaining questions to avoid follow-on inconsistencies and any additional answers provided will be used later to help resolve the inconsistency.

2 Illustration and definitions

Our tool works with decision models, as mentioned in the introduction decision models consist of questions, choices of how to answer these questions, and relations between questions and/or choices. The example we will be using through the remainder of this paper is depicted in Fig. 1, it shows a conceptual decision model of how a modeling tool could be configured out of several software components (much like as if you were to combine several eclipse plug-ins to one tool). The model depicted in Fig. 1 has seven questions (Modeling Tool, Consistency, GUI, ...) with choices (yes, no, CSP, ...) and also outlines relations (dashed arrows), the nature of these relations will be discussed after the introduction to our formalism. While overall the model is quite coarse-grained, we also included one decision about the version of the *Logger* component to show that there are no restrictions on granularity (e.g. the actual version). This example 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 our tool.

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:

$$DecisionModel = (Questions, Relations)$$

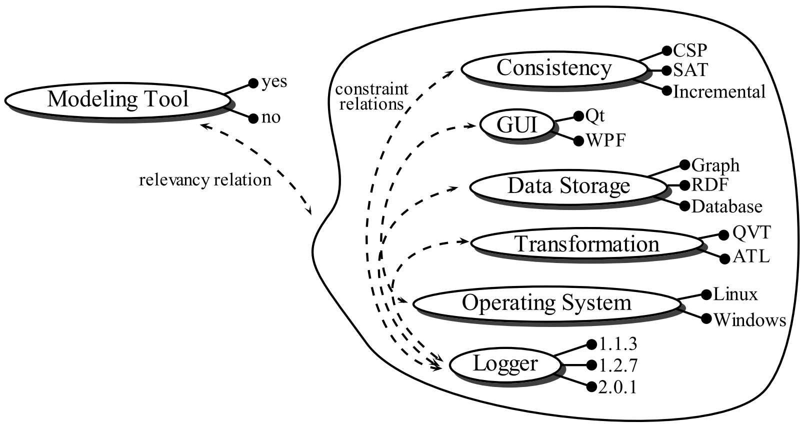


Fig. 1 Illustrative decision model for building a modeling tool

2.1 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:

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

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

$$Question = Choices \quad \Longleftrightarrow \quad Question = \big(Choices,\ [1, 1]\big)$$

Examples for question definitions are: $Requirement = \{trace, no\,trace, ?\}$, $Number = \{x \,|\, x \in \mathbb{N},\ x < 5\}$, $Traces = (\{A,\ B,\ C\},\ [1, 2])$, $Name = \{s \,|\, s\ is\ a\ \texttt{String}\}$.

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 $n \le |Answer| \le m$. 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:

$$Question \leftarrow Answer \quad \Longleftrightarrow \quad Question_{Answer}$$

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: $Requirement_{undecided}$, $Traces_{\{A,\ B\}}$, $Name_{DataAccessObject}$, $Number_3$.

2.2 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 which can be arbitrarily complex.

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 relations 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:

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

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:

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

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:

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

An example for the definition of a CNF relation is:

$$Requirement = \{trace,\ no\,trace,\ unknown\}$$
$$Traces = (\{A,\ B,\ C\},\ [1,2])$$

$$CNFRelation = (Requirement_{trace} \vee \neg Traces_{\{A,\,B\}}) \wedge (Traces_{\{A,\,C\}})$$

2.3 Illustrative example definitions

The illustration, shown in Fig. 1, includes seven questions which are defined next:

$$\begin{aligned}
Modeling\,Tool &= \{yes,\ no\}\\
Consistency &= \{CSP,\ SAT,\ Incremental\}\\
GUI &= \{Qt,\ WPF\}\\
Data\,Storage &= \{Graph,\ RDF,\ Database\}\\
Transformation &= \{QVT,\ ATL\}\\
Operating\,System &= \{Linux,\ Windows\}\\
Logger &= \{1.1.3,\ 1.2.0,\ 2.0.1\}
\end{aligned}$$

Thus the question set of the decision model tuple is: $Questions = \{Modeling\,Tool,$ $Consistency, GUI, Data\,Storage, Transformation, Operating\,System, Logger\}$. But a decision model is not complete without its relations of which five are indicated in Fig. 1:

$$\begin{aligned}
&RR = (Modeling\,Tool \rightarrow \{Consistency,\ GUI,\ Data\,Storage,\\
&\qquad Transformation,\ Operating\,System,\ Logger\})_{RelevancyRelation} = \{\\
&\quad (yes \Rightarrow relevant),\\
&\quad (no \Rightarrow irrelevant)\\
&\}
\end{aligned}$$

$$\begin{aligned}
&CR_1 = (Consistency \rightarrow Logger)_{ConstraintRelation} = \{\\
&\quad (CSP \Rightarrow \{1.1.3,\ 1.2.0\}),\\
&\quad (SAT \Rightarrow \{1.2.0,\ 2.0.1\}),\\
&\quad (Incremental \Rightarrow \{1.1.3,\ 1.2.0,\ 2.0.1\})\\
&\}
\end{aligned}$$

$$\begin{aligned}
&CR_2 = (GUI \rightarrow Operating\,System)_{ConstraintRelation} = \{\\
&\quad (Qt \Rightarrow \{Linux,\ Windows\}),\\
&\quad (WPF \Rightarrow \{Windows\})\\
&\}
\end{aligned}$$

$$\begin{aligned}
&CR_3 = (Data\,Storage \rightarrow Logger)_{ConstraintRelation} = \{\\
&\quad (Graph \Rightarrow \{1.2.0,\ 2.0.1\}),\\
&\quad (RDF \Rightarrow \{1.1.3,\ 1.2.0,\ 2.0.1\}),\\
&\quad (Database \Rightarrow \{1.2.0,\ 2.0.1\})\\
&\}
\end{aligned}$$

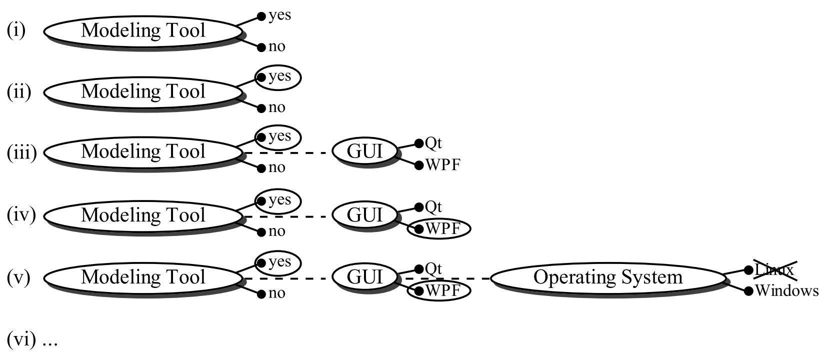


Fig. 2 Graphical illustration of the decision-making process with our example

$$
\begin{aligned}
&CR_4 = (Transformation \rightarrow Logger)_{ConstraintRelation} = \{\\
&\quad (QVT \Rightarrow \{1.1.3,\ 2.0.1\}),\\
&\quad (ATL \Rightarrow \{1.2.0,\ 2.0.1\})\\
&\}
\end{aligned}
$$

