Conflict Resolution Strategies during Product Configuration

Alexander Nöhrer
Johannes Kepler University
Institute for Systems Engineering and Automation (SEA)
Linz, Austria
alexander.noehrer@jku.at

Alexander Egyed
Johannes Kepler University
Institute for Systems Engineering and Automation (SEA)
Linz, Austria
alexander.egyed@jku.at

***Abstract*—During product configuration, users are prone to make errors because of complexity and lack of system knowledge. Such errors cause conflicts (i.e., incompatible choices selected) and current state-of-the-art configurators require users to undo earlier decisions made or restart the decision process altogether. This paper discusses these and other conflict resolution strategies; even ones that allow users to introduce conflicts and solve them at a later time of their choosing. This is analogous to the notion of living with inconsistencies which is not only tolerated but deemed necessary in modeling. We will discuss that allowing conflicts to exist during the configuration process (living with conflicts) is likewise beneficial during the configuration process because it is easier to resolve conflicts at a later time when the user's intention is better understood (i.e., more input was provided). However, the dilemma with living with conflicts is that traditional reasoning mechanisms become inadequate. For example, it is common during configuration to eliminate choices of future decisions (unanswered questions) based on decisions that have already been provided and we will discuss how to continue doing so in the presence of conflicts. Furthermore, we will discuss that understanding the order (history) of decisions made is beneficial for better resolving conflicts later.**

Keywords-Product Line Engineering; Formal reasoning; User Guidance

I. INTRODUCTION

User guidance during product configuration is perceived to be a straightforward activity where a user answers a set of pre-defined questions, usually by selecting among their choices. For example, an online laptop configurator is such a system. It typically has predefined questions for building a laptop, each question with a predefined set of choices (e.g. RAM 4GB or 8GB).

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 [1]. For example, after selecting a 32-bit operating system for the laptop, the 8GB RAM choice becomes unavailable. Initially all choices are available, these are then incrementally reduced as the user answers questions (decides on a choice). Current approaches to product configuration are able to restrict user choices based on decisions already provided, as for instance covered in [2-4] and other commercial and research prototype configurators. However, if the configurator does not support conflicts and a desired choice is already eliminated then the user has only one option: undo previous decisions and trying alternative combinations that might be acceptable *without knowing whether the alternatives will lead to such dead ends again!* For example, once the user encounters that the 8GB RAM choice is not available, then this requires undoing the operating system choice. However, without domain knowledge this would be hard to guess. Moreover, the user might be uncertain as to whether changing the laptop type would resolve the problem also. This leads to exponentially increasing combinations on how to resolve such conflicts. Without good tool support, users will find it very hard indeed to navigate this jungle of questions and choices – particularly, if the user is not an expert user which is the case in most situations.

To help the user in this complex conflict resolution task, for example the works by [5] could help identify those decisions that are in conflict with a desired choice. Only these decisions must be revisited (i.e., undone) which is more efficient than revisiting all decisions. We can think of such an approach as a selective undoing of conflicting decisions to be used at the time where a desired choice is not available. While such an approach reduces the complexity of the problem, it does not avoid the fundamental problem: the user still needs to navigate alternative choices of those questions in the hopes of identifying those that do not eliminate the desired choice. Moreover, in the case where multiple, alternative options exist on how to resolve the conflict; the user has to make a suboptimal decision of which answer to undo. A decision is suboptimal because the undoing would not consider future user decisions for a more optimal reasoning.

To avoid making suboptimal decisions, the alternative is to resolve conflicts at the end – after all questions have been answered. For instance the approach by [6] advocates such an alternative by helping identify conflicting decisions at any point in time, to find the one valid (partial) configuration that closest matches the desired decisions (valid configuration with minimal deviation from the user selected configuration). Such an approach allows users to select conflicting choices. However, the disadvantage of resolving conflicts at the end is that users then lose the ability to have choices reduced automatically and incrementally as answers are provided – an important feature discussed earlier. This downside exists because existing reasoning engines (e.g., Theorem proofing to check for satisfiability also known as SAT solvers [7]) do not readily function in the presence of contradicting information. The user is on his/her own which may be ok for the choices the user wanted to have despite the conflicts, but unnecessary for the other choices where multiple choices would have been acceptable. Moreover, reasoning as it is currently done, ignores the order in which questions are answered (and conflicts are encountered) which we will see later weakens the kinds of analyses we can do.

