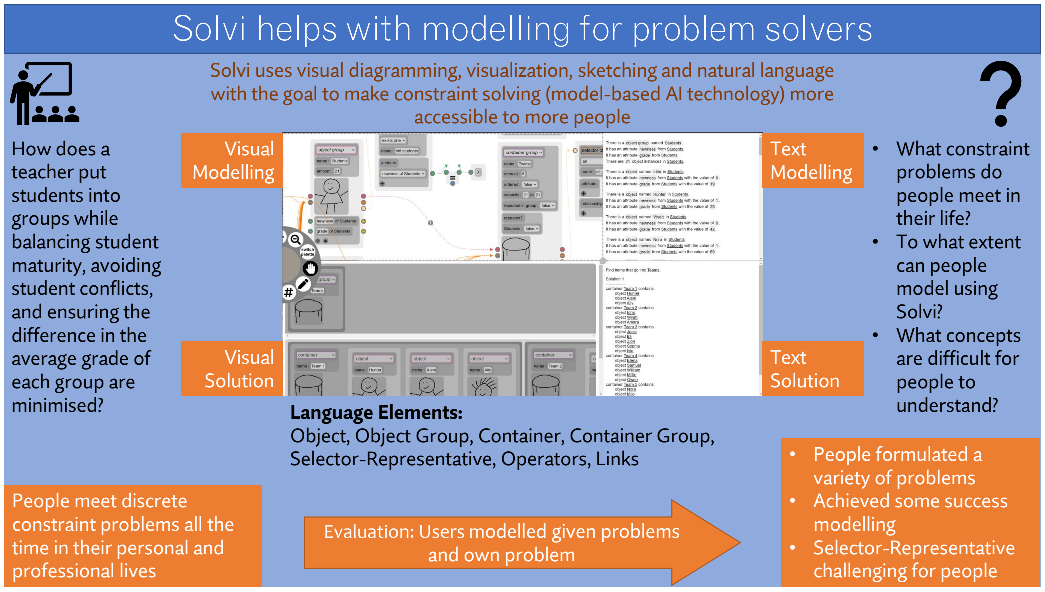
Graphical Abstract

Solvi: a Visual Constraint Modeling Tool

Xu Zhu, Miguel A Nacenta, Özgür Akgün, Daniel Zenkovitch



Solvi helps with modelling for problem solvers

Solvi uses visual diagramming, visualization, sketching and natural language with the goal to make constraint solving (model-based AI technology) more accessible to more people

How does a teacher put students into groups while balancing student maturity, avoiding student conflicts, and ensuring the difference in the average grade of each group are minimised?

- What constraint problems do people meet in their life?
- To what extent can people model using Solvi?
- What concepts are difficult for people to understand?

Language Elements:
Object, Object Group, Container, Container Group, Selector-Representative, Operators, Links

People meet discrete constraint problems all the time in their personal and professional lives

Evaluation: Users modelled given problems and own problem

- People formulated a variety of problems
- Achieved some success modelling
- Selector-Representative challenging for people

Highlights

Solvi: a Visual Constraint Modeling Tool

Xu Zhu, Miguel A Nacenta, Özgür Akgün, Daniel Zenkovitch

- We contributed a language and tool (Solvi) that leverage knowledge of how non-experts conceptualize problems to facilitate the expression of constraint models.
- We conducted a quantitative user study using Solvi that surveyed the advantages of the tool and remaining challenges towards making constraint problem modelling accessible to the wider public.

Solvi: a Visual Constraint Modeling Tool

Xu Zhu[a], Miguel A Nacenta[b], Özgür Akgün[a], Daniel Zenkovitch[b]

[a]*School of Computer Science, University of St Andrews, St Andrews, Fife, United Kingdom*
[b]*Department of Computer Science, University of Victoria, Victoria, British Columbia, Canada*

Abstract

Discrete constraint problems surface often in everyday life. Teachers might group students with complex considerations and hospital administrators need to produce staff rosters. Constraint programming (CP) provides techniques to efficiently find solutions. However, there remains a key challenge: these techniques are still largely inaccessible because expressing constraint problems requires sophisticated programming and logic skills. In this work we contribute a language and tool that leverage knowledge of how non-experts conceptualize problems to facilitate the expression of constraint models. Additionally, we report the results of a study surveying the advantages and remaining challenges towards making CP accessible to the wider public.

Keywords: constraints programming, visualization, visual modelling, visual language, human computer interaction

---

1. Introduction

In professional and personal life, people are often faced with what mathematicians and computer scientists call *constraint problems*: problems in which the solution involves a set of states of objects that satisfy certain conditions (the constraints). For example, when a teacher assigns students to project groups, the teacher might want to enforce certain group constraints,

Email addresses: `xz32@st-andrews.ac.uk` (Xu Zhu), `nacenta@uvic.ca` (Miguel A Nacenta), `ozgur.akgun@st-andrews.ac.uk` (Özgür Akgün), `danielzenkovitch@uvic.ca` (Daniel Zenkovitch)

such as each group having at least one extrovert, and global constraints such as all groups having students from multiple sexes. Other examples include assigning tasks to different members of a team, building a family schedule, nurse rostering in hospitals, or planning a wedding seating arrangement.

Researchers in the research communities of constraint solving, constraint satisfaction problems, operations research and others have devoted much time and resources to develop theories, methods and software that efficiently help find solutions for constraint problems (e.g., [1, 2]). Existing programming languages and constraint solvers allow their users to express problems as *problem models* and, for a large number of those problems, efficiently find a number of solutions, or show that no solution complies with all the constraints.

We believe that the research areas of visual languages, visual representation (including Information Visualization) and HCI/interface design can contribute new approaches to creating more accessible constraint problem specification. Visual languages seem particularly fitting to represent CP problems because these involve many objects that have complex relationships between them (constraints) and, unlike textual constraint programming languages, naturally provide a diagrammatic overview of the problems and might require less specialized skills to read and write. Additionally, visual languages might enable expression of problems in ways closer to how people naturally describe this type of problems [3, 4, 5].

The main goal of this paper is to explore a novel alternative to express constraint models that will be more accessible. We aim at extending the target audience as much as possible, but recognize that proficient users of constraint programming languages are unlikely to need or prefer such system. On the other end of the spectrum, some novel users might not be able to express any model if they lack the basic logical or mathematical concepts. Thus, our secondary goal is to assess to what extent non-experts in constraint programming languages can effectively express constraint problems when supported by a language and interface design for this purpose.

Building upon previous work on how people naturally describe constraint problems [5, 4], we designed the first visual language for constraint problem specification and implemented a novel prototype tool that uses this language while applying several UI design innovations. The tool, which we call Solvi, enables people to create visual descriptions of constraint problems, supports them checking that the model expresses what they wanted to express (through an alternative natural language representation), translates the models to a state-of-the-art CP language, sends it to a solver, and visualizes the solutions. We also contribute a study that assesses the effectiveness of the tool and, perhaps more importantly, identifies which concepts are more challenging for non-expert users.

2. Example Problem

We now provide an example scenario where a constraint problem needs a solution. It serves as a running example throughout the rest of the paper to introduce Solvi's design. We selected this scenario according to three criteria: it must allow us to demonstrate most of the features of the language and give an indication of its expressive power, it has to be simple and familiar enough to be readily understandable for the reader, and it must be a plausible representative of a real task. Some readers might object that the example below can be reasonably solved by hand in a time comparable to what it would take to express the model. Although this is tenable, consider that the human effort and time required grow fast with increasing quantities of elements and constraints, that tasks would often require successive small fixes that might force the human solving process to start from scratch, and that humans can easily miss very advantageous solutions when they take shortcuts in the name of tractability and timeliness.

> Taylor is a teacher at a high school who is planning a project for the students on a class. Taylor's class has 21 students, some of which are new to the school, with a range of previous grades that are presumed to somewhat reflect their academic skills. Taylor wants to create 5 teams of three to five students but also wants to make sure that new students can benefit from the experience of other students at the school (and have the opportunity to connect with the existing social fabric). Taylor ensures this by having at least one older student in each team. Finally, the assignment should take into account that students Idris and Ally had a conflict in the past and the school advisor has recommended that they do not sit together in a team.

What kind of team assignments are possible? Can Taylor also balance out the average grade in the teams so that all teams have similarly strong chances at a good grade?

3. Background and Related Work

This section provides the necessary background regarding constraint programming (CP), problem modeling and existing systems for visual programming and visualization in CP.

A constraint satisfaction problem is a problem that can be expressed through decision variables and a set of constraints. Each decision variable has an associated domain encoding the potential values it can take. A constraint is a condition on a subset of the variables that limits the values they are allowed to take. Constraint Programming (CP) [6] is a declarative method for stating and solving constraint satisfaction problems. CP is successfully applied in many high-impact areas such as timetabling, staff rostering, logistics, production planning and experiment design [7, 8].

The process of applying CP to a problem can be crudely divided into two parts: *modeling* and *solving*. Once a problem is modeled into a suitable language, it can be automatically solved using a constraint solver. For complex real life problems, the modeling step presents a real difficulty: capturing a correct and efficient model is hard, even for experts. High-level modeling languages like Essence [9, 10] and Zinc [11] reduce the need for this expertise somewhat through abstract domain types like sets, functions, and relations.

This paper is the first on the topic of supporting the CP modeling process through visual means and visualization. The existing constraint visualization systems focus on debugging or for understanding the progress of the *solving* process, mainly through visualizing the search-tree and result set. Existing systems allow the user to create a visualization for a specific problem by visualizing the underlying constraint network directly [12], or by creating vector images and writing code to link the images to the model [13], and sometimes add a time dimension to allow visualizing the progress over time [14, 15]. In contrast, other systems allow creating a search-tree based visualization in a model-independent way [16, 17]. There is also preliminary research to understand user expectations for search-tree based visualizations [5, 18].

Existing research recognizes problem representation as one of the key elements or stages for solving a problem. Scientists often propose notations as a way to advance their fields [19, 20]. Writing and sketching are often seen to be a natural extension of internal mental processes and help to augment human memory and processing capacity [21] and have a significant role in the visualization and understanding of data [22, 23]. The role of representation has been studied in educational contexts [24], for understanding how people build models of working systems [25], and for problem modeling, sketching and visualization of large datasets [26].

