Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

Application-Specific Constraints for Multimedia Presentation Generation

J.P.T.M. Geurts, J.R. van Ossenbruggen, H.L. Hardman

Information Systems (INS)

INS-R0107 May 31, 2001

Report INS-R0107
ISSN 1386-3681

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Application-Specific Constraints for Multimedia Presentation Generation

Joost Geurts, Jacco van Ossenbruggen and Lynda Hardman
Firstname{.van}.Lastname@cwi.nl

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT

A multimedia presentation can be viewed as a collection of multimedia items (such as image, text, video and audio), along with detailed information that describes the spatial and temporal placement of the items as part of the presentation. Manual multimedia authoring involves explicitly stating the placement of each media item in the spatial and temporal dimensions. The drawback of this approach is that resulting presentations are hard to adapt to different target platforms, network resources, and user preferences.

An approach to solving this problem is to abstract from the low-level presentation details, for example by specifying the high-level semantic relations between the media items. The presentation itself can then be generated from the semantic relations along with a generic set of transformation rules, specifying how each semantic relation can be conveyed using multimedia constructs. These constructs may differ depending on the target platform, current network conditions or user preferences. We are thus able to automatically adapt the presentation to a wide variety of different circumstances while ensuring that the underlying message of the presentation remains the same.

This approach requires an execution environment in which transformation rules, resulting in a set of constraints, are derived from a given semantic description. The resulting set of constraints can then be solved to create a final multimedia presentation. The paper describes the design and implementation of such a system. It explains the advantages of using constraint logic programming to realize the implementation of both the transformation rules and the constraints system. It also demonstrates the need for two different types of constraints. Quantitative constraints are needed to verify whether the final form presentation meets all the numeric constraints that are required by the environment. Qualitative constraints are needed to facilitate high-level reasoning and presentation encoding. While the quantitative constraints can be handled by off-the-shelf constraint solvers, the qualitative constraints needed are specific to the multimedia domain and need to be defined explicitly.

1998 ACM Computing Classification System: H.5.4, H.5.1, I.7
Keywords and Phrases: Multimedia, transformations, constraints, CLP, CHR, ECLiPSe
Note: The research reported here has been carried out under the Dynamo, RTIPA and ToKeN2000 projects.

1. Introduction

The need for presentation adaptation to a wide variety of circumstances is of crucial importance to current and future content providers. From a technical perspective, different devices such as mobile phones, PDAs and fully equipped multimedia PCs should be able to convey basically the same information while exploiting their device-specific capabilities. In addition, to maximize the user's level of understanding, the presentation should be conveyed in the way most suited to the particular user. Manually authoring all these different presentations is in principle an option, but is in practice too costly. Generating presentations automatically is an alternative for obtaining multiple presentations of reasonable quality which are tailored to the context in which the presentation is played.

In order to construct a presentation, whether by a human author or a system, a number of aspects have to be taken into account. In particular, the "message" of the presentation has to be conveyed using selected media items, the media items have to be arranged in some aesthetically pleasing manner,

and all this has to be presented in the screen dimensions of the user's device and time-limits of the user. Solutions are possible, but only after contradictory requirements have been traded-off against one another. Examples of such trade-offs include:

- **Semantics versus aesthetics** The presentation as perceived by the user is intended to convey the message as envisaged by the author. The message itself is conveyed through the media items included in the presentation, the temporal and spatial layout of the items and the choice of style characteristics. These are both the means used for increasing the aesthetic appeal of a presentation and for expressing the author's message. Just as for established paper-based graphic design, there is a trade-off between using any particular technique for emphasizing either the aesthetic appeal or the semantic content. For example, the padding distance between the items on the screen can be varied purely for aesthetic reasons to improve the layout. On the other hand, padding distance could also have a semantic purpose, for emphasizing structural relations among items by strengthening the visual grouping in the presentation. Determining the resulting padding distances needs to take both semantic and aesthetic effects into account.

- **Aesthetics versus feasibility** Having established design rules for a desired layout, these may prove to be insoluble for the user's device. In which case, some sort of compromise needs to be made in order to come up with a feasible alternative. For example, the designer decides to place a label above an image to provide a title and to do this throughout the presentation for maintaining consistency. This turns out to be infeasible because of lack of screen space. There is, however, sufficient room to position the labels consistently to one side of the image. The designer has to make a choice as to whether the preferred position is used wherever possible, or to use the less aesthetically pleasing position consistently.