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 Problems during 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 Sect. 2.3 the initial state: $State = \{Modeling\,Tool_{undecided}, Consistency_{undecided}, GUI_{undecided}, Data\,Storage_{undecided}, Transformation_{undecided}, Operating\,System_{undecided}, Logger_{undecided}\}$, 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 $C_1 = \{Modeling\,Tool_{no}, Consistency_{irrelevant}, GUI_{irrelevant}, Data\,Storage_{irrelevant}, Transformation_{irrelevant}, Operating\,System_{irrelevant}, Logger_{irrelevant}\}$ 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. For example, the configuration $IC_1 = \{Modeling\,Tool_{yes}, Consistency_{CSP}, GUI_{WPF}, Data\,Storage_{Graph}, Transformation_{ATL}, Operating\,System_{Linux}, Logger_{1.2.0}\}$ is invalid because it violates relation CR_2 that specifies that GUI_{WPF} is only valid in combination with $Operating\,System_{Windows}$.

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 gets smaller and the solution space more complete. This process is visualized in Fig. 2, first the decision maker selects a question in our case the $Modeling\,Tool$ question (i). Given the

Table 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
Modeling Tool$_{yes}$	**1**	**1**		**1**		**1**
Modeling Tool$_{no}$		0		0		0
GUI$_{Qt}$				0		0
GUI$_{WPF}$			**1**	**1**		**1**
Operating System$_{Linux}$				0		0
Operating System$_{Windows}$				1	**1**	1

Notes: q is question, u is user decisions, s is derived state, 0 is false, 1 is true

choices to either buy or not buy a laptop, she decides on *Modeling Tool*$_{yes}$ (ii). After that as a next question *GUI* is selected (iii). After making the decision *GUI*$_{WPF}$ (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 *Linux* choice is disabled because of the relation CR_2, that states that in order to have a WPF GUI one must use Windows as the operating system. This process then continues in a similar fashion as indicated (vi).

The decision-making process could also be visualized in tabular form as shown in Table 1. For the first question the user decided *Modeling Tool*$_{yes}$ 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.

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, there exist techniques how to detect such inconsistencies (Benavides et al. 2010), however this is not the focus of the paper. 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 inconsistent 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. 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? Before we go into detail however, we must distinguish two basic cases:

1. No valid configuration exists: this happens when the decision maker configures something 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 to satisfy a customer's need.
2. A valid configuration the decision maker can live with exists, but some of the questions need to be answered differently.

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 2 Configuration progression of the example given in Fig. 1

Decision	$1^{st}q$		$2^{nd}q$		$3^{rd}q$		$4^{th}q$		$5^{th}q$
	u	s	u	s	u	s	u	s	u
$Modeling\,Tool_{yes}$	**1**	**1**		**1**		**1**		**1**	
$Modeling\,Tool_{no}$		0		0		0		0	
$Consistency_{CSP}$			**1**	**1**		**1**		**1**	
$Consistency_{SAT}$				0		0		0	
$Consistency_{Incremental}$				0		0		0	
GUI_{Qt}						0		0	
GUI_{WPF}					**1**	**1**		**1**	
$Data\,Storage_{Graph}$							**1**	**1**	
$Data\,Storage_{RDF}$								0	
$Data\,Storage_{Database}$								0	
$Transformation_{QVT}$								0	**1?**
$Transformation_{ATL}$								1	
$Operating\,System_{Linux}$						0		0	
$Operating\,System_{Windows}$						1		1	
$Logger_{1.1.3}$								0	
$Logger_{1.2.0}$								1	
$Logger_{2.0.1}$				0		0		0	

Notes: q is question, u is user decisions, s is derived state, 0 is false, 1 is true

Table 2. In this illustration technically after the fourth decision the configuration process could be stopped, since a decision was provided $\{Modeling\,Tool_{yes}$, $Consistency_{CSP}$, GUI_{WPF}, $Data\,Storage_{Graph}\}$ or derived $\{Transformation_{ATL}$, $Operating\,System_{Windows}$, $Logger_{1.2.0}\}$ for all questions. However what if the user is not satisfied with some of the derived decisions as illustrated and wants $Transformation_{QVT}$? Our vision on how to prevent and manage such inconsistencies is discussed next.

4 C2O configurator tool objectives

The main goal for the C2O configurator tool was to develop one possibility of how to present our reasoning approaches, which will be briefly explained in the next section. 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 inconsistent 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.

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. 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 order and presenting only those choices of the remaining questions that are still available (Dhungana et al. 2007). For example, after selecting $\mathit{Operating\,System}_{\mathit{Linux}}$ for the laptop, $\mathit{GUI}_{\mathit{WPF}}$ 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 automations are 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 an order can be for example to provide a certain flow of questions that belong together semantically or to minimize user input. Rearranging the order of questions reduces effort because questions have relations and answering questions makes other questions irrelevant or reduces their choices (e.g., choosing $\mathit{Operating\,System}_{\mathit{Linux}}$ makes asking about the GUI irrelevant in our example because $\mathit{GUI}_{\mathit{Qt}}$ 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. For such a small illustration, it would not even matter if the user answered the questions suboptimally 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 order—a daunting scalability problem. In such a context, a user is no longer capable of optimizing the order of questions manually (MacLean et al. 1989). If the answering of questions takes time (e.g., in steel plant manufacturing) then any reduction in the number of questions asked or choices provided will save significant effort. So as part of our vision we plan to unburden the creators of decision models to have to think about the order of questions, by automating this process.

4.2 Tools adapting to user needs, 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 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 correcting 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 a 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 (Vicente 2001), 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 order 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 order (likely) leads further away from the optimal path. Yet, we must recognize that an arbitrary order of questions, even an optimal one, may not be intuitive or appropriate (van Nimwegen et al. 2006) for every user. We should thus not impose but only suggest a particular order of questions. Allowing the decision maker to deviate from the suggested order, while still trying to suggest the optimal order 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 Sect. 2.3 if the tool forces the decision maker to answer the question about the logger first, this could already lead to inconsistencies later. To elaborate this, for example if the decision maker has no idea what features a certain logger allows to be selected and she actually does not care about the version of the logger, 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. $\mathit{Transformation}_{QVT}$) are then not available for the selected logger.

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 except there is just a single way to fix them. 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 the 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.

5 Tool and architecture

The C2O tool currently puts emphasis on the necessary reasoning behind guided decision-making. We use a SAT-Solver as our main reasoning engine. Feature Models, Product lines or general Constraint Satisfaction Problems (CSP) are transformed into Conjunctive Normal Form (CNF) and fed into the SAT-Solver. The SAT-Solver then serves as an oracle to answer questions about the impact of decisions. Before the decision-making process starts, the initial optimal order of questions is determined with the help of a heuristic (Nöhrer and Egyed 2011). During the decision-making process this order is reevaluated after every decision and the results are communicated to the user. For the creators of decision models this implies savings in not having to predefine the optimal order which is exponentially complex. For the decision maker this implies more freedom in answering questions while still benefiting from optimizations.