Visual programming languages primarily aim to make programming accessible to broader audiences by managing the complexity of specifying systems that are highly interconnected [27, 28, 29][1][2]. Scratch is a prominent example that uses drag and drop blocks to specify a program instead of writing code [30]. These visual programming languages are typically for procedural languages rather than declarative and are not free from their own limitations such as scalability [31] and clutter [32]. Theoretical aspects of the design, parsing and specification of visual programming languages are extensively discussed by Marriott and Meyer [33].

A related technique that aims to make the CP modeling process easier is *programming by example*, where a model is synthesized using examples of correct and incorrect solutions to a particular problem [34, 35, 36, 37]. These systems do not require their users to have any knowledge of CP modeling, however the correctness of a synthesized model is difficult to check and often impossible to prove. For complex problems, the number of required examples can be extremely large.

4. Design Goals and Principles

We designed Solvi with the overarching goal of making constraint solving technology more accessible for personal and professional problem solving. We further make the overarching goal more explicit through four main objectives. The design of Solvi aims at:

- O1 Enabling modeling and communication of a range of constraint problems that is as broad as possible, for a broad range of people.
- O2 Solving and representing solutions to the modeled problem effectively.
- O3 Supporting situated use in locations beyond programming stations (i.e., computers that require at least semi-dedicated spaces).
- O4 General comfort, learnability and ease of use.

[1]`https://cycling74.com/products/max/`
[2]`https://www.mathworks.com/products/simulink.html`

To accomplish those objectives, we selected a design approach based on six key design principles:

DP1. Visual Representations. Many have posited that visual representations are key facilitators of understanding and communication (e.g. [38, 39]), which aligns also with O2; additionally, visual languages are a common approach to address formalization by non-programmers ([40, 32], supports O1). Finally, Zhu et al. found that most people asked to express constraint problems can effectively use graphical representations to a large extent, although not exclusively [4].

DP2. Flexible composability. To achieve the expressiveness of O1 we require substantial flexibility, which is usually offered by a language that enables a limited number of elements (tokens) to be used multiple times and combined in multiple different ways. We followed an atomic approach similar to that reported by Méndez et al. [29, 41], itself based on educational constructivist and constructionist philosophies [42, 43].

DP3. Underspecified to specified. Problem understanding by people is progressive, and partially facilitated by the externalization of the problem model itself (e.g., [44]). It seems reasonable to support a journey in which the problem holder interacts with the software gradually to build the formalization of their problem at the same time that they build their understanding of the problem. We therefore have to assume that the problem starts being underspecified and gets gradually refined through interaction with the system as part of the interaction loop.

DP4. Multiple Notations. Zhu et al. [4] observed that, although graphical representations are useful for people to express problems, different groups of people rely also on more textual notations. A multi-notational approach that uses a visual language in combination with textual representations might better support a broader range of users (O1) as well as provide alternative feedback that assures the problem holder that their model is consistent with their problem (O2,O4).

DP5. Graphical Freedom. Previous research in other domains suggests that the arrangement and appearance of elements in computerized representations is important for people to preserve their mental map and to effectively locate and recognize these elements (e.g., [45]) which, in turn, would affect O4.

Rather than constrain the arrangement and appearance of items represented in the problem, we aim to support a degree of graphical freedom, especially encouraging familiarity of the representation to allow people to relate to their prior experiences. Zhu et al. [4] also found that people also use sketches or shorthand symbols (e.g., happy or sad smileys) to represent elements in constraint problems, perhaps for these reasons.

DP6. Bottom-up and Top-down. Problem specification by humans sometimes takes place in a bottom-up way (e.g., starting with examples and then generalizing to more abstract properties), and sometimes top to bottom (e.g., specification of the structure of the problem first, followed by providing the specifics, or data). Supporting both approaches to the extent possible is consistent with O4, DP3 and avoids Blackwell and Green's *premature commitment* issue [46].

These design principles represent the best knowledge available to us for the design of Solvi, and justify the main design choices described in Section 5. Naturally, many other approaches are possible.

5. Solvi: Design

Solvi is both the visual constraint modeling visual language and the web interface that implements it. In this Section, we use the example from Section 2 to introduce the main elements of Solvi and how they work together to model constraint problems. We reference to items in Section 4 when the design of a feature has been motivated directly by an objective or principle (e.g., O1, DP3). We use SMALL CAPS to introduce the names of features in the interface and *italics* to refer to constructs of the Solvi language.

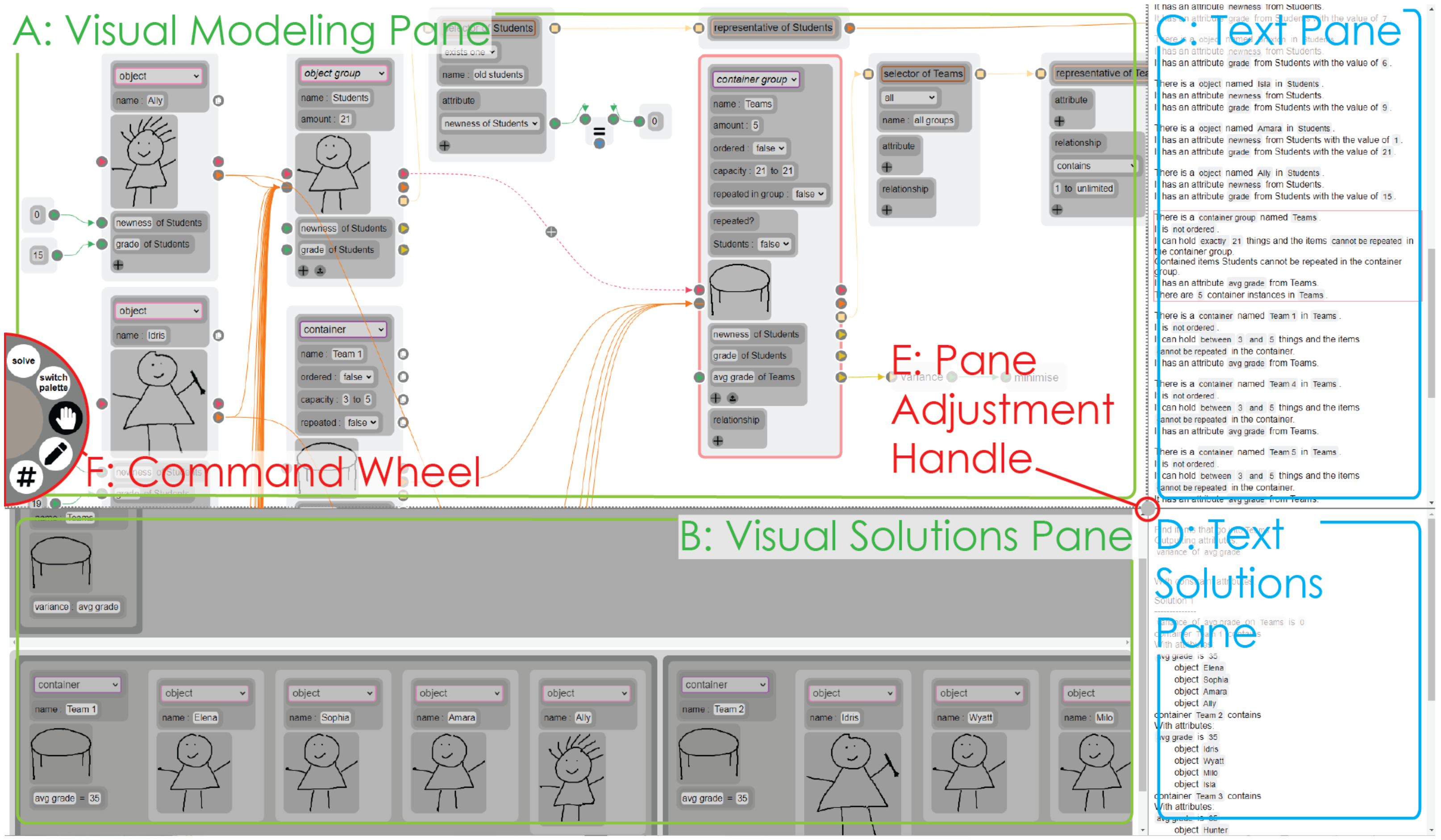
5.1. General Structure

Solvi supports interaction both through touch and through cursor to enable interaction from tablets and from PCs/Laptops (i.e., it is designed to support situated use— Design Goal O3). The main interface has four key PANES (Figure 1.A to D) that we will discuss in their own subsections below.

Visual representations can be screen real estate-hungry. To enable cross-notation interaction (DP4) and avoid having to rely too much on memory (part of O4), a PANE ADJUSTMENT HANDLE (Figure 1.E) allows modelers to quickly change the proportion of the screen devoted to each pane by simply dragging the panes' cross point. We anticipated that this would also better support different stages of the process (i.e., more space on the visual modeling pane at the beginning, and larger area for the solutions at the end—see also the Video Figure[3]).

A Command Wheel contains the different modes of interaction (e.g., drawing, solving) as well as different operators (e.g. equals, minimise, min, etc.) that can be added to the panes. The drawing mode creates the object, which can then be modified to become other types of items. The other modes, when selected, allow placing the different operators on the canvas. The wheel rotates to reveal the different available modes, operators, and tools. The list of modes is fixed and cannot be defined by the user. The wheel is designed so that the thumb can rest in its middle when used without a table or support (O3), and so that it can still be scrolled and elements selected by shifting the hold position of the thumb (O4). This is intended to free up the other hand for other interactions such as placing new objects on the panes. Alternatively, one can switch palette to use a more traditional toolbar as well, especially when using a mouse.

[3]`https://solvi.org.uk/Solvi.mp4`



It has an attribute newness from Students.
It has an attribute grade from Students with the value of 7.

There is a object named Braxton in Students.
It has an attribute newness from Students.
It has an attribute grade from Students with the value of 6.