- **Semantics versus feasibility** A designer may even be forced to discard some of the media items intended to be presented. For example, suppose a number of concepts in the presentation are to be clarified by an example. A query to a multimedia database supplies the author with various media items which are examples of the concepts to be conveyed. The designer, however, is restricted to a maximum duration of the presentation. The designer thus has to make a decision as to which concepts can be illustrated by an example and which have to be omitted.

In practice, there are many reasons why the "ideal" presentation may prove to be infeasible. In the majority of cases, a trade-off between different objectives is necessary. For example, in the last example the media items that best illustrate the concept would be the first choice. These media items, however, might not be suitable for the presentation because of their size or duration, and so some trade-off needs to be made to select media items which satisfy some combination of the objectives as best as possible.

These examples show that the process of creating a presentation is not linear. Rather, the process consists of going back and forth through the conceptual layers of generating a presentation. Solutions to an aesthetic problem can trigger semantic problems and *vice-versa*. While a human author is faced with these difficulties in creating an ideal presentation, the same problems are also faced when attempting to capture the process in a presentation generation system.

As shown by the fundamental trade-offs of semantics, aesthetics and resources, presentation generation does not lend itself readily to the traditional divide and conquer techniques commonly used in computer science. Decisions taken about the semantics of the presentation steer the allocation of screen and temporal resources, while final details at the resource level (*e.g.* an image that is a few pixels too wide or a video that is a tenth of a second too long) may force the system to re-evaluate previously made semantic decisions.

From an implementor's perspective, a system for creating multimedia presentations automatically has to not only make the decisions an author would make, but also cope with the trade-offs that have to be made when chosen solutions fail for other reasons.

Our approach to creating a prototype multimedia presentation generation system is thus to identify a number of different levels of abstraction encountered in the process of creating a multimedia presentation, and to embody these in conceptually distinct steps in a presentation creation process. Starting with a presentation-independent abstraction, specified in semantic terms, the prototype makes selections and choices, calculating potential solutions for providing a presentation, and finally creates a final-form presentation tailored to the specific requirements of the user, device and network conditions. At run-time, backtracking can take place at each step, and a number of steps are highly intertwined.

Constraints play an important role in a number of steps in the generation process, and it is this role which is the focus of this paper. In particular, we show that quantitative constraints are important for making decisions as to the feasibility of potential solutions, and that quantitative constraints are of themselves insufficient, requiring a higher-level constraint mechanism, which we term qualitative constraints, for providing higher-level abstractions for processing.

The remainder of the paper is structured as follows. In the next section we incrementally introduce more complex constraint solving techniques as potential solutions to deal with the complexity issues described above. In section 3 we briefly give an overview of our multimedia presentation generation prototype, Cuypers, and then describe in detail how quantitative and qualitative constraints have been implemented. We discuss related work in section 4 and present our conclusions and future work in section 5.

2. Constraint Solving Techniques

The problem of generating multimedia presentations can be viewed from different perspectives using different techniques. A first step towards a solution is the choice of a suitable platform and framework able to process the type of information needed. We argue that constraint satisfaction techniques are beneficial to the problem of generating multimedia because of their efficiency and their suitability for expressing the problem at a high level.

Constraint solving is similar to other declarative approaches to the extent that the programmer does not provide an algorithm but gives the requirements the solution has to conform to. How these requirements are processed internally is not an issue for the programmer. Constraint solving focuses on problems where there are many variables and large associated domains. It is the task of the constraint solver to reduce the size of the domains as efficiently as possible. For example, suppose we have two variables $x \in [0..5], y \in [0..100]$ and the constraint $x > y$. The values 5..100 of y will never satisfy the constraint and could thus be eliminated from the domain. This results in: $x \in [0..5], y \in [0..4]$ which reduces the domain of y by 96 possibilities. While this is an extremely simple example, similar techniques can be used to reduce the domain efficiently in multimedia presentation generation, in particular when calculating the spatio-temporal layout of the various media items.

Our research group has been involved in developing several prototypes to explore different aspects of multimedia presentation generation. The prototypes have been developed in an incremental way and each prototype can be regarded as a reaction to the lessons learned during the development of its predecessor. While they all use some form of constraint satisfaction, they each approach the problem differently. Below, we give a short overview of the prototypes developed, since they provide key insights in both the practical aspects of constraint solving and the more theoretical aspects of multimedia presentation generation.