In addition we incorporated our approach of tolerating inconsistencies (Nöhrer and Egyed 2010b; Nöhrer et al. 2012) into the tool. It visualizes inconsistent choices, provides explanations on inconsistencies, and allows inconsistent decisions. Most importantly of all, it lets the user proceed in the presence of inconsistencies, even encourages the continuation to gather more information about the source of the inconsistency (defect), to resolve it later.

Both these technologies have been excessively evaluated. We have demonstrated that our approach for determining an ideal order is 92–100 % optimal and automatically reduces up to a third of all questions compared to a random selection (Nöhrer and Egyed 2011). We also found that at the time a inconsistency is discovered, it is around 29–71 % likely that there is more than one option for fixing it (Nöhrer et al. 2012). Yet, the longer inconsistencies are tolerated the less complex becomes their fixing (Nöhrer et al. 2012) (i.e., because decisions not only restrict choices for answering future questions but also choices for fixing inconsistencies).

5.1 Reasoning architecture overview

The reasoning architecture is based on SAT-based reasoning and split into several layers as depicted in Fig. 3. 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* 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

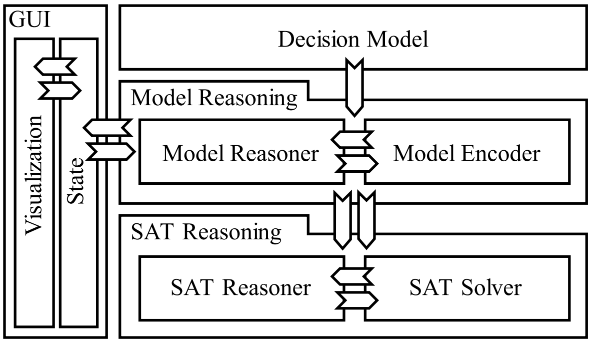


Fig. 3 Overview of the C2O configurator architecture

illustrative example $Modeling\,Tool_{yes} \rightarrow 1$, $Modeling\,Tool_{no} \rightarrow 2$, $Consistency_{CSP} \rightarrow 3$, $Consistency_{SAT} \rightarrow 4$, 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 ($Modeling\,Tool_{yes} \vee Modeling\,Tool_{no}$) and ($\neg Modeling\,Tool_{yes} \vee \neg Modeling\,Tool_{no}$)—the first clause ensures that 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. 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 $a \Rightarrow b \Leftrightarrow \neg a \vee b$. The CNF relations are already in CNF and therefore no transformation is needed.

The *Model Reasoner* is the interface between input from the 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 (Biere 2008) 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 (Berre and Parrain 2010). 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 as clauses, decisions made by the user are represented as assumptions (temporary constraints onto literals, e.g. $Modeling\,Tool_{no}$ meaning that this decision is true for the next SAT call, or $\neg$*Modeling Tool*$_{no}$ meaning that this decision is false). 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 only considered for one SAT call, as a consequence no clauses are learned. On the other hand removing clauses is a tricky thing because the solver would have to know which clauses were derived from the one that should be removed. Since 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. At this point we use the HUMUS (High-level Union of Minimal Unsatisfiable Sets) calculation [more details are given in our paper (Nöhrer et al. 2012)] to isolate all direct and indirect contributors of the inconsistency, in order to be left with uninvolved decision, which when used as a basis for reasoning are guaranteed to result in correct reasoning no matter how the inconsistency is going to be fixed.

If 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 *Modeling Tool*$_{yes}$ which would give us the satisfiable state $\{1\}$ and one possible assignment $\{1, -2, -3, 4, \ldots\}$. Removing thus the state from the assignment we would be left with $\{-2, -3, 4, \ldots\}$, next we would call the SAT solver with the assumptions $\{1, 2\}$ in order to see if *Modeling Tool*$_{yes}$ and *Modeling Tool*$_{no}$ 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 *Modeling Tool*$_{yes}$ is at least $\neg$*Modeling Tool*$_{no}$. As mentioned before other SAT calls would be made with different assumptions $\{1, 3\}$, $\{1, -4\}$ 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, making it an assumption to be included in the state.

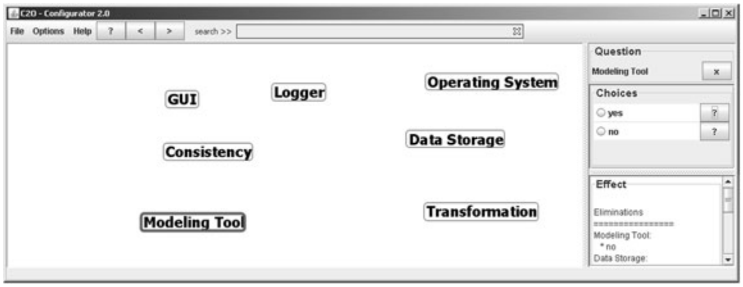


Fig. 4 Visualization of the illustrative example in the C2O configurator

5.2 GUI overview

The C2O configurator tool is built on the reasoning architecture described in Sect. 5.1 and communicates with it via events as depicted in Fig. 3. Internally it keeps a 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 Visualization aspects

The visualization aspects can be categorized into three type of aspects, (i) the general visualization of decision models as described next in Sect. 6.1, (ii) the visualization of the guidance calculation as described in Sect. 6.2 and (iii) the management of inconsistencies as described in Sect. 6.3.

6.1 Decision models

Decision models are visualized in the C2O configurator tool as shown in Fig. 4 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 decision maker is free to browse all the available questions and select the one she wants to answer first, of course during this process she is supported as described in Sect. 6.2. After each question answered the decision maker then selects a new question the same way and answers questions until all of them are answered. 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

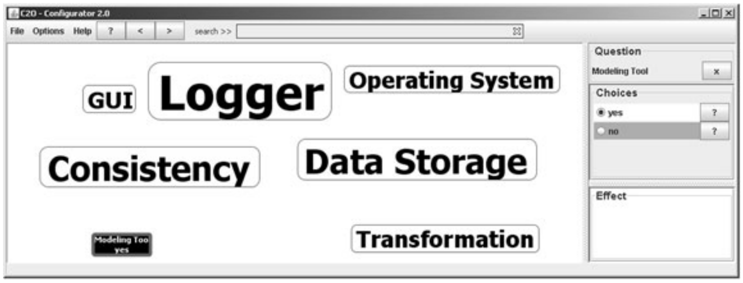


Fig. 5 Visualization of the guidance calculation

one choice one can find out what the effect of selecting the choice would be, as shown for $Modeling\,Tool_{yes}$ in Fig. 4. While some configurators have the ability to visualize constraints [e.g. the S2T2 (Botterweck et al. 2009)], usually constraints such as cross-tree constraints are 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 $Modeling\,Tool$ question are hidden from the decision maker at first, since the other questions only become relevant after the decision $Modeling\,Tool_{yes}$ was made. As soon as they become relevant they will also be shown again to the decision maker.