The ideal solution would be one that allows users to select conflicting choices, however, still supports incremental reasoning, such as the elimination of choices. This requires reasoning in the presence of conflicts. This is not unlike software modeling where reasoning in the presence of inconsistencies is not only tolerated [8] but even advocated as a normal way of life [9]. This paper discusses our approach of living with conflicts during product configuration. For completeness, we describe all possible conflict resolution strategies, from simple ones where living with conflicts is not necessary up to complex ones, which require living with conflicts. *All of them are useful and suitable in different situations of the configuration process. All of them have a right to exist*.

The main contribution of this paper is the ability to reduce the remaining choices of questions despite the presence of conflicts because this aspect is new and novel (i.e., not covered in related work). This is done by conservatively excluding offending decisions from the reasoning core and continuing reasoning with the subset of non-conflicting decisions. Another contribution is the use of the history of decisions made (the order in which questions are answered) as meta information for identifying the offending decisions. In essence, if a user desires a choice and accepts introducing a conflict, then this choice must be more important to the user than some previous decisions. Our approach makes use of this knowledge which is another reason why it is beneficial to still have the ability to reduce choices in the presence of conflicts to deal with multiple conflict situations. The history thus gives insights on what the user's intentions are while he/she is still configuring a product. Depending on these intentions different strategies can be utilized to resolve conflicts.

This paper is structured as follows: In Section II we describe the scenario and problem we address. This is followed by the vision of how we want to tackle the problem in Section III. In Section IV we discuss the state-of-the-art and related work. In Section V we describe in detail how our vision can be realized. The bigger picture is discussed in Section VI. Finally we draw a conclusion and give an outlook to future work in Section VII.

II. Scenario And Problem

During product configuration the preferred working mode is to answer questions by sequentially iterating over features until decisions on all variation points are made.

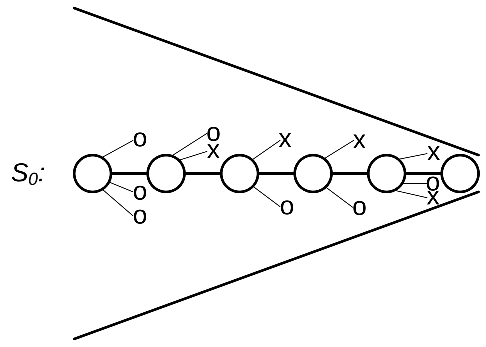

Figure 1. Normal working mode.

Since there are often dependencies among these questions (variation point, or feature), answering a question may affect other questions: it may enable them, it may reduce some of their choices, and it may even answer or eliminate them.

As a real example for a decision-oriented product-line we investigated the laptop configuration system on the DELL website [10] (during the period of February 9th till February 12th 2009) and reverse engineered its product-line model. For illustration purposes we picked three questions about the *Screen Resolution*, the *Screen Size*, and whether a *Webcam* is integrated into the screen. The choices for each question range from more complex enumerations (e.g. *WXGA 1280x800, WUXGA 1920x1200, XGA 1024x768*, etc. for the *Screen Resolution*) to a simple *yes* or *no* (for the *Webcam*). In this model, many relations between questions exist; relations can also be described as dependencies and constraints respectively. For example since no laptop with a *12.1" Screen Size* and a *WUXGA Screen Resolution* exists, these two decisions would not be compatible and result in a conflict. This and other relations are included in the model together with the choices.

