Qualitative Constraints and CHR

Introduction

This note is about some things I found out about Constraint Handling Rules (CHR) and how they could be applied to our IMMPS-system during my 1 month-summer stay. I only describe here a small part of the conceptual system, so a more broader view (as described by Jacco's 2b tech-report) of the conceptual system is recommended to be read first.

The part I am aiming at is the qualitative constraint level. The main question here will be ' what is the proper way to implement them'.

Problems in generating hypermedia

Generating hypermedia documents is a complex issue. Not so much because the problems you encounter require lots of calculations but more in a way that the solution of one problem triggers new problems which are again highly dependent of each other. Besides this it is hard to define a solution because it is more like a trade-off between lots of alternatives. A certain alternative might be good from one point of view but from another perspective it might as well be a bad idea.
This type of problems are also known as so called 'optimalisation problems' which are quite hard in general.

examples.

problem: The choice of the distance (padding) between two images.
The amount of white space available can only be know at the end when screen size and the size of the images are know. So, by then the padding can be calculated to create, lets say a nice distributed lay-out effect. However choosing the same padding for different items could also have a 'semantic' effect. It could for example convey an idea of similar items. Thus an author might just as well as an designer want to specify the padding between various items.

problem: The position of label according to an image.
Suppose for some reason a label can be below an image or at the right side of an image. We can just pick one if both will do. This can be done quite early, but we only know at the end whether this will fit on our screen or we have to revise our earlier decision and try a different approach.

problem: We have some images conveying the same semantic concept, we do not want to show them all because there is not enough space.
We have to make a choice based on specific properties found in the database or knowledge base to find the best suitable images and leave some others out.

problem: We have some images with responding labels and we want to make sure all the labels are placed in the same way to the images throughout the presentation.

As you can see the process of generating a presentation is not pipe-line-like at all. The process consist of lots of going back and forward. Solution to a 'style' problem triggers 'semantic' problems, therefore a highly flexible data-structure is a definite requirement.

A *procedural* approach will probably not work because this approach tend to simplify the problem at every step. In our case there is not a definite way the problem could be reduced every step because of the side-effects mentioned before.

An alternative approach would be *declarative* (eg. prolog). This is probably better because the problem can be specified at a higher level. This is a nice feature but most certainly not enough to deal with the complexity issues

involved here.
Two images can be placed in very many different ways on a screen. Let alone three or more, and this is only in a spatial dimension we don't even talk about a temporal dimensions yet.

Constraint Solvers (like delta blue) are able to make these kind of problems computable because of their ability to exclude certain possible domain values. (eg. the constraint X > 10 reduces the domain of X by removing all value <= 10) Problem with constraint solvers is that once you specified all the constraints you either get a solution or you don't. There is no way that you can revise a constraint once you found out it didn't work.
In our IMMPS-system there are lot of alternatives to choose from and which are the right one is difficult to predict. So in order to get a solution you simply have to calculate each solution and this obviously would take long.

Constraint Logic Programming systems (such as Eclipse) provide a way of combining the pleasant features of Constraint Solvers (domain reduction) and Logic Programming (backtracking)

Our Cuypers systems used the features of Eclipse and it was able to solve some of the problems we had, like the mapping of a higher-level semantic specification to a hypermedia presentation taking into account a variable screen size.

Cuypers was entirely numeric based. The solution of the problem consisted of coordinates and time intervals on which the various media items should be played. The use of numeric constraints turned out to be still too computational explosive. Besides this the structure was not flexible enough to express simple consistency constraint (well not in a nice way at least)

We developed a new conceptual model called Mey, which can be read about in jacco's tech-report but basicly it divided the process in five conceptual separate layers each with their specific task. I emphasize conceptual because in practice the different layers are most likely not as nice separated as in the model. This is because of the fact all layers could influence other layers.