There is a object named Isla in Students.
It has an attribute newness from Students.
It has an attribute grade from Students with the value of 9.

There is a object named Amara in Students.
It has an attribute newness from Students with the value of 1.
It has an attribute grade from Students with the value of 21.

There is a object named Ally in Students.
It has an attribute newness from Students.
It has an attribute grade from Students with the value of 15.

There is a container group named Teams.
It is not ordered.
It can hold exactly 21 things and the items cannot be repeated in the container group.
Contained items Students cannot be repeated in the container group.
It has an attribute avg grade from Teams.
There are 5 container instances in Teams.

There is a container named Team 1 in Teams.
It is not ordered.
It can hold between 3 and 5 things and the items cannot be repeated in the container.
It has an attribute avg grade from Teams.

There is a container named Team 4 in Teams.
It is not ordered.
It can hold between 3 and 5 things and the items cannot be repeated in the container.
It has an attribute avg grade from Teams.

There is a container named Team 5 in Teams.
It is not ordered.
It can hold between 3 and 5 things and the items cannot be repeated in the container.
It has an attribute avg grade from Teams.

Find items that go into Teams.
Outputting attributes:
variance of avg grade

With constant attributes:

Solution 1
----
variance of avg grade on Teams is 0
container Team 1 contains
With attributes:
avg grade is 35
    object Elena
    object Sophia
    object Amara
    object Ally
container Team 2 contains
With attributes:
avg grade is 35
    object Idris
    object Wyatt
    object Milo
    object Isla
container Team 3 contains
With attributes:
avg grade is 35
    object Hunter

Figure 1: Main structure of Solvi interface and top-level interface elements.

5.2. Visual Modeling Pane and Solvi Visual Language

The top left pane of Solvi (Figure 1.A) is an infinite canvas where the user builds the graphical model of the problem using the Solvi visual language. The main operations on the canvas are to create new items, to edit or delete existing items and to connect items. Since the operation of the canvas is tightly interwoven with the design of the visual language, we describe both simultaneously through the scenario where Taylor starts modeling the problem from Section 2.

One key element of Taylor's problem are the students themselves. To represent a student, Taylor uses the command wheel in the SKETCH MODE (Figure 2.A) and makes a quick sketch in the shape of a person, and adds some spiky hair, which is one of student Ally's trademarks (sketching the appearance of the object supports DP5). That creates a widget representing an *object*, which can be given a specific name (e.g., Ally—Figure 2.B). Then, by tapping the plus button at the bottom of the widget and renaming the placeholder text, Taylor creates some *attributes* relevant to the problem, such as their grade, and whether they are new or not (Figure 2.C). For Ally, those values can then be directly edited by creating a *value* and connecting it to the left port of the corresponding attribute (Figure 2.D).

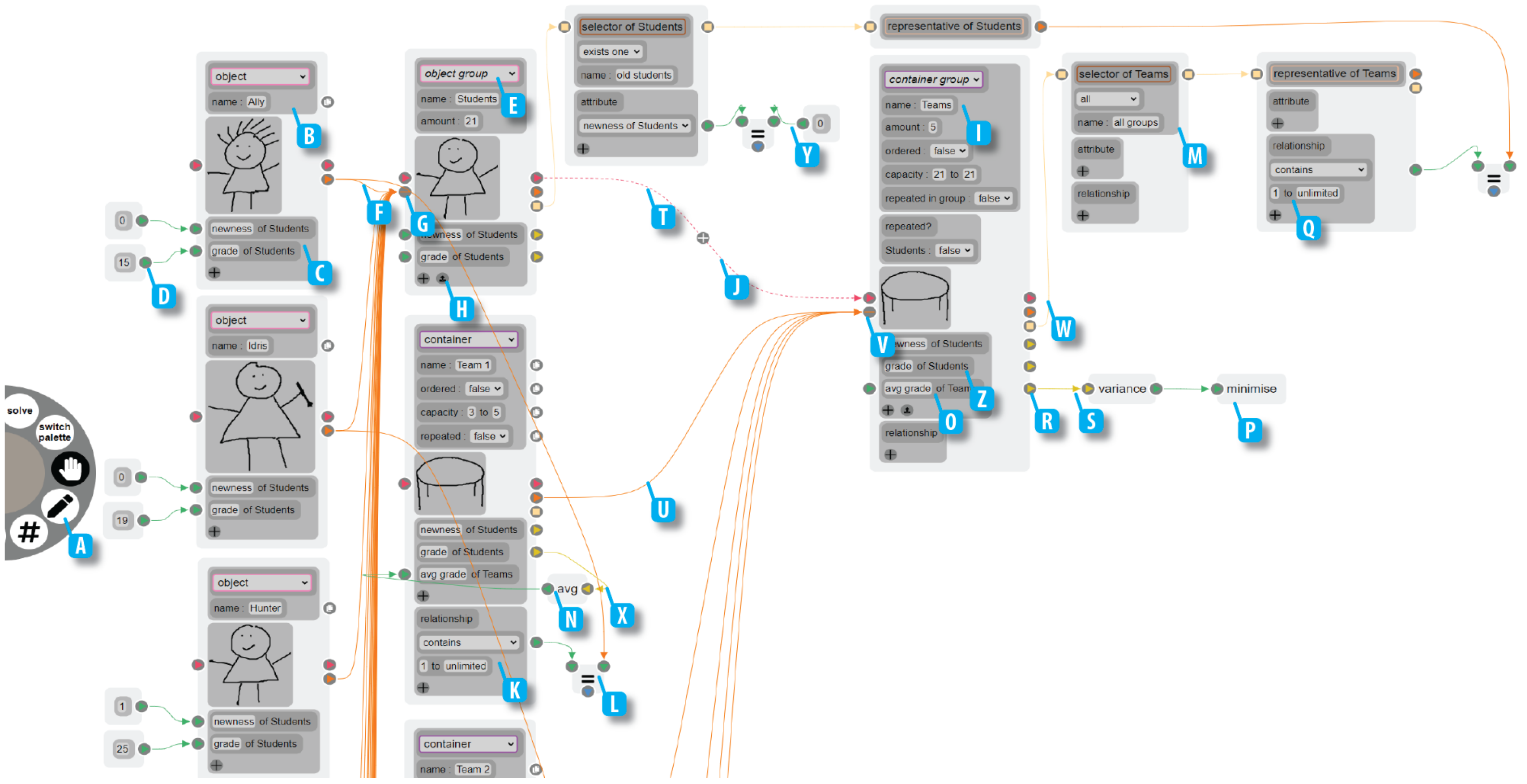


Figure 2: The visual specification of the example problem in the visual pane. Refer to the text for letter references.

Taylor would need then to do the same for all other students, but instead creates an *object group*, which is a more abstract item meant to represent collectives of objects. To create an *object group* widget, Taylor first creates a regular *object* widget, and then changes its type to *object group* (Figure 2.E). Taylor changes the "amount" attribute and then expands the instances to automatically create the rest of the objects. These attributes above the sketch in the widget are *type attributes* and are specific to the type that is selected. Because Taylor wants all 21 student objects to share the same attributes, Taylor connects Ally's object to the "Students" object group (Figure 2.F), which automatically creates attributes in any other objects belonging to the *object group.* Two key features here are that the individual student widgets can be all hidden through a button (Figure 2.G) to avoid cluttering the interface (O4), and that the values of the attributes of each object can also be filled by uploading a simple file with comma separated values, by pressing a button in the *object group* widget (Figure 2.H). Having two mechanisms to create multiple objects (a common feature in almost all constraint problems) supports DP6 because users can build groups from all their constituents or by defining an abstract *group*, or a hybrid of the two. Notice also that this design is also compatible with DP3, since it does not require that existing objects are connected to anything or be consistent while the model is being built.

The crux of Taylor's problem is distributing students into teams. A team is best represented as *container* in the Solvi visual language. This is a common construct that enables modeling a large variety of constraint problems (O1). If Taylor had to select a single team out of the class of students (e.g., for a regional contest), creating a single *container* would be sufficient, but the problem requires multiple teams, for which Taylor must create a *container group.* A widget for the *container group* item is created in the same way as an *object group* (sketch, then switch item *type*) and also supports bottom-up creation (DP6). The resulting widget (Figure 2.I) is named "Teams" and has a sketch of a classroom table in it (teams usually work together in a table in Taylor's class). The widget receives a *put into* link from the Students *object group*, which indicates that teams are made of students (DP2—Figure 2.J). The widget also allows Taylor to specify: a) how many groups to make (5, *amount: 5*); b) how many of the students have to be placed in any group (21, *capacity: 21 to 21*); and, c) whether students can be in multiple groups (no, *repeated in group: false* and *repeated? students: false*). This problem now uses all four possible types: object, object group, container, and container group. It is not possible to create any new types, but using a combination of the four types, it is possible to model a wide variety of problems.

At this point the model will be solvable and would produce all possible combinations of five groups between three and five students in size. This might be sufficient in situations where there are not many solutions and the user can simply select, by inspection, one that works (DP3). However, problems are often combinatorially complex, requiring explicit programming of the constraints. Here we will illustrate two ways of specifying constraints. The first is through *relationships*, which has its own section on the container widget (Figure 2.K). To make Idris and Ally be in a different team, one of the options is to force Idris to be in Team 1, and Ally in Team 2. This can be done by creating a *contains relationship* in the corresponding team that is then connected through an *equal operator* to the Ally object (Figure 2.L), and then the same for Idris in Team 2. Taylor creates the relationship by tapping the plus button in the relationship section of the widget and then selecting the required relationship type from the dropdown. This is a concrete way to specify simple constraints that support bottom-up processes (DP6). Objects can have any number of relationships, and relationships can be of more sophisticated types. For example, relationships can specify the *order* or *adjacency* of elements in a *container* (when the container is ordered—Teams in this example are not ordered).