Current state-of-the-art has ample tool support for eliminating choices that are no longer available after a user decision (an answer to a question). In our example this would mean, that the decision *WUXGA* for *Screen Resolution* would eliminate *12.1"* as a *Screen Size* amongst other effects. As a result of these eliminations, a question may even be answered automatically, if all its choices but one are eliminated. Figure 1 S_0 depicts this desired configuration process. It represents a decision tree (flipped sideways) where the big circles represent the decisions made. The leafs at each level represent the alternative choices that were available, but were not chosen by the user (o) or the alternative choices that were eliminated due to earlier decisions and their effects (x) – and thus were not available to the user at the time the question was being answered. The cone indicates the fact that the more questions the users has answered, the closer the user is to an actual configuration. When the last question is answered then the system is fully configured (and all variability is resolved). The cone thus denotes the number of possible configurations, which gets reduced the more questions are answered. As long as users make no decisions that conflict with earlier decisions, this approach works very well and is also well supported in [2, 11].

During the configuration process, users lacking precise system knowledge may discover at one point that a choice they desire to have is no longer allowed anymore because of earlier decisions – meaning the tool eliminated a desired choice because of dependencies. At this point in the configuration process several aspects come into play:

- Users may want an explanation why the choice is no longer available (and perhaps desire to reconsider earlier decisions made).
- Users may want to continue configuring the product and resolve the problem later.

Note, the DELL example is quite analogous to software engineering product lines which we also studied [12]. This is a small instance of a larger engineering challenge. Until a few months ago, DELL laptops could only be configured through a pre-defined order of questions at a predefined starting point (e.g., what type of laptop). Recently, DELL changed this to allow multiple starting points by means of filtering their products according to specific criteria (e.g., memory, screen size). This filtering seems not to work in every case, and not to be exact. This points to software engineers maintaining this feature independently. Also, DELL follows a rather simplistic and unsatisfactory (but easy to implement) notion of living with conflicts. Choices are not eliminated – analogous to the *Continue Manually* strategy discussed later in Section V.A.2)a). Clearly, software engineers facing similar problems to the DELL configuration system would benefit from our approach to allow arbitrary starting points and still being able to reason in the presence of conflicts.

III. VISION

Our general vision is to guide and support users but also engineers in situations that cannot be automated. This guidance should be systematic, non intrusive and most importantly allow users the highest degree of freedom, meaning:

1) *Users are allowed to make decisions in any order (if so desired).*
2) *Users are allowed to resolve conflicts at any time of their choosing.*
3) *Users should not be bothered with questions that can be answered through reasoning, meaning the interaction should be reduced to the necessary minimum.*

Of course, the users should continue to benefit from the kinds of automations they expect *despite these freedoms* – for example, still eliminating choices that are no longer available based on previous answers or in reducing the needed user/engineer input to a minimum [12]. Finally, no additional annotations should be required from the user. In other words, the user should not be subjected to providing input that goes beyond what is traditionally done during product configuration.

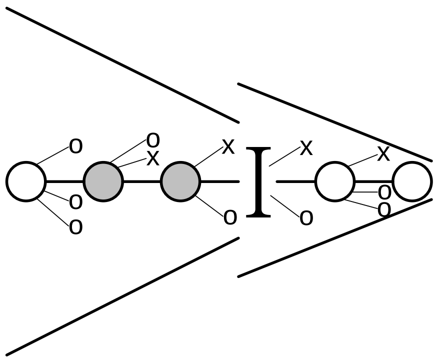

Figure 2. Vision.

To accomplish those goals we are going to tailor a reasoning engine to support reasoning in the presence of conflicts and we are going to use timeline information of the configuration process itself (when happened what). Both concepts have not been used before to resolve configuration errors, to the best of our knowledge. This will enable us to apply different strategies according to different assumptions in combination with the current state in a configuration process – all of which we will describe in detail in the next section.

This work also builds a bridge to the community that works on the problem of living with inconsistencies. Without a similar notion of “living with conflicts”, users would not be allowed to introduce conflicts or would be forced to fix them right away, which are both situations that are not always desirable. The conflict resolution strategies emphasized in this paper are thus strategies where conflicts are tolerated to some degree. This implies that users must be allowed to create conflicting situations.