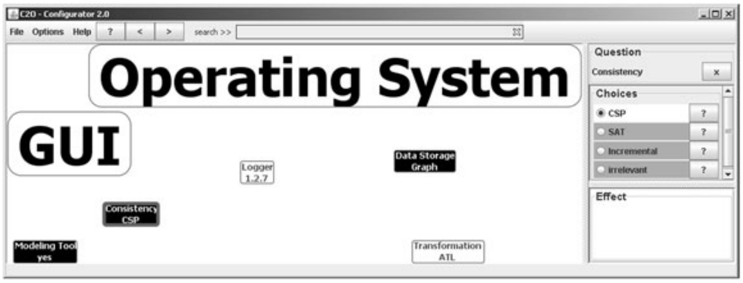
6.2 Guidance

The screenshot shown in Fig. 4 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 Fig. 5, after the decision $Modeling\,Tool_{yes}$ 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 $Data\,Storage_{Graph}$ and $Consistency_{CSP}$.

The incremental calculation and as a result the dynamic adaption to the current state during the configuration process is evident in Fig. 6, since the importance of the questions has changed. Furthermore Fig. 6 also depicts that decision can be derived automatically by eliminating choices based on relations, visualized by the grayed question captions with the selected choice shown beneath.

6.3 Inconsistencies

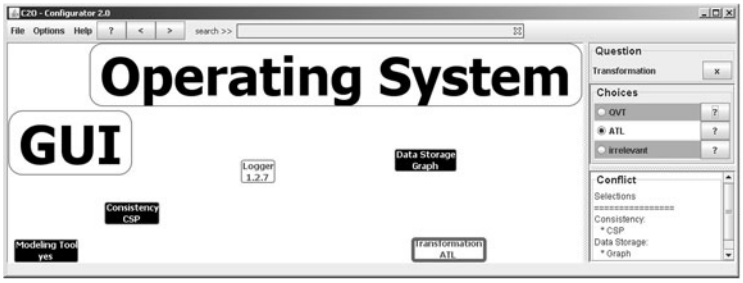
Because of automatically derived decisions, but also eliminated choices due to relations, inconsistencies might be unavoidable. For instance after the decision maker made the three decisions $Modeling\,Tool_{yes}$, $Consistency_{CSP}$ and $Data\,Storage_{Graph}$ the decision $Transformation_{QVT}$ is no longer available. As a result $Transformation_{ATL}$



Question
Consistency

Choices
(●) CSP
( ) SAT
( ) Incremental
( ) irrelevant

Fig. 6 Visualization of derived decisions

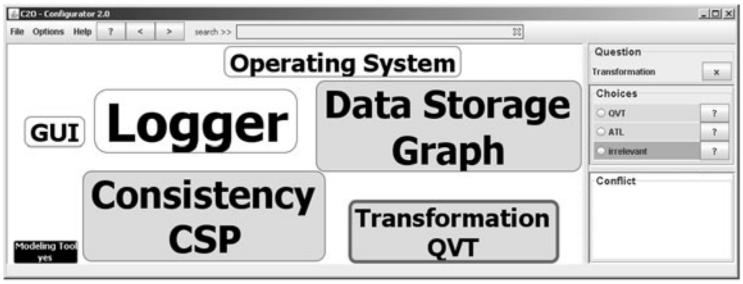


Question
Transformation

Choices
( ) QVT
(●) ATL
( ) irrelevant

Conflict
```
Selections
================
Consistency:
 * CSP
Data Storage:
 * Graph
```

Fig. 7 Explaining an inconsistency



Question
Transformation

Choices
( ) QVT
( ) ATL
( ) irrelevant

Fig. 8 Causing an inconsistency

is automatically derived and clicking on the *Transformation* question shows that *ATL* is no longer available. With the help of HUMUS the tool can now provide an explanation to the decision maker why $Transformation_{QVT}$ is no longer available by clicking onto the "?" besides the choice *ATL*, as depicted in Fig. 7. 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 Figs. 8 and 9. In Fig. 8 all contributing decisions are isolated, as visualized by the grayed out questions with their selected choice beneath including $Transformation_{QVT}$ (in this case the decisions are light gray meaning that each individually is still viable). However, one could argue that explicitly causing an inconsistency means that the decision that caused the inconsistency must be very

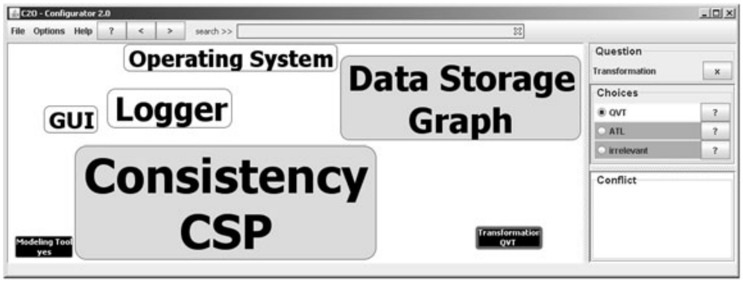


Fig. 9 Causing an inconsistency with trust

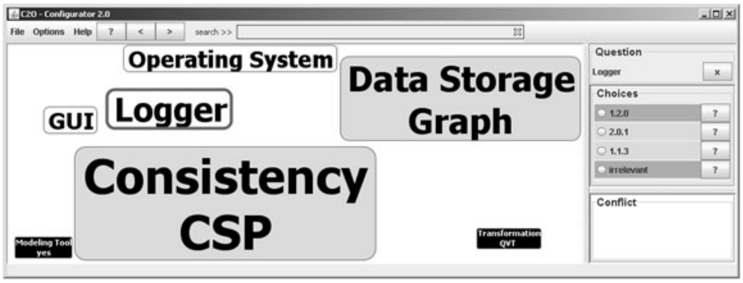
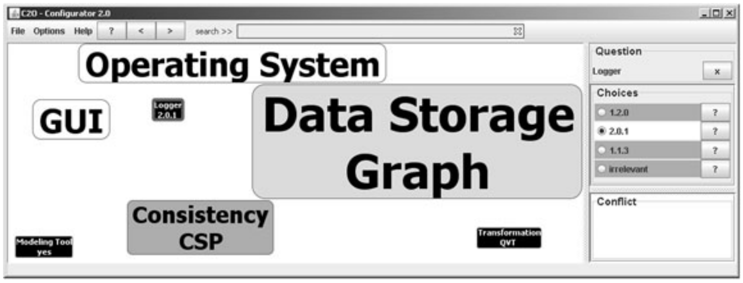


Fig. 10 Choices impacting the inconsistency

important to the decision maker, so as an alternative strategy one can decide to trust this decision to be correct, as shown in Fig. 9.