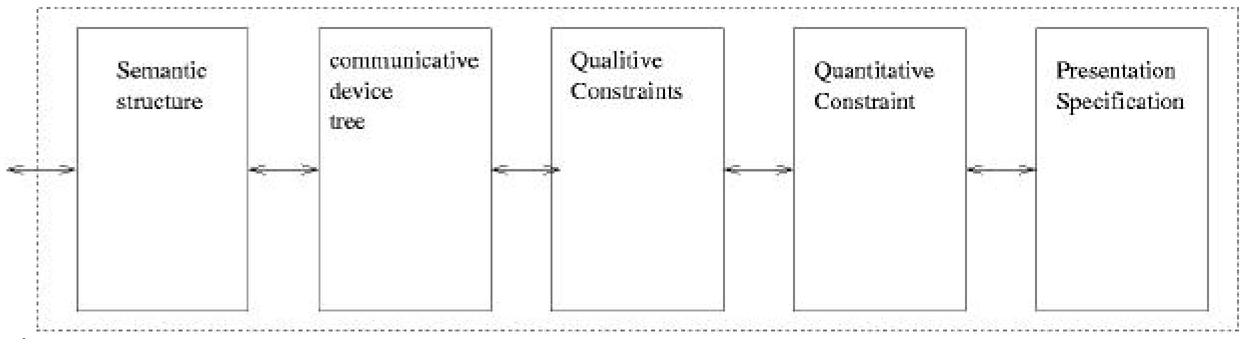

Figure1

Why qualitative constraints

In the Mey model there is less emphasis on the quantitative (numerical) constraints as it was in Cuypers. We introduced a new level of qualitative constraints (eg A leftof B) , which are supposed to give you a higher level of presentation specification. It reduces the complexity because of the smaller domains. Non-numeric properties like linking are more easily expressible in qualitative constraints then they are in quantitative. And higher-level specifications of timing (in smil) and space (html, smil2) are possible as well.

Qualitative Constraints

Talking implementation again. Qualative constraints are also called user-defined-constraints in Constaint Logic Programming systems like Eclipse, which means that there is no library available which specifies how to handle them. In the numeric domain some constraints are pre-defined such as constraints like A > B, which causes the domains of A and B to shrink. User defined constraints on the other hand, you have to specify yourself by means of

'reasoning rules' which state how to handle our constraints. For example, we need a transitive rule which states that if Image A is left of Image B and Image B is left of Image C that Image A is left of Image C as well.
We need a symmetric rule which states that if Image A is left of Image B than Image B is right of Image A.

CHR (see: http://www.pst.informatik.uni-muenchen.de/~fruehwir/chr-intro.html)

One way of specifying these rules is by use of a language called CHR (Constraint Handling Rules) This is a declarative language extension of constrain-based systems such as Eclipse and prolog. It is designed to specify user constraints in a nice manner.

Bassicly a CHR-program consists of several rules. There are two types of rules which are propagation rules and simplification rules. (Actually there are three, the third is called simpagation rule which is a combination of the first two)

	Name	Head	Type	Guard	Body
simplification:					
	Name @	H_1... H_i	<=>	G_1...G_j |	B_1...B_k
example:					
	sym1 @	A rgt B	<=>	not audio(A), not audio(B) |	B lft A.
propagation:					
	Name @	H_1... H_i	==>	G_1...G_j |	B_1...B_k
example:					
	trans1 @	A lft B, B lft C	==>	img(A),img(B),img(C) |	A lft C.

They work as follows. Suppose we have specified the constraint img1 rgt img2 then the set of constraints will look like:

{ img1 rgt img2 } *note: I will call this set the constraint store*

The CHR-rules all have a Name which is only there for debugging reasons. (It is nice to know which rule is applied) They also have Heads. A Head is a constraint (or a sequence of constraints) which determines whether a rule is applicable or not. If a constraint from the constraint store matches a Head, the Rule is applied. So in our example the constraint A rgt B matches the head of the rule sym1. This is a simplification rule as the <=> operator shows. Simplification rules **replace** the Head by a new constraint which is constructed in the Body of the rule. But before entering the Body we have to pass the Guard. A Guard is final test which should succeed before the rule is applied. In our case the Guard states that neither A or B should be audio. Since we only have images here the test will succeed and we pass to the Body. The Body constructs a new constraint and replaces A rgt B for B lft A. So the constraint store was { img1 rgt img2 } but after the rule sym1 is applied the constraint store will look like:

{ img2 lft img1 }.

In order to show how propagation rules work we have to extend our constraint store with img1 lft img3

{ img2 lft img1, img1 lft img3 }

This set of constraints matches the Head of rule trans1 which is a propagation rule. This means that the constrain store is now **extended** with the new constructed constraint from the body. So after the rule trans1 is applied the constrain store will look like:

{ img2 lft img1, img1 lft img3, img2 lft img3 }

Rules have no hierarchical order. What is the next rule will be determined by the CHR-compiler. So the order of the rules does not influence the behavior of the program.