The second, more sophisticated, type of constraint uses a *selector-representative* pair of objects. A *selector* essentially allows us to filter and get a subset of elements from any *object group* or *container group* based on one or more conditions. The *representative* allows the user to apply a constraint or relationship to each of the elements selected by the *selector*, and works in an identical way as a regular *object* or *container*, except it does not have a name because it represents each of the possible selected *objects* or *containers. Selector* and *representative* widgets are always created in pairs, by dragging from the *object group, container* or *container group.* In our example, Taylor uses one *selector-representative* pair to choose students who are not new (i.e., attribute $newness = 0$), and another to operate with each of the teams (selecting all the teams but no conditions on the selector—Figure 2.M). By linking the representative of students who are not new to a *contains relationship* of the representative of all teams, Taylor indicates that every team has to contain at least one student who is not new. The *selector-representative* pair of widgets could have been unified into a single widget because we did not find any modeling situation in which they would not go together, but we chose to split it into two widgets to create a more consistent parallelism with its textual representation (DP4—see Section 5.3), and to scaffold learning by novice modelers, which we anticipated would best understand these as two separate, but consecutive, functions (O4).

A final element to complete the example involves minimization. Taylor wants to balance out the average grade on each Team, which makes sense to give all teams a fair chance. To do this, a new attribute of a team *container* (average grade) is created. Its value is calculated through the *average operator*, from the attribute of the grade of students in the team container itself (Figure 2.N). This automatically propagates and creates a new attribute in the teams *container group* (Figure 2.O). The "avg grade of Teams" in the *container group* is then connected to the *variance* operator which, in turn, is connected to the *minimize* operator (Figure 2.P). This effectively communicates to the system that Taylor wants to minimize the variance among the grade averages of the groups.

For completeness we highlight two additional characteristics of the Solvi visual language and interface that are not obvious from the example above. First, many of the items, including *containers*, *container groups*, *selectors* and many *relationships* can be qualified with a cardinality (e.g., Figure 2.Q). This provides additional power to represent sophisticated models. For example, one could force the "at least one student who is not new" condition on a limited number of teams, or a range of them (between 1 and 3), instead of in all of them.

Second, the interface has a number of different ways to connect elements between them, which embody the syntax of the visual language. Links are enabled (and initiated) through PORTS (small circles with a symbol inside—e.g., Figure 2.R) from which LINKS (arrows—e.g., Figure 2.S) can be dragged to other ports in different items. There are 6 kinds of links. Red discontinuous arrows represent *put into* links (e.g., students are *put into* groups—Figure 2.T). Orange continuous arrows represent *is part of* links (e.g., Ally is part of the "students" *object group*—Figure 2.U—and team 1, 2, 3 and 4 are each connected to the "teams" *container group*). In the interface, *is part of* links can be compressed (and all the constituent items hidden) through a button in the destination port (Figure 2.V) to avoid cluttering the space (O4, DP5). Tan-colored solid arrows with square ports indicate *selected by* links (Figure 2.W—essentially connections to a *selector*, see explanation above). Finally, there are two types of links to connect objects and numeric values: green arrows connect individual values or objects to attributes or operators

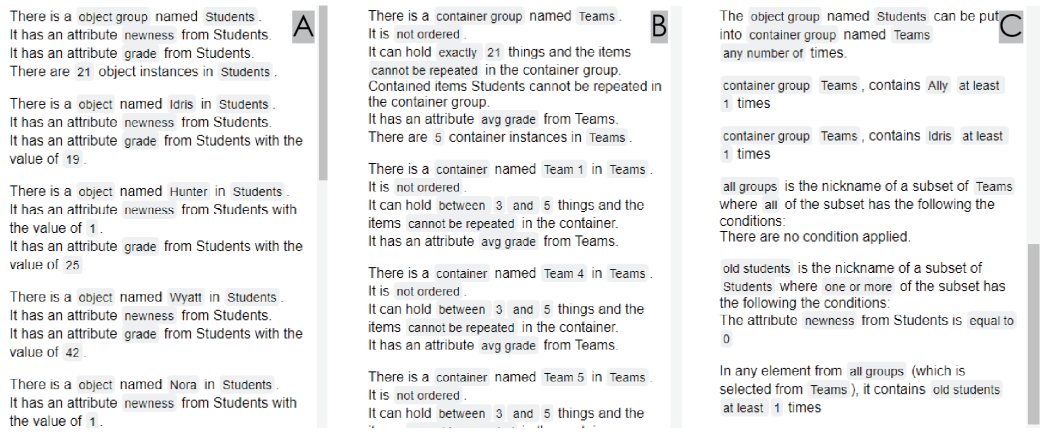
(Figure 2.Y), and yellow links contain lists of elements, usually to be aggregated by operators such as *average*, *sum*, or *variance* (Figure 2.X). The problem uses all the different major widget types that exist in the system.

The relatively large number of connection types is a source of complexity for modelers, but it is a deliberate choice based on Zhu et al.'s findings [4], who found that people use arrows and links to represent relationships of many different types that involve more than one construct. Additionally, there is a limited choice of visual idioms to represent this kind of connections (this is discussed further in Section 8).

5.3. Text Pane

This pane is the main mechanism by which we support DP4. Zhu et al. noted that people expressing constraint problems often require alternative types of notations to express a problem [4], and problem solving in other domains (e.g., physics problems [44]) often depend on accurate transformation of verbal information into diagrams (see also [47, Chapter 2]). This pane (shown in Figure 1.C) provides a relatively simple translation of the contents in the VISUAL MODELING PANE to English text. The main purpose is to allow modelers to check that what they have represented in the diagram corresponds with what they intended, in a different form closer to the problem formulations that they hold in their mind. This can help ameliorate some of the issues of visual notations, which sometimes require memorization of arbitrary graphical symbols and signified constructs. For example, a novice modeler might not be very clear about the meaning of the selector-representative constructs, but after trying them out in the visual modeling pane, the text pane shows "In any element from all groups, it contains old students at least 1 times". Figure 3 shows a few of the sentences that Solvi constructs for the English textual description of the model.

In its current version, the text pane does not allow direct input of text that would be automatically translated in its visual form. However, there are two features that further facilitate multi-pane interactions across the two notations: a) when a modeler selects an item on the text pane, the visual modeling pane highlights that item and can pan to show the object if it is currently out of view, and b) item types, names and numeric attributes can be changed in the text pane.


A

There is a object group named Students.
It has an attribute newness from Students.
It has an attribute grade from Students.
There are 21 object instances in Students.

There is a object named Idris in Students.
It has an attribute newness from Students.
It has an attribute grade from Students with the value of 19.

There is a object named Hunter in Students.
It has an attribute newness from Students with the value of 1.
It has an attribute grade from Students with the value of 25.

There is a object named Wyatt in Students.
It has an attribute newness from Students.
It has an attribute grade from Students with the value of 42.

There is a object named Nora in Students.
It has an attribute newness from Students with the value of 1.

B

There is a container group named Teams.
It is not ordered.
It can hold exactly 21 things and the items cannot be repeated in the container group.
Contained items Students cannot be repeated in the container group.
It has an attribute avg grade from Teams.
There are 5 container instances in Teams.

There is a container named Team 1 in Teams.
It is not ordered.
It can hold between 3 and 5 things and the items cannot be repeated in the container.
It has an attribute avg grade from Teams.

There is a container named Team 4 in Teams.
It is not ordered.
It can hold between 3 and 5 things and the items cannot be repeated in the container.
It has an attribute avg grade from Teams.

There is a container named Team 5 in Teams.
It is not ordered.
It can hold between 3 and 5 things and the

C

The object group named Students can be put into container group named Teams any number of times.

container group Teams, contains Ally at least 1 times

container group Teams, contains Idris at least 1 times

all groups is the nickname of a subset of Teams where all of the subset has the following the conditions:
There are no condition applied.

old students is the nickname of a subset of Students where one or more of the subset has the following the conditions:
The attribute newness from Students is equal to 0

In any element from all groups (which is selected from Teams), it contains old students at least 1 times

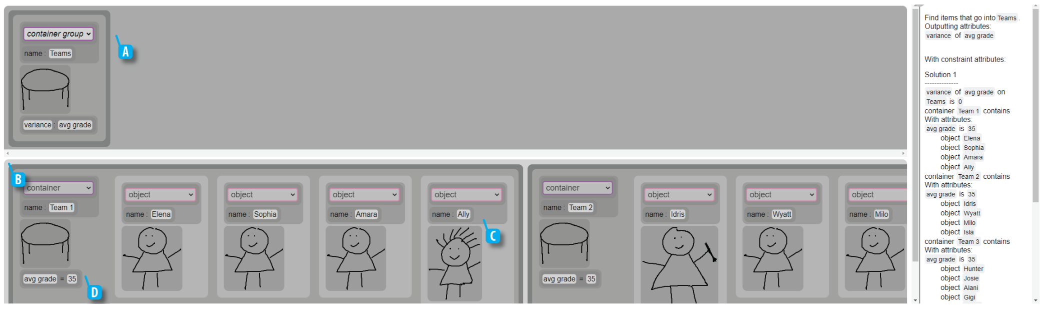
Figure 3: The textual specification of the problem in the text pane. *A* showing the Student descriptions, *B* showing the Teams descriptions, and *C* showing description of the constraints

5.4. Visual and textual Solutions Panes

The bottom left pane (Figure 1.B) is dedicated to visually displaying *solutions* (i.e., combinations of object states that comply with the imposed constraints—O2). Computation of solutions is activated by selecting the Solve mode in the Command Wheel and dragging the item for which we need the solutions from the Visual Modeling Pane. In our example, this is the "Teams" container group, but in other problems there might be several items that can be solved independently (DP6). This feature can be invoked at any time (DP3).

The initial default of dragging a *container* or *container group* to the Solution Pane will display the calculated solutions using a *containment* visual idiom. Each row represents a solution, which will show the specific sub-containers, sub-sub-containers, and so on. By default the solutions will not display any attributes. The interface enables dragging and dropping of specific attributes from items in the Visual Modeling Pane into the Template Solution in the visual solutions pane (a special place holder representing a prototype of the solutions, which appears on top). This forces every solution to display the values of that attribute.