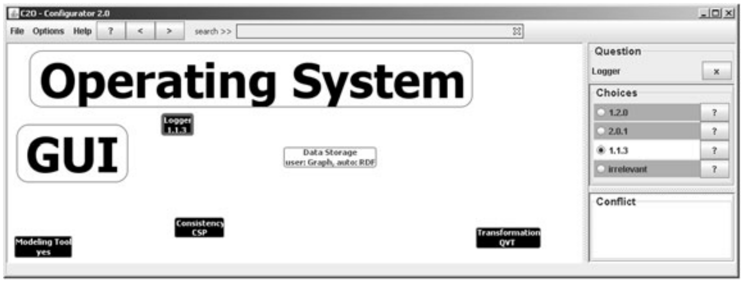
In either case decisions that were isolated from reasoning are kept in GUI and selecting questions to answer e.g. *Logger* now reveals a third state for choice visualization. In addition to a white background for choices that can be selected without causing an inconsistency (e.g. see Fig. 4 both *yes* and *no* for the question *Modeling Tool*) and to a dark gray background for disabled choices that would lead to a new inconsistency (e.g. see Fig. 10 the choice 1.2.0) 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 Fig. 11, by selecting $Logger_{2.0.1}$, the choice *CSP* is disabled for the *Consistency* question, as a result it is then visualized with dark gray background. Another example is shown in Fig. 12, by selecting $Logger_{1.1.3}$, only the choice *RDF* is viable for the *Data Storage* and $Consistency_{CSP}$ becomes compatible again with the other user decisions. As a result the answer for the *Data Storage* is changed automatically, which is indicated to the decision maker and the decision $Consistency_{CSP}$ is again displayed as a consistent user decision and the inconsistency is resolved.



Question: Logger

Choices: 1.2.0 [✓ 2.0.1] 1.1.3 irrelevant

Fig. 11 Reducing the number of possible fixes for an inconsistency



Question: Logger

Choices: 1.2.0 2.0.1 [✓ 1.1.3] irrelevant

Fig. 12 Automatically resolving an inconsistency

7 Evaluation

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. We evaluated different scenarios covering the effectiveness of the guidance calculation with respect to minimizing user input and its scalability. We also investigated the effects of living with inconsistencies during reasoning and its potential to reduce the number of possible fixes for an inconsistency.

7.1 Guidance calculation results

Our approach is incremental and the next most important question is computed always 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 order of questions to minimize user input* and is based on our paper (Nöhrer and Egyed 2011).

We validated our approach on six case studies with a different number of questions (#q), relations (#r) and choices (#c), the details are shown in Table 3. The *Illustration* used in this paper (Fig. 1) is included to compare with the others. Also listed are the *DoplerModel* for the Dopler product line tool suite (Grünbacher et al. 2008), the

Table 3 Case studies overview

	#q	#r	#c	#configurations	
				Actual	Theoretic
Illustration	7	5	23	70	3456
DoplerModel	14	8	48	105	3870720
DellModel1	28	103	147	Not computable	7.3E+17
DellModel2	24	23	147	Not computable	2.9E+15
CC-L2	59	20	137	Not computable	2.4E+20
EShop	286	147	703	Not computable	6E+115

complete laptop configurator reverse engineered from the DELL homepage (during the period of February 9th till February 12th 2009) in two versions, the *CC-L2*, a steel plant manufacturing product line model used by an Industry partner (Dhungana et al. 2011), and a feature model for a Electronic Shopping system (*EShop*) by Sean Quan Lau published on the online feature model repository of S.P.L.O.T. (Software Product Lines Online Tools website[2]). The *EShop* model was the biggest one with 286 questions. Note that this feature model was automatically converted into our own decision model (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 *DellModel1* makes extensive use of relations as opposed to the *DellModel2*.

To validate 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 Fig. 13. For validation purposes, all configurations for the smaller models and 10000 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 for the *EShop* model. Our approach is thus 78–99 % optimal and *we achieved our goal of minimizing user input*.

However the decision maker could ignore the proposed order and answer in any order (highest degree of freedom possible), while still benefiting from the reasoning about choices being eliminated and questions being answered. The question is,

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

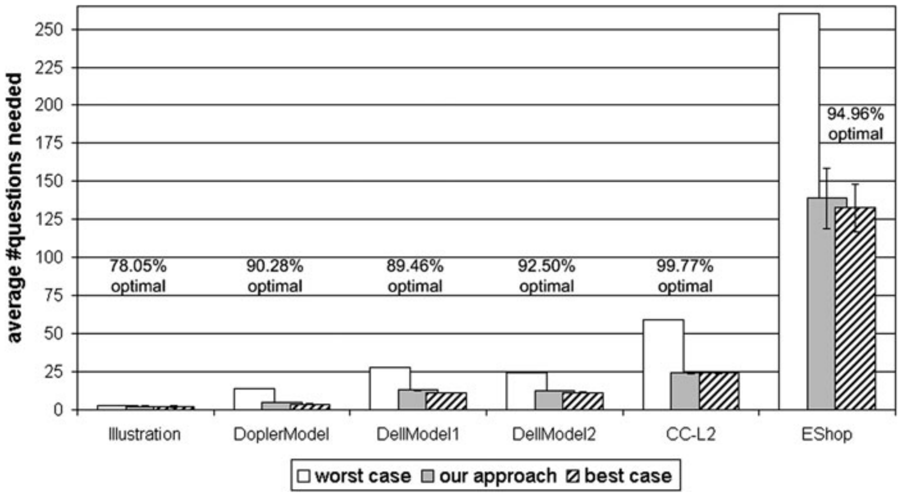


Fig. 13 Comparison optimality of the guidance calculation

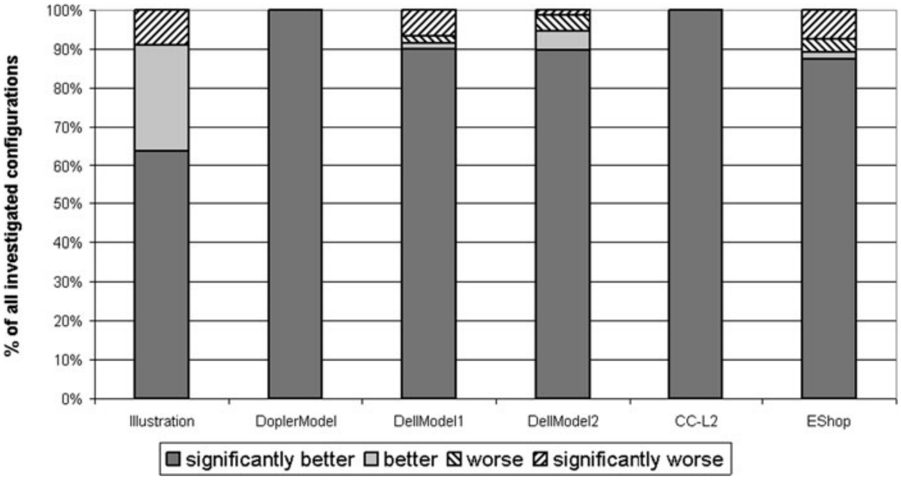


Fig. 14 Overview of t-test results comparing our guidance calculation approach to a random selection

whether following the proposed order is superior to such a more or less random order. The results for each model are shown in Fig. 14 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 order. These individual results were categorized into four classes: our approach is significantly better, better, worse and significantly worse. This comparison was based on 1000 randomly ordered 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 except for the Illustration model with $p = 0.012$. As can be