Our envisioned optimal configuration process is shown in Figure 2, which illustrates the introduction of a conflict into the normal working mode. The **I** denotes the decision that introduced the conflict (inconsistency) and the shaded circles represent the decisions that **I** is in conflict with. Again the cones denote the number of possible choices left to choose from, without violating any constraints (which is analogous to how close the user is to a configuration). As a result of getting closer to a configuration the user’s intentions get more evident and could thus be used as a basis for reasoning. Also if users choose to continue making decisions before resolving the conflicts, the tool should still support them. This support should be realized by eliminating choices of the remaining questions based on the previous answers that are not involved in the conflict. As a consequence, the configuration process is still able to detect new conflicts. Since conflicting answers are no longer used to eliminate the choices of remaining questions, there are likely more choices available. The cone after a conflict is thus bigger again.

In the following, we will discuss resolution strategies for conflicts. As an example for such a conflicting situation we will build on the DELL example discussed earlier - specifically three decisions. The first decision is *12.1"* as a *Screen Size*, the second *XGA 1024x768* as a *Screen Resolution*, and the last *yes* for *Webcam*. The problem with those three decisions is that DELL does not sell a laptop fulfilling all three decisions. Only laptops with any two of those decisions are sold. The *Vostro 1310* with a *Webcam* and *XGA Screen Resolution*, the *Inspiron Mini 12* with a *Webcam* and *12.1"* as a *Screen Size*, and the *Latitude E4200* with *12.1"* as a *Screen Size* and *XGA Screen Resolution*. So assuming the user has not answered the question about the laptop *Model* yet, the *Latitude E4200* should be excluded because it has no *Webcam*. Furthermore the decision whether it should be the *Vostro 1310* or the *Inspiron Mini 12 Model* (other available models are left out for brevity), would in fact point to the real conflict in the configuration.

IV. State Of The Art

Currently different technologies exist to support the proposed normal working mode; but also to detect, explain and fix conflicts. To enable the normal working mode a few key questions need to be answered:

1) *What are the immediate effects of a decision and the elimination of a choice respectively?*
2) *What are the ripple effects of a decision and the elimination of a choice respectively?*

Once the system is modeled as a constraint satisfaction problem (CSP), these effects can be calculated with SAT- or CSP-Solvers and used for eliminating conflicting choices [11]. Translating configuration problems/feature models/ decision to CSPs is solved and described for example in [13].

As soon as the user leaves the normal working mode and introduces a conflict, other approaches are needed in addition to the checking of satisfiability. With a SAT-Solver conflicts can be detected as a result of the system not being satisfiable anymore, but normally it is not possible to explain where the conflict is coming from or even how many conflicts there are in one configuration. As a consequence different technologies are needed to detect/explain/fix a conflict.

For detecting and explaining conflicts in feature models abductive reasoning can be used as described in [5]. In the UML modeling world Egyed [14] proposed a method for instant checking and as a result detection of inconsistencies (can also be seen as a conflict). Furthermore this approach also implicitly explains why an inconsistency occurred, by pointing to the consistency rules that were violated.

Egyed also proposed methods for fixing inconsistencies in UML models [15, 16]. This technology is able to identify the concrete model elements that are violating a given consistency rules. Applied to our work, model elements are our questions and consistency rules are the relations that trigger conflicts. We believe that this technology can be used to efficiently and correctly identify offending question in case of a conflict. White et al. proposed a method for diagnosing product-line feature models in [6] that in addition proposes minimal solutions to the user. Felfernig et al. also proposed methods for resolving conflicts or as they call it: computing reconfigurations [4]. These technologies are very useful but for our approach we are going to need more than one or several solutions. One single solution or even a few different solutions almost never actually involve changes in all decisions that are involved in the conflict. Since our vision is to resolve the conflict with new decisions and reason about choices of yet not made decisions, knowing only a few solutions (not all decisions involved in the conflict) is not enough. Including decisions involved in the reasoning about future decisions would bias the results.