In more concrete terms, Figure 4, left side, shows the solution pane for our example problem. The second row, which shows the first solution (Figure 4.B) is enclosed by a grey rectangular area that contains a further rectangular area for each of the teams with the team sketch (a table) which, in turn, contain rectangles with the sketch of each person (e.g., Figure 4.C is Ally). Notice that the representation of each team in the solution also indicates the average grade (e.g., Figure 4.D). This is because Taylor dragged the “average grade of team” from the teams containers of the VISUAL MODELING PANE into the VISUAL SOLUTION TEMPLATE (Figure 4.A). The pane is scrollable horizontally so that one can see all the components of the solution, even if it does not fit in the current size of the pane, and vertically so that all the available solutions are visible.



```
Find items that go into Teams.
Outputting attributes:
variance of avg grade


With constraint attributes:

Solution 1
--------------
variance of avg grade on
Teams is 0
container Team 1 contains
With attributes:
avg grade is 35
      object Elena
      object Sophia
      object Amara
      object Ally
container Team 2 contains
With attributes:
avg grade is 35
      object Idris
      object Wyatt
      object Milo
      object Isla
container Team 3 contains
With attributes:
avg grade is 35
      object Hunter
      object Josie
      object Alani
      object Gigi
```

Figure 4: The solution panes for the main example. Visual on the left, text on the right.

Displaying the sketches takes a significant amount of space, but also provides the modeler a visual overview of the solution using the same symbols (sketches) that they used for the modeling phase. This represents further support for DP5 and might reduce ambiguity and facilitate recognition by rendering solutions in a way that resembles an Isotype diagram [48].

The corresponding TEXT SOLUTION PANE represents the solutions in text form. Analogically to how the TEXT PANE supports the VISUAL MODELING PANE, the TEXT SOLUTION PANE supports disambiguation of the solutions of the VISUAL SOLUTIONS PANE, but also serves as a useful output format (e.g., to copy-paste into an e-mail).

5.5. *Other Features*

The design of the interface is completed with some necessary features including: save/load functions, undo/redo, a system to show messages to the user (e.g., when an invalid cycle is discovered in the created model), zooming out of the canvas for an overview of the problem, and a way to upload bulk attribute values to a group through a CSV file.

Finally, during the design we noticed that there are cases in which the nested structures of *container groups*, *containers*, *object groups* and *objects* can make it difficult to assess which item an attribute belongs to (e.g., students have a grade, but teams have collections of student grades, and the group of teams can itself have a collection of grades, all of which could be operated with. To address this problem we chose to label attributes with their original items when these are inherited from contained or subgroup types. For example, in Figure 2.Z the grade of Students attribute in the Teams container group refers to the set of values of contained Students. This applies also to the text-based panes.

6. Design Methodology and Implementation

The current prototype of Solvi was designed and implemented based on previous studies of people's understanding of constraint problem representation [4] and of the constraint problem solving process [3]. The design process iteratively developed the Solvi language based on the objectives and design principles, while interleaving design of visuals and interaction techniques with implementation. This allowed us to adapt the language, the interface, and the software infrastructure to each other, and to make sure that the generated code for the solver would be compatible with the designed behaviour. A key technique during the process was the generation of a library of canonical problem examples in plain English which allowed us to incrementally build the language. When the design evolved, we could check whether previously addressed problems were still modelable and whether the constructs designed thus far were still necessary as well as compatible and consistent with the new introduced elements and constructs. We also regularly checked the compatibility of the design with the principles and objectives that we had selected a priori, and informally tested the different versions of the prototypes on ourselves and others.

The Solvi language provides an interface to the Essence [9] textual constraint programming language which, in turn, compiles models of the Conjure modeling system [49] with the Athanor Local Search Solver [50]. Although Essence was itself the product of an effort to make constraint programming more accessible, the Solvi language is far from a one-to-one translation of the elements of Essence. A key technical challenge was generating valid Essence from the specification and behavior of the Solvi language. The validation of this process was done by hand and is detailed in the Supplementary Materials

[4].

The implementation is based on a client-server architecture supported by the Meteor.js library [5], so that heavy computing, especially the running of the constraints solver, can be offloaded to a powerful server. The client side is built on React[6] and SVG.js [7]. The server is implemented in Node.js [8]. The system is available to use at `https://solvi.org.uk/`. The source code will be available open source on GitHub when the paper is published and is currently included in the supplementary materials.

7. Solvi: Evaluation

In this section we evaluate Solvi. Through our evaluation we aim to validate our design goals (Section 4), identify potential weaknesses and use our findings as guidance for future improvements.

7.1. Evaluation Design Rationale

There are many questions about a tool such as Solvi that are best addressed through empirical evaluation methods such as observations, interviews, and laboratory studies. We've identified five central questions concerning Solvi: a) is there an actual need for such kind of system; b) will such need be recognized by potential users?; c) will people using such a system be able to leverage its interface to address their actual needs; d) to what extent? and e) with which level of performance? Although all these questions are relevant to most systems at some point, finding answers to some is more relevant at certain stages of their adoption.

At this stage of Solvi's development, questions a), b), and c) are most pertinent since we have found no previous work answering them in the context of systems for solving constraint problems, and they evaluate the approach's potential, significance, and feasibility. Question d) addresses important issues related to the coverage (power, expressiveness) of the language and interface (e.g., what percentage of problems encountered by people can Solvi cover?), and e) is mostly about the basic usability of the system and aspects

[4]`https://solvi.org.uk/Solvi-supplementary.zip`

[5]`https://www.meteor.com/`

[6]`https://reactjs.org/`

[7]`https://svgjs.dev/`

[8]`https://nodejs.org/`

that would make the system more useful. Questions of type d) and e) are important, but we consider them somewhat premature in this area. Moreover, answering these questions often demands methodologies that rely on participant pools that do not exist today, given the limited number of people familiar with constraint problem-solving tools.

Therefore, our evaluation centers on understanding people's problem identification skills, the perceived value of our solutions, and their capability (after basic training) to articulate problems using the Solvi language. We opted for a mixed-methods study. Initially, we interviewed participants about their experiences with constraint problems. Following that, we introduced them to a controlled environment with specific tasks, assessing their ability to understand and use the proposed system. Recognizing that prolonged sessions might fatigue participants and compromise data quality, we designed an evaluation within the time constraint of two hours. This required making hard choices on how many different problems it was reasonable to cover and how to split the time between more open questioning of participants and the different activities that they had to carry out. While various designs are possible, the experimental design described below represents our best attempt at answering the most urgent subset of questions a-d above within a reasonable scope. More specifically, the purpose of the empirical evaluation is: 1) to validate the motivation of our work (do people encounter constraint problems that they would like to solve in a better way?); 2) to assess people's ability to identify and model problems (how well can people express constraint-based problems?); 3) to assess the design of the Solvi language (to what extent can participants express their problem in the Solvi language? Which constructs are hardest to understand?), and; 4) to assess the main components of the Solvi User Interface (what UI components present problems or need improvement? What UI components offered clear benefits?).

7.2. Participants and Procedure

Our evaluation design exposed participants to the Solvi system for individual sessions of approximately two hours per participant. We recruited 12 participants (8 female, 4 male, age 18 to 34) into two expertise groups (6 participants each): people with a Computer Science background (but without explicit constraint programming expertise—the CS group), and people without (the non-CS, or novice group). We chose to expose people with different expertise to Solvi to see whether lack of programming abilities would present specific barriers to the novice group. The experimental design and protocol was approved in advance by the local Research Ethics Board. Participants received a £20 online voucher or equivalent in their local currency as compensation for their time.

Participation was remote over the Microsoft Teams platform. Participants used their own computers, and were encouraged to use a tablet, pen or touch device when possible. The web-based Solvi system was accessible directly to their devices through a web URL.

After providing consent, participants underwent the experiment's six phases as follows: 1) introduction and constraint problem examples (video of a real-world situation that can be modeled as a constraint problem, and description of two example constraint problems: a knapsack filling problem[9] and the wedding table problem[10]); 2) participants reflected on constraint-based problems of their own; 3) the experimenter demonstrated the operation and features of Solvi (30 minutes); 4) participants used Solvi to model a wedding table problem; 5) participants used Solvi to model one of their own problems; 6) participants filled a NASA TLX questionnaire about the tasks, and; 7) the experimenter conducted a semi-structured interview about the constructs of the language and the general usability of the tool.

During phases 4 and 5 (participant using Solvi to model) the experimenter provided two types of guidance. If a participant did not remember a particular feature or completed a part of the representation incorrectly, the experimenter highlighted what they did wrong. If they still could not progress, they received guidance on the next step to fix the mistake. The experimenter carefully recorded how much guidance each participant needed, which is part of the analysis. This process was necessary to ensure that all participants reached as far as possible within the modeling process. Further details, including the problem descriptions and example problem model (modeled by the researchers) are in the supplementary materials.

7.3. Measurements and Analysis Methodology

The inputs to the analysis were: data from the video of the participants' progress through modeling the problem; the final representation that participants achieved in the system (including the level of help provided by the experimenter); the NASA TLX questionnaire answers (a 20-point scale), and

[9]`https://www.csplib.org/Problems/prob133/`

[10]`https://github.com/RishabhTyagiHub/Constraint-Satisfaction-Problem---Wedding-Seating-Arrangement`

the semi-structured interview answers. Most of the analysis (all except the NASA TLX) is qualitative in nature. We chose a hybrid analysis procedure. On one side we follow a straightforward thematic analysis focused on topics of interest determined *a priori* from the purposes listed at the top of this Evaluation Section. On the other side, we wanted to discover issues, topics and themes that we had not considered a priori, for which we carried out a multi-pass analysis procedure reminiscent of grounded theory [51], with three passes over the data: identification of constructs of interest, codebook construction and coding, and topics aggregation through affinity diagramming. The NASA TLX was analyzed for differences between the novice and CS groups through a Mann-Whitney U test [52].

7.4. Results

We discuss the key results in the order of the goals listed above. Items 2 and 3 are tightly intertwined and hence discussed together.