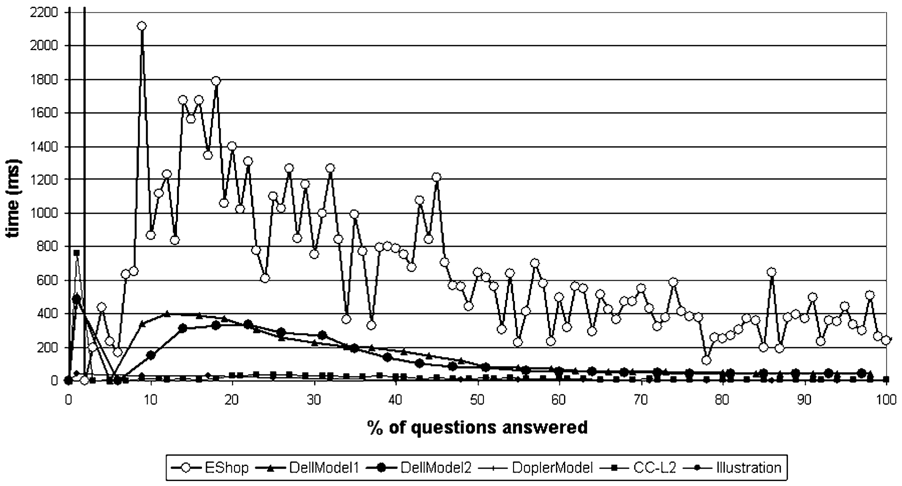


Fig. 15 Response time measurements

seen, there were cases where a random order 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).

7.1.1 Scalability

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 an order. 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) and the confidence intervals (95 %) are shown in Fig. 15 according to how many questions were answered as a percentage of the total number of questions. 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 order 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, meaning that the memory consumption of this cache grows linearly with the number of choices in the model), 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. Nonetheless, we see ample opportunities for improving even this performance which will be explored in future work.

7.1.2 Threats to validity

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.

7.2 Living with inconsistencies results

The evaluation of our approach of living with inconsistencies, based on Nöhrer et al. (2012), consists of four product line and decision models from various domains (e-commerce, decision models, feature models). The models were different in size and complexity. For example, the *DELL* e-commerce model (only a very simplified version thereof was used as illustration in this paper) had 28 questions, with roughly

Table 4 Decision models used for evaluation

Model	#q	#c	#r	#literals	#clauses
Dell	28	147	111	137	2127
V1	59	137	20	135	257
WebPortal	42	113	31	113	253
Graph	29	70	24	70	163

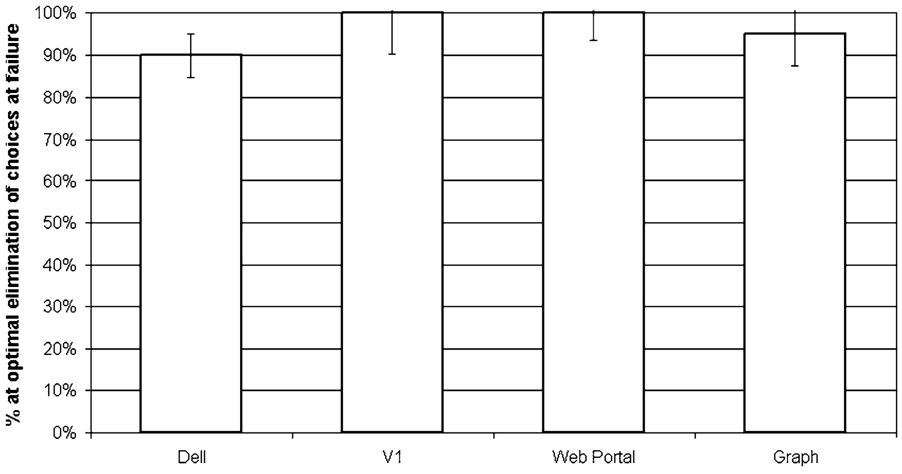


Fig. 16 Effectiveness of guidance despite conservative isolation

5 choices per question, and 111 relations. Due to the number of relations per question, this model was also the most complex decision model, as is reflected by the number of clauses. Additionally, we investigated a decision-oriented product line for a steel plant configuration (*V1*) (Dhungana et al. 2011) and several feature models (*WebPortal* by M. Mendoca, *Graph* by Hong Mei) available on the S.P.L.O.T. website (Software Product Lines Online Tools website[3]). Key characteristics of those models are stated in Table 4 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 is stated. Note that the feature models were automatically converted into our own decision model (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.

Our approach is conservative in always isolating the defect, however, at the expense of also potentially isolating correct decisions. This reduces the effectiveness of automated guidance. Recall that the aim of guidance is to disable choices of questions that are no longer allowed. The conservative nature of our approach does so less effectively. Figure 16 measures this disadvantage. The data was normalized such that

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

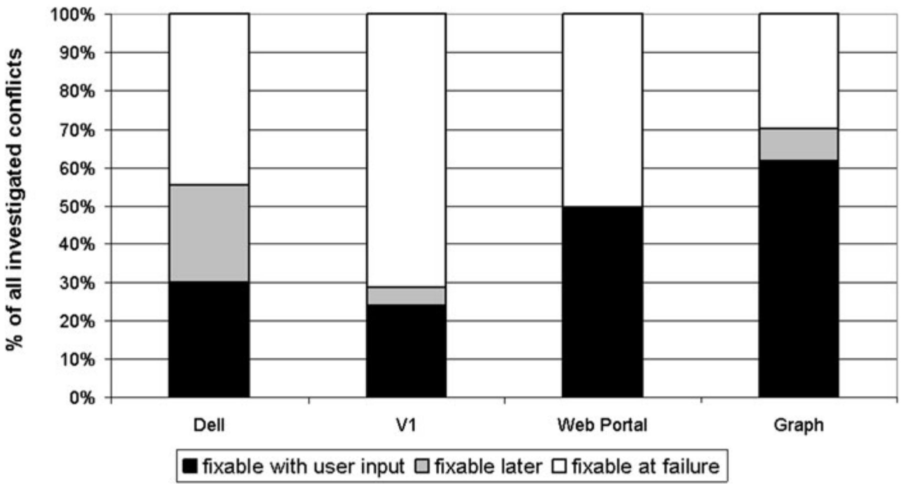


Fig. 17 Distribution of fixable situations without additional user interaction

100 % on the y-axis represents optimal guidance where only the defect is isolated from reasoning (ideal); and 0 % represents the worst case guidance where all decisions are isolated and the reasoning process starts over. We see that guidance remains 90–100 % optimal despite the conservative nature of HUMUS due to two factors we identified. One being that, if the constraints in the model have a lot of overlaps and the number of decisions identified by HUMUS is large, lost information can be replaced with new decisions due to other constraints. The other being that, if the constraints in the model have little overlaps, then the number of decisions identified by HUMUS is small.