Applying User defined Constraints on non-integer domains

User defined constraints are supposed to map eventually on built-in constraints such as inequality-constraints (<, >), equality (=) and truth-values (True, False) But since in-equality constraints are only defined for numerical values we have to enumerate all possibilities by means of equality constraints.
Example:
Suppose we have an boolean domain { false, true } and a constrain and(A,B,C). These are not numeric, so we have to express the And constrain by means of equality constraint. A,B and C can all be either true or false. But once we know C is true we also know that A and B have to be true. A rule to express this would be

and(A,B,true)	==>	A=true, B=true

other rules for the and constraint

and(true, true, C)	==>	C=true
and(A, true, true)	==>	A=true
and(true, B, true)	==>	B=true
and(A, false, C)	==>	A=false, C=false
and(false, B,C)	==>	B=false, C=false

A simple 'And-constraint requires already 6 rules to express it. You can imagine the number of rules required to express more complex domains and constraints becomes quite large. In fact to express a transitive relation on the 13 Allen temporal relations (eg tr(overlaps, overlaps, before)) we will need about 500 rules. Our domain in Mey is more complex than Allen's so we will need considerable more rules to express a transitive relation. Too large to compute by hand.

Fortunately Krzystof Apt and Eric Montfroy have developed an algorithm to generate these rules automatically. (see Automatic Generation of Constraint Propagation Algorithms for small finite domains) If we take again the transitive relation on Allen Relation this means that we only have to give input for 13 x 13 = <u>169</u> cases. Which is still quite a lot but since we only have to specify it once it should be feasible.

Applying User Constraints in Mey.

Now we know something about qualitative constraints and how to implement them it is time to have a look at how they are applicable in our situation. What would be the data structure, what are the domains we are talking about, and how should we reason with them.

Structure

The structure should be highly flexible the limitation of possible operations on the structure should be as small as possible. In Cuypers a tree structure turned out to be not flexible enough since we were not able to express simple consistency constraints. A completely connected directed graph where the nodes are media-items and the edges are relations between the media items doesn't limit the expressibility of the operations in any way.
Although the structure is very flexible it is quite poor as well. We can only specify a relation between two arbitrary media-items. Most of the time we have groups of media-items which we want to relate. To provide this functionality we have to extent our structure a little with so called group-nodes. A group-node is about the same as a media-item-node the only difference is that there is no media-items associated with the node. Media items can be part-of a group node and the group-node can have an explicit relation with other nodes which results in several implicit relation between the members of the group and that particular node.

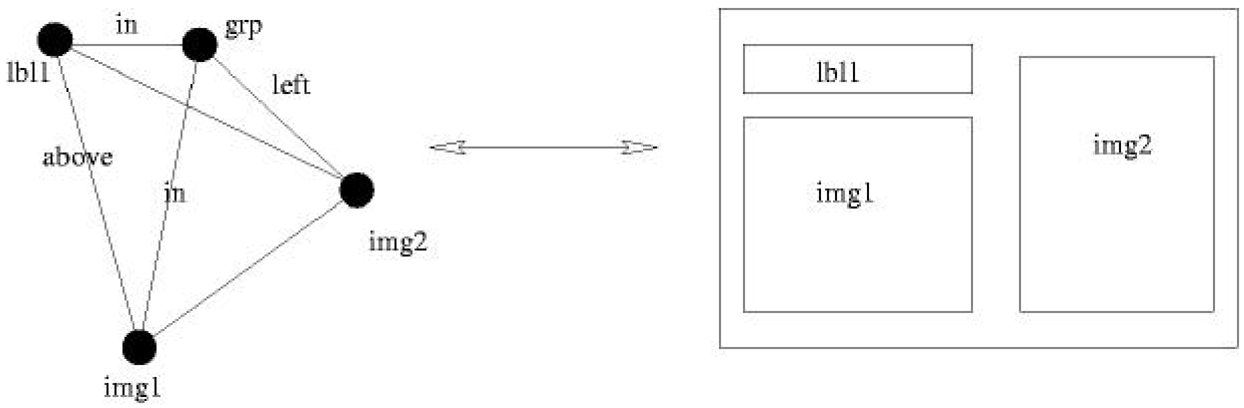

Figure 2

In the picture above only a single relation between the media items is displayed. Since the graph is directed an opposite relation is required as well. And we only talked about one dimension yet, in fact there are many more.