Validation Motivation. All participants were able to provide examples of their daily life that they thought they would benefit from if modeled and solved computationally. Participants mentioned a variety of valid constraint problems, from furniture arrangement to organizing COVID-safe bonfires. Several participants highlighted in the interview the potential value to them of finding better solutions, finding them faster, or not having to find solutions by hand. For example, P8 said in the interview regarding the applicability of the system to real life problems: "Yeah, for sure. And I could imagine like I was explaining where there is like 400 something classes [being scheduled], it would become a lot more useful.". P9 said, "I think if there were anything that was related to trying to organize something that has a lot of variables or considerations, it would be useful to organize it out, both visually for people to see where things are related to each other as well as the possible outcomes."

Expressing and Modeling Problems (with the Solvi language). Participants completed between 35% and 98% of the modeling task of Phase 4 (the wedding table problem) during the allocated time. Participants managed to complete the majority of the modeling task, with around 66% of the participants completing 70% or more of the wedding table task within the given time. However, all participants had some guidance. A large proportion of the required guidance was related to elements on the interface, rather than the constructs of the language. Around two thirds of the participants modeled a reasonably easy problem in Phase 5 with up to two constraints, and they fully completed the task. The last third of the participants tried modeling a hard problem which required the use of the selector-representative and many constraints. However, they were not successful in finishing the problem in the given time.

As expected, participants from the non-CS group had more difficulty with modeling. One notable issue was the use of terminology; although we carefully considered the nomenclature to avoid technical terms as much as possible, Solvi still required naming the main constructs (e.g., *container*, *container group*, *attribute*). Participants, especially from the non-CS group, had difficulty remembering those terms. More generically, non-CS participants also had problems translating the problem specification to the Solvi language, in a similar way to how novice programmers face the challenge of computational thinking (e.g., [53]). P7 mentioned that they were "not really used to thinking in this way".

These intrinsic difficulties are confirmed by the analysis of the TLX data, which shows that a majority of non-CS participants found the task mentally taxing and, to a lesser extent, frustrating (see Table 1). The CS group attributed significantly less mental load to the problem, as well as less frustration. This clear divide suggests that the system relies, at least at this initial level of training, on the programming and/or logical background of the participants. Interestingly, when modeling their own problem right after, the differences between the two groups are not as marked (or statistically significant—see Table 2). This might be because participants' own problems are less demanding, their choice of problem is less demanding, or because they learned from the previous task.

As expected, some constructs were perceived as more difficult to use, as well as creating more trouble for the participants. The two main challenges encountered by participants were in the *selector-representative* construct. Participants 3, 4, 5, 7, and 8 were confused with how to set the relationship constraint on the representative. Additionally, some participants had misconceptions about the use of the selector. For instance, one non-CS participant connected the attribute on the selector to the right side of an *equal to* constraint on a container. 10 out of the 12 participants had issues applying either the filter or the constraint. 6 out of the 12 participants used the quantifier selector-representative incorrectly as part of the model representation. Participants often confused the difference between"*exists one*", "*exists*" and

Wedding Table		Score 0-4	5-8	9-12	13-18	17-20	Mean	Median	p-value	MWU
Mental	**Non-CS**	0	1	0	1	4	16.2	17.5	0.041	5.000
	CS	0	2	3	1	0	10.0	11.0		
Physical	**Non-CS**	4	2	0	0	0	2.5	2.0	1.000	18.500
	CS	4	0	1	1	0	4.5	1.5		
Temporal	**Non-CS**	1	1	3	1	0	8.8	10.0	0.240	10.000
	CS	1	4	1	0	0	5.8	5.5		
Performance	**Non-CS**	0	0	2	2	2	14.5	14.0	0.026	4.500
	CS	1	2	2	1	0	8.5	8.5		
Effort	**Non-CS**	0	0	2	2	2	15.3	15.5	0.065	6.500
	CS	0	2	1	3	0	10.7	11.5		
Frustration	**Non-CS**	0	1	1	2	2	14.0	14.0	0.026	4.500
	CS	2	2	1	1	0	6.8	7.5		

Table 1: NASA TLX answers and statistical comparison of CS vs. Non-CS participants for the wedding table modeling task. Responses are on a 20-point scale and grouped into bins of size 4. MWU stands for Mann-Whitney U. The Mann-Whitney U test is a non-parametric test for hypothesis testing. [52] The calculated p-value from the Mann-Whitney U test shows the significance of the difference between the Non-CS and CS cohorts. A p-value less than 0.05 is considered significant.

Own Problem		Score 0-4	5-8	9-12	13-18	17-20	Mean	Median	p-value	MWU
Mental	**Non-CS**	0	1	4	1	0	9.8	10.0	0.310	11.000
	CS	3	1	0	2	0	6.8	5.0		
Physical	**Non-CS**	6	0	0	0	0	1.3	1.5	0.818	19.500
	CS	4	2	0	0	0	2.5	1.5		
Temporal	**Non-CS**	1	1	2	2	0	10.0	9.5	0.132	8.500
	CS	2	3	0	1	0	5.7	5.0		
Performance	**Non-CS**	3	0	2	1	0	7.0	6.5	0.818	16.500
	CS	1	4	1	0	0	5.7	6.0		
Effort	**Non-CS**	0	2	3	1	0	9.3	9.5	0.349	12.000
	CS	1	3	1	1	0	7.5	7.0		
Frustration	**Non-CS**	0	4	2	0	0	8.2	8.0	0.065	6.500
	CS	2	3	1	0	0	4.8	5.0		

Table 2: NASA TLX answers and statistical comparison of CS vs. Non-CS participants for their own problem modeling task. Responses are on a 20-point scale and grouped into bins of size 4. MWU stands for Mann-Whitney U. The Mann-Whitney U test is a non-parametric test for hypothesis testing. [52] The calculated p-value from the Mann-Whitney U test shows the significance of the difference between the Non-CS and CS cohorts. A p-value less than 0.05 is considered significant.

"*all*". All 12 of the participants experienced issues with these widgets and reported that it was their least understood part of the system.

Other issues include problems with the *ordered* attribute of containers (which was necessary to model the wedding problem because it enables constraints such as "adjacent to"). 9 out of the 12 participants did not set *ordered* to true initially on the container group. This omission meant that people sitting around a table *container* did not have a sequence, and hence missed the ability to use positional *relationship constraints*. This reveals that participants did not understand the meaning of *ordered*, and only became aware of the problem after trying and failing to locate *next to* in the *representative of tables* widget element.

6 out of the 12 participants also confused the capacity property on the container group and the capacity on the individual container instances. They thought the capacity on container group was the capacity on the individual containers as opposed to the total capacity of all the containers within the group.

Solvi UI's Components. As intended, the sessions uncovered multiple issues of the interface, but also highlighted some of the perceived benefit of specific features. On the positive side, 8 participants appreciated the OBJECT SKETCH feature, which they found fun or engaging (e.g., P7: "Because you can draw things ... that's quite fun for me"), and useful to understand the representations in the UI (e.g., of the solution—P9 "I do like that when it gives me the solution, I can see it both visually and textually."). 3 of 12 participants also explicitly used the text pane to confirm or understand what they had done. P1 mentioned that "So, even if I got a little bit confused ... even if had any questions, I could just take a look here [the text pane] and it was very detailed information." Although we expected to observe more use of the text panes, several participants highlighted that they focused on the modeling first, and did not have sufficient time to check the text in the TEXT PANE.

On the negative side, the COMMAND WHEEL attracted much negative attention because it impeded participant's ability to remember and easily access important commands. Although this is likely a crucial issue of the interface design, it might have been aggravated by the wheel not having any obvious benefit to someone using the system while sitting and not on a touch-enabled device. Another key issue is that 10 participants made mistakes in the direction of the ARROWS created, and had difficulty remembering which

PORTS to use. Other problems were more circumstantial: the remote testing platform sometimes made the interaction slow (especially on slow machines, when the participant did have a greater delay connection to the server, and when the number of elements on screen grew), and the testing system at the time was occasionally slow.

Several of the issues raised by the participants were taken into account in later iterations of the improved implementation which can be seen in the Figures containing screenshots of the interface and the Video Figure[11].

8. Discussion

The subsections below interpret the empirical results, address general challenges of interfaces for CP, report what we learnt from the current design, and highlight limitations of the work.

8.1. Interpretations of Empirical Evaluation Results

The results from the evaluation are mixed, but also encouraging and useful. Participants were highly supportive of the goal and readily able to identify situations in which they would benefit from this kind of system. This is indicative of the potential impact of making constraint problem solving technologies more broadly approachable.

The results show that people without computer science backgrounds can achieve some degree of modeling, although they struggle with the most sophisticated constructs (e.g., the *selector-representative*), and with logical and set-theory statements (e.g., all, any, exists), especially when they are compounded in separate parts of the model. We believe that this reflects, for the most part, the inherent required effort, knowledge and background of mastering mathematical, logical and programming constructs. Nevertheless, interpreting the results in our initial study should consider the limitations of the training (30 minutes of instruction), and the general lack of familiarity of participants with formal problem expression and constraint modeling. Participants confirmed in the interview that, after modeling, they felt more familiar with the concepts and showed confidence that they would get better with more use.

[11]`https://solvi.org.uk/Solvi.mp4`

The data collected also points to issues in the design of the Solvi language and interface. The simplest have already been addressed in the current version. For example, Solvi provides on demand an alternative to the Command Wheel that sits along the left edge of the screen and does not hide any modes or tools. There are also improvements in the general performance of the tool and on the generation of natural language for the Text Pane, and an additional feature that allows Solvi users to change object names and item types from the text pane. Some other issues are more complex and might require more radical solutions; for example, the reported difficulties due to the multiple types of relationships between items, and hence the use of multiple types of links could require additional UI techniques, implementation of system inferences, or a deeper rethinking of the interface.