In addition to the ability to live with inconsistencies at a minimal cost, living with inconsistencies can reduce the number of fixing possibilities, sometimes even to a single fix. We calculated the number of possible fixes at the point of the inconsistency detection and after answering each following question.

7.2.1 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*).

We can see in Fig. 17, that 29–71 % 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, 0–25 % of the remaining defects were fixable simply by letting the decision-making continue (while the

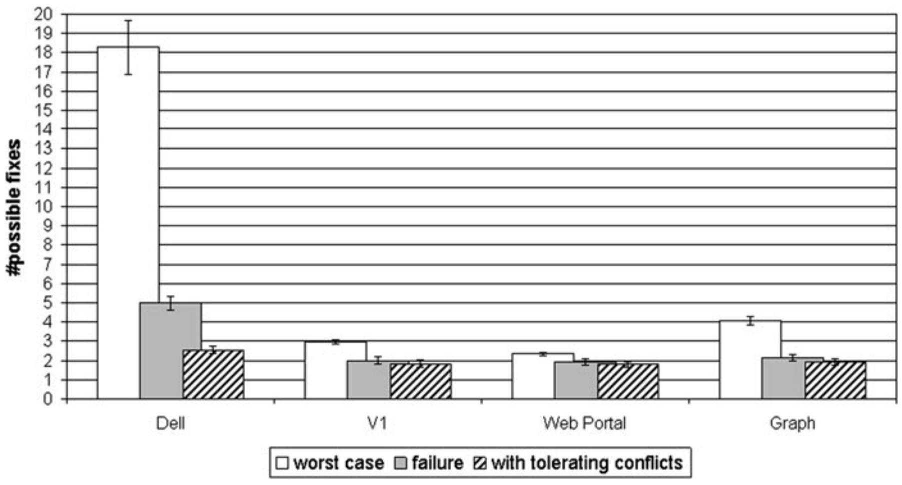


Fig. 18 Overview over the number of possible fixes

inconsistency was tolerated) without requiring additional information. Tolerating inconsistencies 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.

7.2.2 Automatically reducing choices for fixing

For the 29–71 % of defects in Fig. 17 that were not fixable at the time the inconsistency was detected, Fig. 18 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 conflicts*).

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 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

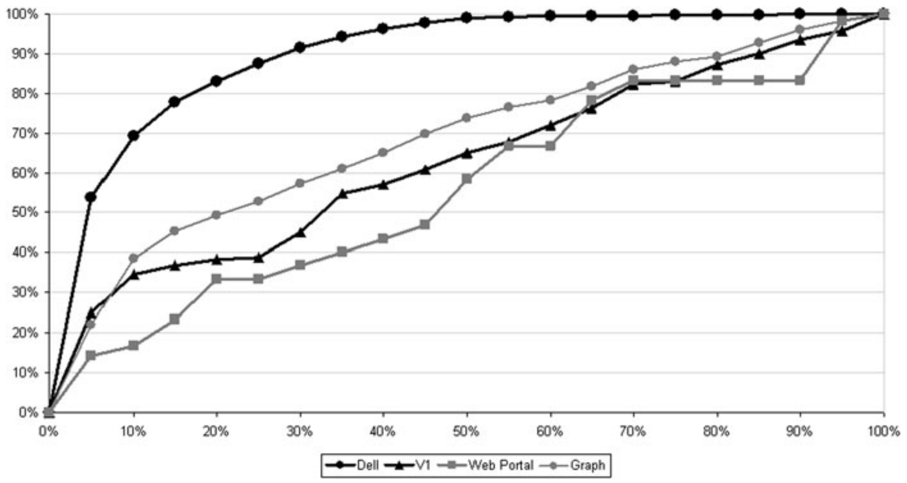


Fig. 19 Normalized progression of fix reduction

to fix defects. The improvements observed in Fig. 18 are more substantial in decision models where there are the more relations (e.g., DELL 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 Fig. 19 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. On the decision model with the most relations (DELL), we observe that 50 % of the remaining questions answered after the detection (while tolerating inconsistencies) are enough to achieve near optimal reasoning. If the decision model is less constrained, the effect appears more linear. However, we observe 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).

7.2.3 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 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 Fig. 18. This always works even if new decisions cannot be trusted and need additional fixing.

Table 5 Scalability test results on artificial SAT problems

#Contributors	10	100	1000	10000	100000
HUMUS	1 ms	3 ms	241 ms	∼28 s	∼1 h

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.

7.2.4 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 0 ms and 1 ms 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. However further evaluations on artificial SAT models (Table 5) show an exponential growth but acceptable performance for inconsistencies involving up to 10000 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 ($l_1 \vee l_2 \vee \cdots \vee l_n$) 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, it represents the worst case for our HUMUS implementation resulting in $n * n!$ SAT calls.

7.2.5 Threats to validity

Threats to construct validity imply whether we are evaluating the different isolation strategies with the proper criteria. This paper 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.

8 Related work

Product line engineering is a domain that is connected to our approach to decision-making in many ways. The process of configuring a product, therefore also often called the configuration process, product derivation or more general the decision-making process is in essence what our tool is about. 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 (Asikainen et al. 2004; Dhungana et al. 2011; Trinidad et al. 2008; Mendonca et al. 2009a; Nöhrer and Egyed 2010a; Botterweck et al. 2009), 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.

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 (Benavides et al. 2010) like for instance dead features and false optional features, to reasoning about non-functional properties of the product lines (Siegmund et al. 2011). 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, for instance, with SAT-Solvers and used for eliminating conflicting choices (Rosa et al. 2009). Translating configuration problems/feature models/decision to SAT problems (Tseitin 1983; Mendonça et al. 2009b) is solved.

SAT-based reasoning is state-of-the-practice (Mendonca et al. 2009a; Trinidad et al. 2008). It has several primary uses: SAT reasoning is used (i) to validate products (Benavides et al. 2010), (ii) to find viable alternative solutions if a product is not valid (White et al. 2008) or auto complete partial products (Mendonca et al. 2009a), and (iii) to provide guidance during the configuration process (Rosa et al. 2009; Nöhrer and Egyed 2011). 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.

Another area of related work is about guidance itself. Guidance is foremost about tool support to get the users to their goal (Haw et al. 1994; Cobleigh et al. 2006). 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 van Nimwegen et al. (2006). 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 Vicente (2001) 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 (Miller 1956) 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. (2011).

9 Conclusion and future work

In this paper we presented a configurator tool supporting 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 a reasoning strategy for living with inconsistencies during decision-making.