In the field of SAT-Solvers minimal solutions can be obtained by searching for a minimal unsatisfiable subset (MUS) of a CNF formula [17]. With the help of such a MUS, decisions that have to be changed to get to a valid configuration can be easily identified. Identifying the maximum number of satisfied constraints, and CNF clauses respectively, in a conflicting configuration is also a possibility. This can be achieved for example with algorithms that solve the Maximum Satisfiability problem (Max-SAT) [18] or solutions to over-constrained Constraint Satisfaction Problems [19]. But again the problem is that not all the decisions involved in the conflict are necessarily identified with these approaches. Nevertheless those technologies are useful for resolving conflicts and are part of the resolving strategies described in the next Section, but not applicable for our envisioned working mode.

As mentioned in the vision in Section III to ensure that users have the highest degree of freedom depends on two things. With regard to the first point that our current work [12] describes how to order questions so that the user input gets minimized without imposing the order onto the user. The order is determined automatically based on effects decisions would have on other questions. This is an incremental process that happens after each decision made by users. In addition to supporting users choosing the next question, engineers also profit from this approach as they do not need to think about an optimal order during modeling. But engineers can influence the outcome of the proposed order through special relations if they wish to do so.

V. Conflict Resolving Strategies

In this paper, 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 most important concept that we are using is the history of user decisions, as we mentioned earlier. The pieces of information the sequence reveals are very important to us and need in our opinion to be considered to effectively find solutions on how to fix conflicts. Moreover, automatically made decisions (or eliminated choices) should not be considered as important as user made decisions when taking the sequence and history of decisions into account.

Next we describe the different strategies. First and foremost, we must distinguish two basic cases:

1) *No valid configuration exists: this happens when the user configures a product that in this manner does not exist.* Eventually a conflict is found which reveals this problem. This problem can only be fixed by identifying a valid configuration that is "close" to the intent of the

user. The works by White et al. [6] solved this problem with respects to a minimal solution so we do not address it here.

2) *A valid configuration exists but a conflict was encountered.* This is possible if a previous question was answered erroneously or if the configuration process "forced" the user to answer an earlier question without the user understanding the true implications of the choices. As a consequence, a valid solution does exist, albeit some of the questions need to be answered differently.

Note that the two cases are similar in that both find a conflict. The difference is simply in the argument whether an error was made earlier that needs to be fixed (case 2) or whether no error was made and the desired configuration is simply not available (case 1). In case 1, a heuristic needs to be explored to find a "close enough" solution the user might be satisfied with (even if not desired quite as such). In case 2, we have a clear error that must be identified and fixed. No heuristics, no approximations are necessary. We will mostly focus on case 2 in section A below. Case 1 will be briefly discussed in section B.

A. *Identifying the Error in a Conflicting Configuration*

1) Fix right away: At the exact moment the user introduces a conflict 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 a conflict right away ensures that the model stays consistent and never contains a conflict (no reasoning in the presence of conflicts is necessary). It is fairly simple to realize and handle with reasoning engines, since the knowledge base stays consistent. Different strategies to fix a conflict right away are illustrated in Figure 3, where the same notation is used as in Figure 2, for sake of brevity the alternatives choices for the decisions and cone are left out. To illustrate the different strategies we again use the example given in Section II. The shaded circles represent the decisions *12.1"* as a *Screen Size* and *XGA 1024x768* as a *Screen Resolution*. The **I** represents the decision *Webcam yes*, the other white circles represent other decisions that are not conflicting with each other. Such decisions could be for examples about the CPU, RAM, hard disk, and other laptop components. Next the strategies are described in detail:

a) Single Undo: The simplest way to fix a conflict and return to normal working mode is to retract the decision that caused the conflict as illustrated in Figure 3 S_1. The user is told to try something else instead. Often this is not desired by the user since he/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 he is not completely familiar with; in such a case *Undo* might not be such a bad idea. In approaches that do not care

