User-Interaction Requirements and its Implications for Efficient Implementations of Interactive Constraint Satisfaction Systems

Felix Frayman

Samilof@yahoo.com

Abstract. This paper presents a compilation of user interface requirements that arise in interactive constraint satisfaction problems. It draws implications imposed by the user interaction requirements on the architectures of constraint processing systems. It defines a special class of interactive problems called reusable CSP or RCSP. Various mechanisms of providing efficient scaleable implementations of RCSP processing systems are discussed. This paper also draws attention to the need for more careful complexity analysis of the established algorithms to take into account separation of processing into off-line and on-line stages utilizing repeatable interactions with the same problem characteristic of RCSPs.

1 Introduction

The context of this paper is interactive constraint satisfaction for reusable problems. Reusable constraint satisfaction problems or RCSPs are characterized by a fixed problem representation that is employed by multiple users who may interact with a system consecutively over time or simultaneously. Such problems arise in the configuration domain [Mittal and Frayman, 1989], where the problem itself is often represented as a variant of a standard CSP problem formulation with common configuration rules represented as a set of constraints [Frayman and Mittal, 1987]. User interactions with a constraint engine can be treated as additional unary constraints being added into the common problem formulation [Kuokka et.al., 1995]. For RCSPs, the common problem representation changes relatively infrequently – on the order of hours, days or weeks and there is usually some time available before the change and the start of user interactions.

This class of problems presents some unique challenges from the processing standpoint. In many cases it requires specialized architectures for efficient processing. This type of problem also influences the choice of algorithms that can be applied. In addition, the style of interactions with reusable CSP problems can be exploited to achieve further levels of efficiency. Given a fixed common problem formulation one may separate processing into a preprocessing or off-line stage and on-line processing that occurs for each individual user [Amilhastre and Fargier, 2000], [Calico, 2000], [Møller, 1998].

The advent of the web facilitated construction of systems that may be used by thousands of users at the same time. This increases the need for developing efficient systems that exploit every possibility for improvement. The approach of simply replicating the constraint-processing engine does not scale up to provide reasonable performance at an acceptable cost, in many situations.

The emphasis of this paper is on the reusable CSPs, but many raised issues will apply to other use scenarios as well. It should be self-evident to the reader, which points are of more general applicability.

There are several classes of users that interact with constraint systems. They may be broadly grouped into system developers and end-users. System developers are responsible for creating a common model and crafting end-user interface [Fleischanderl, 2000]. The end-users interact with a system via the presented end-user interface. This paper will focus on the end-user issues.

The paper defines a list of requirements that arises in interactive constraint satisfaction, draws architectural implications from these requirements, discusses some additional issues arising in construction of interactive systems, and points to future fruitful areas of research.

2 Requirements for the end-user experience

Following is a list of requirements for the end-user experience. It is based on the assumption of a graphical menu-based user interface.

1. Provide support for single select, multi-select and type-in style controls.

2. Support capability to select items in quantities.

3. Support the following user gestures:
    1.1. Select an item,
    1.2. Retract the selection,
    1.3. Reselect another item,
    1.4. Specify - I do not want this item,
    1.5. Restrict range of values desired.

4. Differentiate on the display between optional variables and required variables. Optional variables do not necessarily require a user selection, while required variables do.

5. Visually differentiate on the display user selections, computer selections (inductive and deductive) and computer recommended default selections.

6. Whenever possible, preclude or visually warn user about selections that may lead to inconsistency. Allow removing from the display or visually marking "inconsistent" items. Visual marking should be customizable to include "gray out" of inconsistent selections or other indicators.

7. Allow selection of "inconsistent" items even if it leads to a contradiction in the expectation that the user will correct the problem later.

8. Detect and report inconsistent selections. Reporting should be done as soon as possible after the inconsistent selection is performed.

9. Provide explanation of inconsistent selections.

10. Provide capabilities for a user to inquire why certain selections are made.

11. Facilitate calculation and display of solution aggregate functions such as total price.

12. Provide support for the display of delta aggregates, e.g. delta pricing. Delta aggregates show the individual contribution of a selection towards the total aggregate calculation.

13. Support multiple channels and/or multiple modalities of interaction – self-service web, salesperson assisted, WAP enabled PDAs and phones, audio response units, etc.

14. In cases where a web-based interface spans multiple pages, support a mechanism for navigating to the previously visited pages.

15. For certain applications it is necessary to provide support for multiple users interacting and solving different parts of the same problem.

16. Provide guaranteed response time for user interactions for a quoted number of simultaneous users.

In some cases it is a requirement to provide direct manipulation, palette type selection style of interfaces. In other cases it is sufficient to augment menu-based interface with visualization of the constructed selections [Heiderscheit and Skovgaard, 1999].

3 Architectural implications from user interaction requirements

These user interaction requirements result in a number of architectural implications.