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.

In summary the C2O configurator tool presents a new unique way of making decisions putting the users first, giving them the freedom to make decisions and resolve inconsistencies the way they prefer to, while supporting them with the shortest path through the questions to any possible configuration and providing explanations in case of inconsistencies. Even to live with inconsistencies while profiting from correct reasoning at the expense of marginally more incomplete reasoning, thus also allowing any existing automations to work correctly in the presence of inconsistencies without adoptions needed.

We see many opportunities to improve our work further. First of all some sort of user evaluation would be nice to know how well this type of configurator will be accepted by users and how much they actually feel it helps during the decision-making process. 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 configurators. 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.

Acknowledgements The authors would like to thank Thomas Schartmüller for his work on the GUI. The work was kindly supported by the Austrian Science Fund (FWF): P23115-N23 and the Austrian Center of Competence in Mechatronics (ACCM), a K2-Center of the COMET/K2 program, which is aided by funds of the Austrian Republic and the Provincial Government of Upper Austria. The authors thank all involved partners for their support.

References

Asikainen, T., Männistö, T., Soininen, T.: Using a configurator for modelling and configuring software product lines based on feature models. In: Workshop on Software Variability Management for Product Derivation in Conjunction with SPLC, Boston, Massachusetts, USA (2004)

Balzer, R.: Tolerating inconsistency. In: 13th ICSE, Austin, Texas, USA, pp. 158–165 (1991)

Benavides, D., Segura, S., Cortés, A.R.: Automated analysis of feature models 20 years later: a literature review. Inf. Syst. **35**(6), 615–636 (2010)

Berre, D.L., Parrain, A.: The Sat4j library, release 2.2. J. Satisf. Boolean Model. Comput. **7**(2–3), 56–59 (2010)

Biere, A.: PicoSAT essentials. J. Satisf. Boolean Model. Comput. **4**(2–4), 75–97 (2008)

Botterweck, G., Janota, M., Schneeweiss, D.: A design of a configurable feature model configurator. In: Benavides, D., Metzger, A., Eisenecker, U.W. (eds.) VaMoS. ICB Research Report, vol. 29, pp. 165–168. Universität Duisburg-Essen, Essen (2009)

Cobleigh, R.L., Avrunin, G.S., Clarke, L.A.: User guidance for creating precise and accessible property specifications. In: 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Portland, Oregon, USA, pp. 208–218 (2006)

Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)

Dhungana, D., Rabiser, R., Grünbacher, P., Lehner, K., Federspiel, C.: DOPLER: An adaptable tool suite for product line engineering. In: Software Product Lines, 11th International Conference, Kyoto, Japan, pp. 151–152 (2007)

Dhungana, D., Grünbacher, P., Rabiser, R.: The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. Autom. Softw. Eng. **18**, 77–114 (2011)

Egyed, A.: Instant consistency checking for the UML. In: 28th ICSE, Shanghai, China, pp. 381–390 (2006)

Grünbacher, P., Rabiser, R., Dhungana, D.: 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, pp. 351–354 (2008)

Haw, D., Goble, C.A., Rector, A.L.: GUIDANCE: Making it easy for the user to be an expert. In: IDS, pp. 25–48 (1994)

Hayes, J.H., Dekhtyar, A.: Humans in the traceability loop: can't live with 'em, can't live without 'em. In: 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, New York, NY, USA, pp. 20–23. ACM, New York (2005)

MacLean, A., Young, R.M., Moran, T.P.: Design rationale: the argument behind the artifact. In: SIGCHI Conference on Human Factors in Computing Systems: Wings for the Mind, CHI '89, New York, NY, USA, pp. 247–252. ACM, New York (1989)

Mendonca, M., Branco, M., Cowan, D.: 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, pp. 761–762. ACM, New York (2009a)

Mendonça, M., Wasowski, A., Czarnecki, K.: SAT-based analysis of feature models is easy. In: Software Product Lines, 13th International Conference, San Francisco, California, USA, pp. 231–240 (2009b)

Miller, G.A.: The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychol. Rev. **63**, 81–97 (1956)

Nöhrer, A., Egyed, A.: C2O: a tool for guided decision-making. In: 25th International Conference on Automated Software Engineering, Antwerp, Belgium, pp. 363–364 (2010a)

Nöhrer, A., Egyed, A.: Conflict resolution strategies during product configuration. In: Fourth International Workshop on Variability Modelling of Software-Intensive Systems. Linz, Austria, vol. 37, pp. 107–114. Universität Duisburg-Essen, Essen (2010b). ICB Research Report

Nöhrer, A., Egyed, A.: Optimizing user guidance during decision-making. In: Software Product Lines, 15th International Conference, Munich, Germany (2011)

Nöhrer, A., Biere, A., Egyed, A.: Managing SAT inconsistencies with HUMUS. In: Eisenecker, U.W., Apel, S., Gnesi, S. (eds.) VaMoS, pp. 83–91. ACM, New York (2012)

Pleuss, A., Rabiser, R., Botterweck, G.: Visualization techniques for application in interactive product configuration. In: Schaefer, I., John, I., Schmid, K. (eds.) SPLC Workshops, p. 22. ACM, New York (2011)

Rosa, M.L., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Questionnaire-based variability modeling for system configuration. Softw. Syst. Model. **8**(2), 251–274 (2009)

She, S., Lotufo, R., Berger, T., Wąsowski, A., Czarnecki, K.: Reverse engineering feature models. In: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, New York, NY, USA, pp. 461–470. ACM, New York (2011)

Siegmund, N., Rosenmüller, M., Kästner, C., Giarrusso, P.G., Apel, S., Kolesnikov, S.S.: Scalable prediction of non-functional properties in software product lines. In: de Almeida, E.S., Kishi, T., Schwanninger, C., John, I., Schmid, K. (eds.) SPLC, pp. 160–169. IEEE Press, New York (2011)

Trinidad, P., Benavides, D., Cortés, A.R., Segura, S., Jimenez, A.: FAMA framework. In: Software Product Lines, 12th International Conference, Limerick, Ireland, p. 359 (2008)

Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Siekmann, J., Wrightson, G. (eds.) Automation of Reasoning 2: Classical Papers on Computational Logic 1967–1970, pp. 466–483. Springer, Berlin (1983)

van Nimwegen, C., Burgos, D.D., van Oostendorp, H., Schijf, H.: The paradox of the assisted user: guidance can be counterproductive. In: Conference on Human Factors in Computing Systems, Montréal, Québec, Canada, pp. 917–926 (2006)

Vicente, K.J.: Crazy clocks: counterintuitive consequences of "Intelligent" automation. IEEE Intell. Syst. **16**(6), 74–76 (2001)

White, J., Schmidt, D.C., Benavides, D., Trinidad, P., Cortés, A.R.: Automated diagnosis of product-line configuration errors in feature models. In: Software Product Lines, 12th International Conference, Limerick, Ireland, pp. 225–234 (2008)