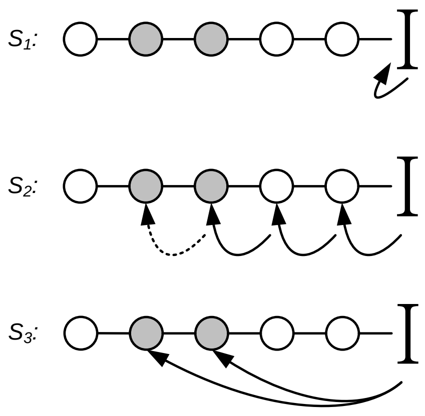

Figure 3. Fix right away strategies.

about the sequence this could also be the solution identified as the minimal solution, since it typically is less effort than changing other conflicting decisions. Applied to our example this would mean retracting the decision *Webcam yes* which certainly would resolve the conflict but may not be desired.

b) Multiple, Sequential Undo: Assuming the decision that caused the conflict is important to the user and therefore 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 is available. This could also imply retracting decisions that did not contribute to the conflict as illustrated in Figure 3 S_2 (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 conflicting decisions. Multiple, sequential undo would retract the most recent one first which could fix the conflict but may not be the desired one. This is also the case in our example, since retracting either the *Screen Size* or the *Screen Resolution* would be sufficient to resolve the conflict, which one gets retracted would depend on the order the decisions were made in.

c) Selective (Multiple) Undo: To avoid retracting valid decisions that do not contribute to the conflict, the involved decisions need to be identified. This can be accomplished with abductive reasoning mentioned earlier (Section IV). After the responsible decisions are identified they can be retracted directly as illustrated in Figure 3 S_3. This approach helps reducing the needed user input compared to the multiple, sequential undo approach (obvious 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 user has to be asked which one he/she wants to retract – a question the

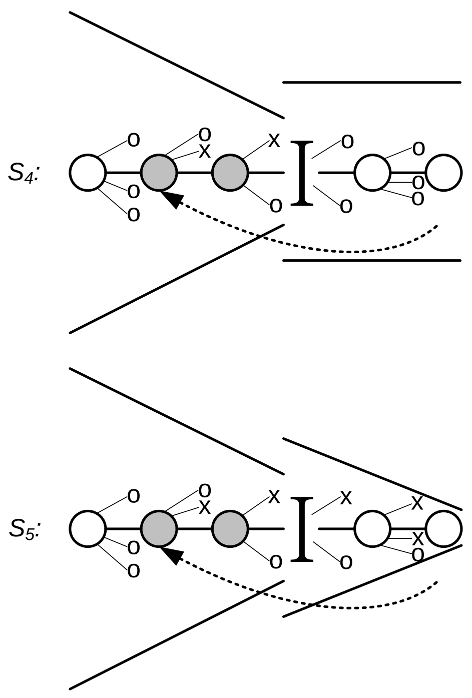

Figure 4. Allow conflicts strategies.

user may not be able to answer correctly! Again this situation can also be found in our example, since retracting either the *Screen Size* or the *Screen Resolution* would be sufficient to resolve the conflict, but this cannot be automatically decided. Selecting one of those decisions randomly or both of them is not a desirable solution. As mentioned above also asking the user about which decision to retract might not be the best thing to do.

The "fix right away" strategies are thus valid but often not desirable. However, in the absence of reasoning in the presence of conflicts, they are the only options available. The following "allow conflicts" strategies present additional options by living with conflicts:

2) *Allow Conflicts:* Instead of fixing a conflict right away, it is more beneficial to let the user answer more/all questions. The more information is collected, the better any reasoning we plan on using works. This additional information may, for example, help in deciding between two alternative options for resolving a conflict. It should be the user's decision when he/she wants to resolve the conflict. Different strategies to continue the configuration process with a conflict are illustrated in Figure 4, where again the same notation as in Figure 2 is used. These strategies are described in detail here:

a) Continue Manually: Since reasoning with conflicts is hard, the simplest way to continue the configuration process is to let the user continue answer questions without such reasoning. However, without reasoning, the user no longer benefits from knowledge of how choices are affected by decisions; i.e., what choices are still valid as illustrated in Figure 4 S_4. The negative effect would be that the user 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 user is an expert user, this resolution strategy leads to follow-on conflicts where the user unintentionally makes additional errors. For any reasonably complex system, asking the user to configure a system without automated guidance is a recipe for failure.

b) Continue with Trust: Continue with trust means, that assumptions are made on how much certain answers provided by the user can be trusted. For example, the decision that introduced the conflict is a decision that could be trusted to be important to the user – and perhaps even to be final. After all, if a choice is no longer available and the user 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 conflict could thus be considered less trustworthy. Based on this implicit trust (implied through the order in which questions were answered), the remaining decisions could still be reasoned about – at the very least to conservatively reason about which remaining choices to exclude. With this approach the user is still guided through the remaining questions and informed about decisions that would cause new conflicts as illustrated in Figure 4 S_5.

The last strategy S_5 *Continue with Trust* thus is the strategy that fulfills all our requirements presented in our vision in Section III. Applied to our example this would mean: Assuming the first decision was about the *RAM*, the second *12.1"* as a *Screen Size*, the third *XGA 1024x768* as a *Screen Resolution*, and the fourth introducing the conflict *yes* for *Webcam*. The next question could be about the laptop *Model*. Since reasoning occurs based on implicit trust the *Latitude E4200* would be eliminated since it has no *Webcam*. In addition the remaining choices *Vostro 1310* or the *Inspiron Mini 12* would help to locate the error, since the *Vostro 1310* is in conflict with *12.1"* as a *Screen Size* too and the *Inspiron Mini 12* is in conflict with the *XGA Screen Resolution*. As required by our vision the requirement that the user can continue making decisions without resolving the conflict is fulfilled. He/she was even supported in doing so through the elimination of choices that would introduce new conflicts and finally the follow up decisions helped in resolving the conflict by pointing to the error.

In addition to the implicit trust described in the *Continue with Trust* section, trust could also be based on user queries: As mentioned in the *Undo* section, 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 conflicting decisions. This idea has yet to be elaborated and tested to work out the details. But we think that it could provide useful information and help to assist users even better. In any case the decisions that can be trusted to be valid are used for a conservative reasoning process just like in the described in the strategy.

B. *Identifying a Suitable Alternative*

In case the assumption that a solution exist is wrong, meaning that the conflict cannot be resolved with the user satisfied, different strategies have to be exercised to guide the user to an acceptable solution. The details of these strategies have yet to be worked out as well, but the main ideas to get to nearest solution are:

Users can weight their decisions according to the importance to them. To avoid additional user input another possibility would be to somehow automate the weighting according to the decision history. With the help of such weights an optimal compromise could then be found via algorithms like the ones used to solve the Knapsack problem [20]. Of course also less complex solution, like finding the nearest solution as described by White et al. [6] or Felfernig et al. [4], could be sufficient.

VI. User Guidance – the Bigger Picture

Indeed, looking at the bigger picture, our vision is to provide such guidance not only to product configuration but also to design modeling and traceability management. We discuss in [12] that the user guidance problem during product configuration is not that different from the user guidance problem elsewhere. For example we want to use technologies developed in this modeling scenario also for modeling with the UML in the context of semantic constraints to ensure different UML views of a system to be coherent [15].

VII. Conclusions And Future Work

In this work, we presented our vision of user guidance for model scenarios, especially product configuration. We described strategies of how to manage and resolve conflicts during the configuration process, which we hope will also be applicable for UML and other modeling scenarios. These strategies are used for resolving conflicts during the configuration process, the reasoning occurs incrementally and is refined with every decision users make.