Since web browser based interfaces are pervasive and familiar to the majority of the users, coupled with the fact that they allow one to build intranet, extranet and local

user interfaces leveraging the same technologies, we will continue the discussion under the assumption of a web-based interface. But, some of the presented points do not necessarily rely on this assumption.

3.1 Client vs. server processing

One of the most profound decisions that need to be made is where the constraint processing should occur – on the client side, on the server or combined. To evaluate this decision, it is necessary to take into account:

- Complexity of the problem,
- Client side processing capabilities,
- Client loading/transfer time,
- Security concerns,
- Accuracy of calculations concerns and
- Compatibility across client software platforms

For example, for a WAP enabled device one would have considerable limitations imposed by the communication channel transfer speed as well as the processing limitations of the device itself. Client side processing may expose the constraint model and calculations such as pricing that may be desirable to keep private. Also, client side processing may allow tampering with the model and calculations by a malicious user.

For the rest of the paper we will assume server side processing architecture.

3.2 Separation of problem formulation and session state

Most of the original constraint processing systems achieved support for simultaneous users by replicating the whole problem representation for each user session. This is an acceptable solution for a small number of simultaneous users. But, for a large number of users, memory requirements to achieve the required response time and scalability would require an exorbitant amount of computing hardware. Also, such an approach may have a negative impact on the response time when dealing with complex problems since initialization time may be significant. To alleviate this problem, more modern CSP architectures create a separation of the problem formulation and the problem solving (user session) state, which allows representing the problem once and sharing it across all user sessions on a given machine [Kuokka et.al., 1995].

3.3 Stateful vs. stateless processing engine

A user interactive session may last for a considerable amount of time, depending on the complexity of the problem. During this time a user may actually require a response from the server only a few times, the rest of the time involves the user evaluating alternatives or possibly attending to other business.

A stateful engine is resident in memory and maintains the complete state of the user interactive session up to that point in time. The client sends to the engine a new set of user requests, which are treated as delta for the current state.

In a situation of a large number of simultaneous users there is an engine allocated for each connected user, occupying memory resources. But, given infrequency of user requests, only a small proportion of the users require compute resources at any single point in time. This leads to a situation where after a period of inactivity the user session is swapped out from the memory to disk. In effect, processing time is dominated by the virtual memory swap time for this scenario.

An alternative architecture involves a stateless engine that always starts from the initial problem state and processes a complete set of user requests as opposed to the delta requests. A stateless engine approach provides better characteristics for achieving fail over and load balancing. In addition, the stateless engine eliminates the time-out issue that arises frequently with a stateful engine design. In order to save on hardware resources in a stateful situation, after a certain period of inactivity, the engine with a user state is deleted from the system for anonymous users. When the user comes back to continue the interactions, he is greeted with "Your session is timed out" message and then must re-create a complete set of previous actions. Another potential benefit of the stateless engine approach is that it is easier to achieve independence of user interactions on the results of computations.

For smaller problems stateless engine architecture may lead to faster response time, with less hardware resources. For more complex problems stateful design may be preferable since it may take too long to recreate the full problem state in the stateless engine. It is possible to create systems that are configurable and select a stateful or stateless approach depending on the complexity of the problem.

In most practical situations when the system has to service a large number of simultaneous users, the complexity of problem would be relatively low. The fates smiled on us here. One does not need to create a system for solving extremely complex problems geared towards very large number of simultaneous users.

3.4 On-line vs. off-line computations

Reusable constraint problems are characterized by the existence of a common problem formulation that changes relatively infrequently compared to the frequency of user interactions. There is usually a substantial amount of time available after the common problem formulation change until the start of user interactions. It is possible to utilize this time to perform various pre-computations such as:

- Perform problem compilation into faster representational forms. For example, one could transform certain constraints from an extensional to an intentional form or

create representations that facilitate generation of explanations [Vempaty, 1992], [Møller, 1998].

- There are some CSP processing algorithms that are naturally split into a pre-processing stage that could be performed in an off-line and on-line interactive stage that is carried out for every user session.

- It is possible to pre-compute all the solutions to certain sub-problem parts of the problem to speed-up on-line processing.

- One may also employ problem decomposition techniques based on the graph analysis algorithms during the off-line stage to optimize the performance of the on-line interactive stage.

While selecting processing algorithms for reusable constraint problems one should not rely purely on the standard complexity analysis commonly reported. It is necessary to carefully evaluate what parts of the proposed algorithms may be carried out off-line vs. on-line. Also, it is necessary to pay attention to how much memory is required to represent the individual user session state vs. the amount of memory that could be shared by a common problem representation. In some cases, this type of analysis may lead to surprising results in the algorithm selection.