8.2. Solvi Design Discussion

Solvi is the first visual language and tool for modeling constraint problems for non CP experts. Problem modeling is essentially a type of declarative programming and one of the reasons of the lack of attempts to make CP accessible might be the inherent difficulty of end-user programming. It would be naïve to assume that an interface, regardless of how clever or carefully designed, can eliminate the challenges of programming, a point already made by Nardi in 1993 [32]. Nevertheless, end-user programming for CP seems quite suitable to a visual interface-assisted approach such as ours because it lacks some of the complexity of other types of programming (e.g., procedural programming requires understanding of control flows). Moreover, recent advances on visual programming, understanding of problem specification, and new tools and algorithms (e.g., Machine Learning-Driven Natural Language Processing and generation such as [54, 55]) might make it worth for more people to learn a new skill, language or interface. A UI can also become a successful conduit to teach people about how to formalize their problems. It might be tempting to believe that end-user programming is too difficult and, hence, a dead-end, but we know that there are extremely successful forms of end-user programming, such as spreadsheets and (for a narrower audience) tools such as MAX-MSP and Scratch [27]. We see Solvi as a step towards this vision. It is, nonetheless, a hard problem. Modeling constraint problems might never fit a “walk-up-and-use scenario”.

One of the most delicate parts of our process was to find a good balance between the language’s expressivity and simplicity of use. Solvi tries to strike a balance, applying lessons learnt from Zhu et al.’s study of how people graphically represent problems [4], without simplifying the range of representative problems so much that the tool would become useless. Other approaches are possible which might be complementary. For example, Solvi could include a searchable library of pre-modeled problems and commonly used constructs ready for people to adapt to their own circumstances or to add to their models. Interestingly, this points to an additional potential benefit of Solvi and future tools in it class: producing visual representations of problems that are formal but also easily glanceable might help different stakeholders communicate with each other about problems. Providing pre-made sections of commonly used constructs that people could use as part of their model.

A related design tradeoff between preciseness and familiarity of the naming of terms became also obvious. Using technical definitions of terms such as *object*, *class* or *collection* simplifies linkage to general programming constructs but forces people to learn definitions that might overlap or interfere with more familiar meanings of the same word. Additionally, some of these constructs do not match the CP context exactly (e.g., *object groups* are similar to *classes* in object-oriented programming parlance, but it is not a perfect fit). An interesting prospective solution would be to enable a kind of "term localisation" which adapts nomenclature to the circumstances and expertise of the user.

Our interface included several UI design innovations which we consider secondary contributions of this work. The main one is the dual graphical-textual interface for both specification and solution display that was derived from Zhu et al.'s findings [4], which is also theoretically supported by principles of multi-media instruction [56]. This element is reminiscent of coordinated views [57] and brushing and linking [58]. For example, work by Zhi et. al. [59] looked at using linking between text and visulisation to enhance storytelling. Simultaneously, some applications use visual/UI representations and textual programming in simultaneously visible screens (e.g., Wrangler for data cleaning and transformation [60], and Anteater for programming and debugging with visualization [61]), but we are not aware of any visual programming systems that offer parallel natural language support to make sense of the visual semantics. Binks et al. [62] prototyped a system with parallel natural text and visual representations or ideas, but it does not involve formal specification and is meant for a very different purpose (support argumentative essay writing). Other relevant work includes Kizil et al.'s work [63], who investigated how formal validation of mathematics elements within a mostly natural text document can be validated automatically to assist the user, and Carter et al.'s work, which uses Machine Learning and NLP techniques to interpret textual descriptions of constraint problems. These approaches are complementary to ours and could be integrated as an enhanced way to connect the TEXT PANE with the VISUAL MODELING PANE.

Other UI innovations of our system are the simultaneous visual representation of the problem model and the solutions in coordinated and linked manner. This includes the use of the same visual idioms in both panes, the specification of items to solve by dragging from one pane to another, and the use of a configurable template solution to indicate which attributes and elements to display in the solution. We also designed an agile mechanism to quickly reallocate screen real estate to different representations in 2 dimensions simultaneously that we believe will be particularly useful for problem specification due to the requirement that several representations appear simultaneously on screen, as well as a type of menu (the COMMAND WHEEL) that is designed to maximize the use of multi-hand interaction while holding a tablet. However, the value and effectiveness of these two UI mechanisms need to be validated in independent studies; in fact, the wheel showed to be undesirable for non-tablet setups, which is why we allow to switch to a more standard linear menu.

8.3. Limitations and Future Work

We acknowledge that the study provides only an initial evaluation of Solvi. Section 7.1 introduced a general framework that can help plan the next steps in the evaluation of the tool now and as it progresses through the next stages of maturity. Specifically, next steps to evaluate the tool include: a) an extension of the problem types tested (possibly through an additional controlled task-oriented study); b) an in-the-wild, longitudinal study that could assess the match of the tool with actual scenarios of use of real users (e.g., similar to [64]); c) an in-depth usability study of the features of Solvi, and specifically those which are novel such as the simultaneous presentation of diagrams/text and the problem/solution spaces; d) additional studies with specific populations such as high school learners or people without any advanced formal logical or mathematical education, to evaluate the potential breadth of use of the tool.

There is certainly much more work to be done if we want to make constraint-based problem solving accessible to everyone. Nevertheless, we believe that our system and results demonstrate how one can provide a lower threshold access point to CP for more people. Specifically, our approach does not require seating at a computer, installing software and learning how to compile and execute models in textual files. Our approach might also offer an intermediate step for knowledgeable but non-formally trained users, perhaps in the same way as spreadsheets offer access to data analysis and database functionality to non-experts.

Finally, Solvi's language and interface is quite expressive, but needs to be extended to be able to express some types of problems that can be currently modeled with textual CP languages. These classes of problems include two-way relationships and graphical problems with two-dimensional (or more-dimensional) data structures such as chess problems or room space arrangements.

9. Conclusion

This paper presents the design, implementation and initial evaluation of Solvi, a system to support modeling and solving (using existing solvers) of constraint problems for broader audiences. We contribute the design of a visual language that can express a wide range of problems and a supporting novel interface that enables visual referencing through user-generated sketches, a dual-representation style of interaction (node and link diagram and natural language), and simple visualization of the solutions based on visual containment.

The results of an initial evaluation show that participants are indeed holders of this kind of problems. Participants also achieved some degree of success with modeling of a pre-determined problem and one of their own problems with relatively short formal training. Having a computer science background mattered for modeling achievement and perceived mental load of the task. The study also highlighted the constructs that presented the most difficulty, which include the *selector-representative* and the different ways in which components of the model can relate to each other.

Acknowledgments

The authors would like to thank Daniel Zenkovitch for his input into the Solvi application development.

Current Funding Sources List Add a Funding Source Natural Sciences and Engineering Research Council of Canada Award Number: 2020-04401

— Recipient: Miguel A Nacenta Engineering and Physical Sciences Research Council Award Number: DTG1796157 — Recipient: Xu Zhu

References

[1] R. Barták, History of Constraint Programming, in: Wiley Encyclopedia of Operations Research and Management Science, American Cancer Society, 2011. doi:`https://doi.org/10.1002/9780470400531.eorms0382`.

[2] C. Jefferson, I. Miguel, B. Hnich, T. Walsh, I. P. Gent, CSPLib: A problem library for constraints, 1999. URL: `http://www.csplib.org`.

[3] R. Hoffmann, X. Zhu, Ö. Akgün, M. A. Nacenta, Understanding How People Approach Constraint Modelling and Solving, in: C. Solnon (Ed.), 28th International Conference on Principles and Practice of Constraint Programming (CP 2022), volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2022, pp. 28:1–28:18. doi:`10.4230/LIPIcs.CP.2022.28`.

[4] X. Zhu, M. A. Nacenta, Ö. Akgün, P. Nightingale, How People Visually Represent Discrete Constraint Problems, IEEE Transactions on Visualization and Computer Graphics 26 (2020) 2603–2619. doi:`10.1109/TVCG.2019.2895085`.

[5] S. Goodwin, C. Mears, T. Dwyer, M. G. de la Banda, G. Tack, M. Wallace, What do Constraint Programming Users Want to See? Exploring the Role of Visualisation in Profiling of Models and Search, IEEE Transactions on Visualization and Computer Graphics 23 (2017) 281–290. doi:`10.1109/TVCG.2016.2598545`.

[6] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006.

[7] J.-F. Puget, Applications of constraint programming, in: International Conference on Principles and Practice of Constraint Programming, Springer, 1995, pp. 647–650.

[8] M. Wallace, Practical applications of constraint programming, Constraints 1 (1996) 139–168.

[9] A. M. Frisch, W. Harvey, C. Jefferson, B. Martínez-Hernández, I. Miguel, Essence: A constraint language for specifying combinatorial problems, Constraints 13 (2008) 268–306. doi:10.1007/s10601-008-9047-y.

[10] Ö. Akgün, I. Miguel, C. Jefferson, A. M. Frisch, B. Hnich, Extensible automated constraint modelling, in: Proceedings of theTwenty-Fifth AAAI Conference on Artificial Intelligence, AAAI Press, 2011, pp. 4–11.

[11] M. G. de la Banda, K. Marriott, R. Rafeh, M. Wallace, The modelling language Zinc, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2006, pp. 700–705.

[12] M. Paltrinieri, A visual constraint-programming environment, in: International Conference on Principles and Practice of Constraint Programming, Springer, 1995, pp. 499–514.

[13] A. Bauer, V. Botea, M. Brown, M. Gray, D. Harabor, J. Slaney, An integrated modelling, debugging, and visualisation environment for G12, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2010, pp. 522–536.

[14] M. Carro, M. Hermenegildo, Tools for Constraint Visualisation: The VIFID/TRIFID Tool, in: P. Deransart, M. V. Hermenegildo, J. Małuszynski (Eds.), Analysis and Visualization Tools for Constraint Programming: Constraint Debugging, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 253–272. doi:10.1007/10722311_11.

[15] M. Carro, M. Hermenegildo, Tools for Search-Tree Visualisation: The APT Tool, in: P. Deransart, M. V. Hermenegildo, J. Małuszynski (Eds.), Analysis and Visualization Tools for Constraint Programming: Constraint Debugging, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 237–252. doi:10.1007/10722311_10.

[16] G. Dooms, P. Hentenryck, L. Michel, Model-Driven Visualizations of Constraint-Based Local Search, Constraints 14 (2009) 294–324.