Once we have evaluated and validated these concrete strategies for the product configuration scenario and demonstrated that they are effective, we are planning to apply our techniques to UML modeling scenarios. During this transition we also plan to refine the roughly outlined strategies for *Trust based on user queries* and *Identifying a Suitable Alternative*. Open issues that also need to be investigated are how to handle several independent conflicts during the configuration process and how these strategies could be applied to a multi-user configurator.

Acknowledgment

This research was funded by the Austrian FWF under agreement P21321-N15.

References

[1] D. Dhungana, R. Rabiser, P. Grünbacher, K. Lehner, and C. Federspiel. *DOPLER: An Adaptable Tool Suite for Product Line Engineering*. in *11th International Software Product Line Conference (SPLC 2007), Proceedings: The Second Volume*. 2007. Kyoto, Japan: Kindai Kagaku Sha Co. Ltd., Tokyo.

[2] T. Asikainen, T. Männistö, and T. Soininen, *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 Software Product Line Conference*. 2004: Boston, Massachusetts, USA.

[3] B. Yu and H.J. Skovgaard, *A Configuration Tool to Increase Product Competitiveness.* IEEE Intelligent Systems, 1998. **13**(4): p. 34-41.

[4] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. *Intelligent Support for Interactive Configuration of Mass-Customized Products*. in *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE*. 2001. Budapest, Hungary.

[5] P. Trinidad and A. Ruiz-Cortés, *Abductive Reasoning and Automated Analysis of Feature Models: How are they connected?*, in *VaMoS*. 2009: Sevilla, Spain. p. 145-153.

[6] J. White, D.C. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortés, *Automated Diagnosis of Product-Line Configuration Errors in Feature Models*, in *Software Product Lines, 12th International Conference*. 2008: Limerick, Ireland. p. 225-234.

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

[8] R. Balzer. *Tolerating Inconsistency*. in *Proceedings of 13th International Conference on Software Engineering (ICSE)*. 1991.

[9] S. Fickas, M. Feather, and J. Kramer, *Proceedings of ICSE-97 Workshop on Living with Inconsistency*. . 1997, Boston, USA.

[10] *DELL Website*. [Accessed February 12th, 2009]; Available from: http://www.dell.com/.

[11] M.L. Rosa, W.M.P.v.d. Aalst, M. Dumas, and A.H.M.t. Hofstede, *Questionnaire-based variability modeling for system configuration.* Software and System Modeling, 2009. **8**(2): p. 251-274.

[12] A. Nöhrer and A. Egyed, *Optimizing User Guidance during Product Configuration*. 2009: unpublished.

[13] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, *Automated Reasoning on Feature Models*, in *CAiSE*. 2005. p. 491-503.

[14] A. Egyed. *Instant Consistency Checking for the UML*. in *Proceedings of the International Conference on Software Engineering (ICSE)*. 2006.

[15] A. Egyed. *Fixing Inconsistencies in UML Design Models*. in *Proceedings of the International Conference on Software Engineering* 2007.

[16] A. Egyed, E. Letier, and A. Finkelstein. *Generating and Evaluating Choices for Fixing Inconsistencies in UML Design Models*. in *Proceedings of the 23rd International Conference on Automated Software Engineering (ASE)*. 2008. L'Aquila, Italy.

[17] H.v. Maaren and S. Wieringa. *Finding Guaranteed MUSes Fast*. in *11th International Conference, SAT*. 2008. Guangzhou, China.

[18] C.M. Li and F. Manyà, *MaxSAT, Hard and Soft Constraints*, in *Handbook of Satisfiability*, A. Biere, et al., Editors. 2009, IOS Press.

[19] R.J. Wallace and E.C. Freuder, *Heuristic Methods for Over-Constrained Constraint Satisfaction Problems*, in *Over-Constrained Systems*, M. Jampel, E.C. Freuder, and M.J. Maher, Editors. 1995, Springer.

[20] G. Borradaile, B. Heeringa, and G.T. Wilfong, *Approximation Algorithms for Constrained Knapsack Problems.* CoRR, 2009. abs/0910.0777.