3.5 Separation of the presentation and constraint processing mechanisms.

Given the requirement to support multiple channels and interface modalities, it is necessary to accurately separate presentation and constraint processing in the system. Some of the earlier interactive systems mixed together processing and presentation concerns. Although it leads to simpler construction of a system, this approach is not conducive to support multiple interfaces. In a system that provides a proper separation, it is necessary to establish a proper protocol of communication between the presentation and processing layer, leaving certain decisions to the appropriate part. For example, if a consistency algorithm eliminated a domain value, it should be up to the presentation layer to decide whether the corresponding item should be made invisible or marked as eliminated on the display. It might also be useful for the presentation layer to communicate to the processing layer whether certain variables are used for input only, input and output or output or display only. Also, separation of processing facilitates construction of interfaces that are easily adoptable to multiple languages.

3.6 Use of TMS systems

Requirements number 1.2 and 1.3 call for support of non-monotonic operations of selection retraction and reselection. One way to achieve this functionality with a stateful engine is via justification type truth maintenance system (TMS) [Doyle, 1979]. TMS system may also be used in the generation of explanations. One potential

drawback with TMS is the associated overhead for maintaining the required data structures. Also, one needs to pay special attention to insure sequence independence of the results given non-monotonic operations.

3.7 User interactions as unary constraints

It is possible to treat user gestures listed in the requirement number 3 as unary constraints. This allows graceful treatment of user interactions for adoption of the existing CSP algorithms.

4 Additional considerations in user interactions

This section of the paper covers additional considerations for constructing good user interaction experience.

4.1 Explanation of conflicts

Requirement # 9 necessitates providing an explanation for inconsistent selections. A good explanation helps the user to understand causes of the detected conflict. A conflict may simply be the result of a failed constraint in which case explanation might be fairly straightforward. A failure may also be detected as a result of a consistency algorithm application, where the previously selected value is eliminated or all values are removed from the domain of a variable. In this case explanation may be considerably more involved. Explanation may reference a subset(s) of selections that cause a conflict and possibly contain references to the violated constraints. In some cases there might be multiple dependent or independent conflicts generated by a set of user actions.

It is possible to consider using a TMS system [Doyle, 1979] as the basis for generating explanations. But, it might be a fairly heavy mechanism just for this purpose. Alternative approaches were proposed by [Amilhastre and Fargier, 2000], [Calico, 2000], [Freuder et.al., 2001], and [Junker, 2001b]. When constructing good explanations one needs to consider explanation length as well as understandability in addition to the computational overhead in creating explanations.

One step beyond explanation of the failure is a set of suggestions on how to fix the conflict. In general, it is a non-trivial task to propose a viable conflict resolution taking into account explicit or implicit user preferences that are guaranteed to be conflict free [Freuder and O'Sullivan, 2001].

4.2 Proper representation of eliminated items

Requirement # 6 calls for providing visual indicators for selections that will lead to a contradiction. If a selected visual indicator is gray out, then the meaning of such an indicator is – "if a user selects a grayed out item it will lead to a conflict." Care must be exercised in truthfully supporting such interpretations in the presence of non-monotonic user operations such as reselection. Given a control with a selection on it, it is necessary to isolate the eliminated values on this control that are attributed to the selection itself [Calico, 2000].

4.3 Defaults

Requirement # 5 entails providing the notion of default selections or preferences [Junker, 2001b], [Brewka, 1989]. There are two ways to treat default selections. One would be similar to user selections. Another would be to treat them as a recommendation that is activated unless it contradicts already made selections. A default in this interpretation would quietly go away if it is in conflict. The second interpretation provides more flexibility in problem formulation. But, it creates additional processing difficulty due to the implied non-monotonicity of this interpretation. In this interpretation one also needs to handle with care defaults that may change on the previous pages in a multi-page user interface to avoid user surprises.

4.4 Earliest detection of conflicts

Requirement # 8 requests the earliest possible reporting of inconsistent selections. This is an argument in favor of performing full consistency which in most situations is computationally not feasible. In practice, most systems provide low order consistency. Some systems provide adaptive consistency algorithms based on the time allotted to satisfy response time requirements [Yu and Skovgaard, 1998]. This might be a preferred solution, granted one can gracefully explain to the user differences in the results given the same input.

4.5 Consideration for constructing multi-page interfaces

When constructing a user interface for a complex CSP problem one is faced with a decision of creating a single page or a multi-page user interface. When creating a multi-page user interface it is necessary to decide on the grouping and sequencing of the pages. There are several considerations that need to be taken into account in the construction of such interfaces:

- Consider grouping interactive controls by a common theme.

- From the user standpoint is there a natural sequence of actions?

- How many controls to display on a single page? Some designers prefer to avoid the user scrolling through the page, which imposes a limit on the size of a single page.

- What style of controls to use? – Drop-downs or radio buttons. It involves the trade-off between being able to present more info vs. the required screen real estate.