[17] H. Simonis, P. Davern, J. Feldman, D. Mehta, L. Quesada, M. Carlsson, A Generic Visualization Platform for CP, in: D. Cohen (Ed.), Principles

and Practice of Constraint Programming – CP 2010, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 460–474.

[18] C. Schulte, Oz explorer: A visual constraint programming tool, in: International Symposium on Programming Language Implementation and Logic Programming, Springer, 1996, pp. 477–478.

[19] D. Kaiser, Physics and Feynman's Diagrams: In the hands of a postwar generation, a tool intended to lead quantum electrodynamics out of a decades-long morass helped transform physics, American Scientist 93 (2005) 156–165. URL: `http://www.jstor.org/stable/27858550`.

[20] R. Penrose, Applications of negative dimensional tensors, Combinatorial mathematics and its applications 1 (1971) 221–244.

[21] B. Tversky, What do sketches say about thinking, in: 2002 AAAI Spring Symposium, Sketch Understanding Workshop, Stanford University, AAAI Technical Report SS-02-08, 2002, pp. 148–151.

[22] J. Walny, S. Huron, S. Carpendale, An Exploratory Study of Data Sketching for Visual Representation, Computer Graphics Forum 34 (2015) 231–240. doi:`10.1111/cgf.12635`.

[23] Z. Liu, J. T. Stasko, Mental models, visual reasoning and interaction in information visualization: A top-down perspective, IEEE Transactions on Visualization & Computer Graphics 16 (2010) 999–1008. doi:`10.1109/tvcg.2010.177`.

[24] A. Kohnle, G. Passante, Characterizing representational learning: A combined simulation and tutorial on perturbation theory, Physical Review Physics Education Research 13 (2017) 20131.

[25] D. Gentner, A. L. Stevens, Mental models, Psychology Press, 1983. doi:`10.4324/9781315802725`.

[26] A. W. Crapo, L. B. Waisel, W. A. Wallace, T. R. Willemain, Visualization and the Process of Modeling: A Cognitive-Theoretic View, in: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '00, Association for Computing Machinery, New York, NY, USA, 2000, p. 218–226. doi:`10.1145/347090.347129`.

[27] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. a. Y. Silver, B. Silverman, Y. Kafai, Scratch: Programming for All, Commun. ACM 52 (2009) 60–67. doi:`10.1145/1592761.1592779`.

[28] D. J. Rough, A. J. Quigley, Jeeves-an Experience Sampling study creation tool, BCS Health Informatics Scotland (HIS) (2017).

[29] G. G. Méndez, M. A. Nacenta, S. Vandenheste, iVoLVER: Interactive visual language for visualization extraction and reconstruction, Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16 (2016) 4073–4085. doi:`10.1145/2858036.2858435`.

[30] J. Maloney, M. Resnick, N. Rusk, B. Silverman, E. Eastmond, The scratch programming language and environment, ACM Transactions on Computing Education (TOCE) 10 (2010) 1–15.

[31] M. M. Burnett, M. J. Baker, C. Bohus, P. Carlson, S. Yang, P. V. Zee, Scaling up visual programming languages, Computer 28 (1995) 45–54. doi:`10.1109/2.366157`.

[32] B. A. Nardi, A Small Matter of Programming: Perspectives on End User Computing, 1st ed., MIT Press, Cambridge, MA, USA, 1993.

[33] K. Marriott, B. Meyer, Visual language theory, Springer-Verlag New York, Inc., New York, NY, USA, 1998.

[34] N. Beldiceanu, H. Simonis, A Constraint Seeker: Finding and Ranking Global Constraints from Examples, in: J. Lee (Ed.), Principles and Practice of Constraint Programming – CP 2011, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 12–26.

[35] N. Beldiceanu, H. Simonis, A Model Seeker: Extracting Global Constraint Models from Positive Examples, in: M. Milano (Ed.), Principles and Practice of Constraint Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 141–157.

[36] N. Beldiceanu, G. Ifrim, A. Lenoir, H. Simonis, Describing and Generating Solutions for the EDF Unit Commitment Problem with the ModelSeeker, in: C. Schulte (Ed.), Principles and Practice of Constraint

Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 733–748.

[37] G. Fedyukovich, A. Gupta, Functional Synthesis with Examples, in: T. Schiex, S. de Givry (Eds.), Principles and Practice of Constraint Programming, Springer International Publishing, Cham, 2019, pp. 547–564.

[38] R. E. Horn, Visual language: Global Communication for the 21st Century, MacroVu Inc. Washington (1998).

[39] E. R. Tufte, The Visual Display of Quantitative Information, Cheshire, Conn. : Graphics Press, 2001. URL: `http://archive.org/details/visualdisplayofq00tuft`.

[40] B. A. Myers, Taxonomies of visual programming and program visualization, Journal of Visual Languages & Computing 1 (1990) 97–123. doi:`10.1016/S1045-926X(05)80036-9`.

[41] G. G. Méndez, U. Hinrichs, M. A. Nacenta, Bottom-up vs. Top-down: Trade-offs in efficiency, understanding, freedom and creativity with infovis tools, in: Conference on Human Factors in Computing Systems - Proceedings, volume 2017-May of *CHI '17*, Association for Computing Machinery, New York, NY, USA, 2017, pp. 841–852. doi:`10.1145/3025453.3025942`.

[42] J. Piaget, M. Cook, The Origins of Intelligence in Children, volume 8, International Universities Press New York, 1952.

[43] S. Papert, Mindstorms: Children, Computers, and Powerful Ideas, Basic Books, Inc., USA, 1980.

[44] J. Larkin, J. McDermott, D. P. Simon, H. A. Simon, Expert and Novice Performance in Solving Physics Problems, Science 208 (1980) 1335–1342. doi:`10.1126/science.208.4450.1335`.

[45] M.-A. D. Storey, F. D. Fracchia, H. A. Müller, Customizing a Fisheye View Algorithm to Preserve the Mental Map, Journal of Visual Languages & Computing 10 (1999) 245–267. doi:`10.1006/jvlc.1999.0124`.

[46] A. F. Blackwell, C. Britton, A. Cox, T. R. G. Green, C. Gurr, G. Kadoda, M. S. Kutar, M. Loomes, C. L. Nehaniv, M. Petre, others, Cognitive dimensions of notations: Design tools for cognitive technology, in: International Conference on Cognitive Technology, Springer, 2001, pp. 325–341.

[47] S. I. Robertson, Problem Solving: Perspectives from Cognition and Neuroscience (1st ed.), Psychology Press, 2001. doi:10.4324/9780203457955.

[48] M. Neurath, Isotype, Instructional Science 3 (1974) 127–150. URL: http://www.jstor.org/stable/23368119.

[49] Ö. Akgün, I. Miguel, C. Jefferson, A. M. Frisch, B. Hnich, Extensible Automated Constraint Modelling, in: W. Burgard, D. Roth (Eds.), AAAI 2011 - Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011, AAAI Press, 2011.

[50] S. Attieh, N. Dang, C. Jefferson, I. Miguel, P. Nightingale, Athanor: High-Level Local Search Over Abstract Constraint Specifications in Essence, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, {IJCAI-19}, International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 1056–1063. doi:10.24963/ijcai.2019/148.

[51] D. LIncoln, N. K. Denzin, Y. S. Lincoln, The SAGE Handbook of Qualitative Research, Sage Publications, 2005.

[52] P. E. McKnight, J. Najab, Mann-Whitney U Test, in: The Corsini Encyclopedia of Psychology, American Cancer Society, 2010, p. 1. doi:https://doi.org/10.1002/9780470479216.corpsy0524.

[53] A. Robins, J. Rountree, N. Rountree, Learning and Teaching Programming: A Review and Discussion, Computer Science Education 13 (2003) 137–172. doi:10.1076/csed.13.2.137.14200.

[54] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler,

J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language Models are Few-Shot Learners, ArXiv abs/2005.1 (2020).

[55] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. Ponde, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, I. Babuschkin, S. Balaji, S. Jain, A. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, W. Zaremba, Evaluating Large Language Models Trained on Code, ArXiv abs/2107.0 (2021).

[56] R. Moreno, R. E. Mayer, Cognitive principles of multimedia learning: The role of modality and contiguity., Journal of Educational Psychology 91 (1999) 358–368. doi:10.1037/0022-0663.91.2.358.

[57] J. C. Roberts, State of the art: Coordinated & multiple views in exploratory visualization, in: Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007), 2007, pp. 61–71. doi:10.1109/CMV.2007.20.

[58] A. Buja, J. A. McDonald, J. Michalak, W. Stuetzle, Interactive data visualization using focusing and linking, in: Proceeding Visualization '91, 1991, pp. 156–163. doi:10.1109/VISUAL.1991.175794.

[59] Q. Zhi, A. Ottley, R. Metoyer, Linking and layout: Exploring the integration of text and visualization in storytelling, Computer Graphics Forum 38 (2019) 675–685. doi:https://doi.org/10.1111/cgf.13719.

[60] S. Kandel, A. Paepcke, J. Hellerstein, J. Heer, Wrangler: Interactive visual specification of data transformation scripts, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 3363–3372. doi:10.1145/1978942.1979444.

[61] R. Faust, K. Isaacs, W. Z. Bernstein, M. Sharp, C. Scheidegger, Anteater: Interactive visualization for program understanding, CoRR abs/1907.02872 (2019). URL: http://arxiv.org/abs/1907.02872. arXiv:1907.02872.

[62] A. Binks, A. Toniolo, M. A. Nacenta, Representational transformations: Using maps to write essays, International Journal of Human-Computer Studies 165 (2022) 102851. doi:https://doi.org/10.1016/j.ijhcs.2022.102851.

[63] Z. Kiziltan, M. Lippi, P. Torroni, Constraint Detection in Natural Language Problem Descriptions., in: IJCAI, 2016, pp. 744–750.

[64] A. J. Parkes, H. S. Raffle, H. Ishii, Topobo in the wild: Longitudinal evaluations of educators appropriating a tangible interface, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '08, Association for Computing Machinery, New York, NY, USA, 2008, p. 1129–1138. doi:10.1145/1357054.1357232.