What dimension do we actually need, and what are the domains we are talking about?

Six Hypermedia Dimensions

X-dimension

The X dimension is the spatial dimension in the horizontal direction. The domain of the X dimension is similar to the Allan-relations for time. There are 13 distinct ways two spatial items can be placed according to each other.

Y-dimension

The Y dimension is a spatial dimension in the vertical direction. The domain here are the 13 Allan relations as well. Combined with the X-dimension we could express things like centred, horizontal- and vertical alignment and partial overlap.

Z-dimension

Two spatial items can overlap, which one should go on top of the other is specified in the Z dimension. The domain of the Z can be either binairy, {above, underneath} or consist of a level-structure. {0,1,2,3,4,5...} The latter one is more expressive since you can specify things like 'these items should be placed on the same level' but if this is really nessesary we have to find out.

Time-dimension

Allan's 13 time relations

Link-dimension

The link-dimension is binary, two media items can either have a link or they don't.

Type-dimension

The type of the media-items is required since it determines which dimensions are relevant. An image and an audio-fragment can't have a spatial relation for example. Besides this you also want to specify that two video's cannot be played at the same time. To be able to do this, you will also need the type of the media items. The domain will be something like{ thumbnail, image, text, audio, music, video ...}

Except for the type dimension all these dimensions are independent of each other

Rules

transitive (spatial, time)
symmetric (spatial, time)
equality (spatial, time, type)
meta-constraint

Communicative Device 2 Qualitative Constraints

Before we are able to reason with constraints there has to be made an translation from the communicative devices to quality constraints. One way this might be possible is by considering communicative devices as constrains. If we take for example the communicative device 'Bookshelf', which orders an arbitrary number of media items from left to right and top to bottom. and we transform it to a constraint it might look like

```
bookshelf( [] )           <=> true
bookshelf( Items )        <=> append(L1,L2,Items), % L1 starts as large as possible
                                  left2right(L1),
                                  bookshelf(L2),
                                  above(L1,L2).

left2right([] )           <=> true.
left2right([_Item])       <=> true.
left2right([A,B|Items])   <=> A leftof B,
                                  left2right(Items).
```

This approach is based on the fact that in a constraint definition the body of the constraint can execute prolog-clauses. This way you can transform a communicative device which ought to be procedural to a declarative constraint. Once all constraints have been specified and the solver tries to find an answer it might be possible that the left2right constraints fails because it exceeds the screen boundary. The process backtracks over the append clause and tries again with a new instantiation . This approach is in a way similar to the way as it used to be in Cuypers but the difference is that in Mey the backtracking occurs in the violated constraint while in Cuypers backtracking occurred on the last made choice. If the constraint which caused the problem was specified early in the process it could take quite some time before the backtracking came to revise this particular constraint.

For now I don't have any results if this approach works in practice. To be able to try some things out we will need the implementation of the rules generation algorithm of Krzysztof and Eric.

Why depth first approach.

The qualitative layer is only a part of the Mey system. There are roughly two ways to continue the development of Mey. The breadth-first approach which bassicly tries to implement the process as a whole from top to bottom as fast as possible, leaving the details for later. And the depth-first approach which implements only a small step at a time but does this in full detail.
The breadth-first approach has some obvious advantages which are parallelization, different people could work separately on a part of the system. The sub-systems could be combined later in one system. And the overall structure of the process and problems becomes clearer.

Although in many cases I would prefer the breath-first approach for the reasons above. In this particular case however I think the depth-first approach is probably better.
As mentioned before the architecture of Mey is highly inter connected. Although the different tasks are divided into separated layers from an implementation point-of-view they are not since decisions in the first layer can influence the input in the last layer and vice-versa. So, from an implementation point of view they are not really different sub-systems but in fact they are one.

A breadth-first approach on a single complex task might introduce unexpected errors when the sub-tasks are combined. Where exactly the errors comes from you don't know. Maybe when you have two subtask you eventually find out but when there are five and there are multiple errors it could become quite difficult.

I think an incremental development for Mey is better because in this way you can control the working of the system at any stage. Once you have a stable system and you extend it with a particular functionality, you know what are the causes of the errors you will get. I don't know if the whole system should be incremental, I can for example imagine the step from rhetoric and narrative to communicative devices is in a way quite independent and could be developed separately.

That's about it I guess, comments are most welcome.

joost