- Should one use auto-submit or submit on demand? One needs to balance probability of a conflict vs. delay time and potential flashing of the screen. More tightly constrained problems may require more frequent engine interactions to avoid conflicts.

- Are there natural problem decomposition boundaries from the CSP graph analysis perspective? How does it interact with the first two issues?

4.6 Ways to improve system responsiveness

Following are several means of improving responsiveness of a system:

- For loosely constrained CSP problems in a stateless engine scenario, it is beneficial to run with explanation off until a conflict is detected.

- In the case of a stateful engine without the TMS system, it is possible to continue the computation for monotonic user requests and restart the engine for non-monotonic requests, leading to a partially stateful system.

- For stateless engine operation it is possible to maintain a pool of pre-initialized engines to speed-up the user response time.

5 Conclusion and directions for future research

This paper provided a list of user interface requirements for constructing good user experience. A special class of CSP problems called reusable CSP problems or RCSPs was identified. This class of problems presents opportunities to improve the system performance relying on the special structure of such problems. We pointed out to some ways to exploit this structure. Additional research could be warranted in the following areas to further exploit the nature of such problems:

- One may experiment with various caching schemas for memorizing results of computations across user sessions.

- Additional performance improvements could be gained by learning across the user sessions. Such learning may involve nogoods, value and variable ordering heuristics, tuning problem decompositions, etc.

- Further exploration is warranted of the techniques to preserve deeper learning after the constraint problem change, similar in spirit to the work by [Geneste and Ruet, 2001].

- More time could be devoted to devising and experimenting with algorithms that are partitioned into off-line and on-line stages for optimum on-line stage performance.

References

[Amilhastre and Fargier, 2000] J. Amilhastre and H. Fargier. Handling interactivity in a constraint-based approach of configuration. In ECAI-2000 Workshop on Configuration, 2000.

[Brewka, 1989] G. Brewka. Preferred subtheories: An extended logical framework for default reasoning. In IJCAI-1989, 1989.

[Calico, 2000] Calico. Calico Advisor and Visual Modeler Training Guide. 2000.

[Doyle, 1979] J. Doyle. A Truth Maintenance System. Artificial Intelligence, 12, 1979, pp. 231-272.

[Fleischanderl, 2000] G. Fleischanderl. User Interface Requirements for Knowledge Acquisition and Modeling. In ECAI-2000 Workshop on Configuration, 2000.

[Frayman and Mittal, 1987] F. Frayman and S. Mittal. Cossack: A Constraints-Based Expert System for Configuration Tasks. In D. Sriram and R.A. Adey, eds. Knowledge Based Expert Systems in Engineering: Planning and Design. Proceedings of the Second International Conference on Applications of Artificial Intelligence in Engineering Problems, Cambridge, Mass., August 1987. Computational Mechanics Publications. pp. 143-166.

[Freuder et.al., 2001] E. C. Freuder, C. Likitvivatanavong, and R. J. Wallace. Explanation and Implication for Configuration Problems. In IJCAI-2001 Workshop on Configuration, 2001.

[Freuder and O'Sullivan, 2001] E. C. Freuder and B. O'Sullivan. Modeling and Generating Tradeoffs for Constraint-Based Configuration. In IJCAI-2001 Workshop on Configuration, 2001.

[Geneste and Ruet, 2001] L. Geneste and M. Ruet, Experience based configuration. In IJCAI-2001 Workshop on Configuration, 2001.

[Heiderscheit and Skovgaard, 1999] D. Heiderscheit and H. J. Skovgaard. Visualization of Configurations: Simplifying Configuration User Interfaces. In AAAI-1999 Workshop on Configuration, 1999.

[Junker, 2001a] U. Junker QuickXplain: Conflict detection for arbitrary constraint propagation algorithms. In IJCAI-2001 Workshop on Modeling and Solving Problems with Constraints, 2001.

[Junker, 2001b] U. Junker Preference Programming for Configuration. In IJCAI-2001 Workshop on Configuration, 2001.

[Kuokka et.al., 1995] D. Kuokka, S. Jefferson, L. Barford and F. Frayman. A parametric design assistant for concurrent engineering. In Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 9, 1995, pp. 135-144.

[Mittal and Frayman, 1989] S. Mittal and F. Frayman. Towards a generic model of configuration tasks. In IJCAI-1989, 1989.

[Møller, 1998] G. L. Møller. On the Technology of Array-based Logic. Ph.D. Thesis, Technical University of Denmark, Second Edition, Array Technology, 1998.

[Vempaty, 1992] N.R. Vempaty. Solving constraint satisfaction problems using finite state automata. In AAAI-92, pp. 453-458, 1992

[Yu and Skovgaard, 1998] B. Yu and H. J. Skovgaard. A configuration Tool To Increase Product Competitiveness. IEEE Intelligent Systems, Volume 14, Number 4, July-August 1998.