2.1 Constraint Programming

The first prototype, developed by Bailey *et al.* [8], aimed at the automatic transformation of the results of simple multimedia database queries into a multimedia presentation encoded in SMIL. The system initially consisted of a set of high-level transformation rules, specified in Prolog, that converted structured results from the database into SMIL presentation constructs. However, the detailed calculation of the exact spatial coordinates of the presentation's visual layout and the precise synchronization requirements made it difficult to ensure that the transformation rules remained generic and were specified at the right level of abstraction. To address this problem, an extra layer was added to

the system. The transformation rules in Prolog no longer needed to do the calculations, and instead needed only to generate the set of constraints that the desired spatio-temporal layout should meet. A second program, written in Java, used these constraints to determine the exact coordinates of the resulting SMIL presentation. The prototype used an off-the-shelf constraint solver, Cassowary [4], to apply different types of domain reduction rules.

The problem with this approach, however, is that it is in general not possible to efficiently revise a constraint. Once a set of constraints has been determined, the system can either solve them or not. If it cannot, it is not known which constraint caused a failure and a complete new set of constraints needs to be developed, in the hope that the new set will resolve the previous problem. Only when all possible sets of constraints have been tried is it known whether they were feasible or not. Additionally, the fact that the constraint generator (in Prolog) and the constraint solver (in Java) were two separate programs made it even harder to control the revision of constraints in a convenient way. Constraint Logic Programming(CLP) focuses on this type of problem.

2.2 Constraint Logic Programming

Constraint Logic Programming is a combination of logic programming and constraint solving. It combines features from Prolog, such as backtracking and unification, with the domain reduction techniques from the constraint solver paradigm. Practically, this means that alternative constraints can be invoked when the original one caused a failure. This is an improvement in comparison with the previous approach, since we need only try alternatives for constraints which caused a failure.

The next prototype was based on ECLiPSe [14], a true constraint logic programming system that supports different types of domains such as integers, reals, sets etc. In addition to its built-in ability to revise constraints, another improvement of ECLiPSe is that the system is able to combine the processing of the transformation rules and the constraints within a single execution environment. This is not only more efficient, but also allows for a more sophisticated interaction between these two phases in the generation process.

Another improvement of the ECLiPSe-based prototype is the introduction of another abstraction layer for the transformation rules. In fact, when transforming from the semantic structure of the document to a set of constraints, two types of decisions need to be made. First, one needs to decide on how the semantics can best be presented. Second, one needs to decide on how this presentation is to be represented by a set of constraints. To be able to discriminate between these two different levels, we introduced the notion of a *communicative device* [9]. Communicative devices are patterns of often used multimedia presentation design, and allow the designer to choose the most appropriate presentation pattern to convey the semantics of the document to the user. The transformation of the communicative devices into a set of constraints is a separate task, and is the responsibility of the programmer, not the designer.

Despite these improvements, the new prototype suffered from a different drawback. While multimedia presentations could be described easily by sets of constraints that were automatically generated by the transformation rules, for various types of processing these constraints proved to be too low level. For example, due to the low-level specification of spatial layout in SMIL, a solved set of constraints provides the necessary spatial information at exactly the right level: in the form of x and y coordinates, with *width* and *height* values. In contrast, SMIL has much higher level facilities for defining the temporal structure of the presentation, and the flat list of numeric begin and end times for each media item is *not* the most appropriate level of abstraction. Clearly, higher-level descriptions of the temporal relations, including grouping and hierarchical structures, are needed to deliver high quality SMIL output.

Other, perhaps more fundamental, drawbacks of the low-level nature are related to inappropriate backtracking behavior. For example, when the domain of a specific coordinate has been reduced to, say [5..15], and the resulting layout with $x = 5$ fails for some other reason and causes the system to backtrack, the solver might try $x = 6$, $x = 7$, etc. This will, when the coordinates are expressed in pixel units, generate a number of similar layouts that differ by only one pixel value for one media

item: clearly, in most cases this is not the desired backtrack behavior. Instead of backtracking on the quantitative level, it is often more appropriate to backtrack on a more qualitative level, *e.g.* by backtracking over the decision that A should be left of B, by trying, for example, A above B.

While ECLiPSe has no built-in solvers for qualitative constraints such as "left-of" or "above", it supports the definition of application-specific domains and constraints. These are typically called *user-defined constraints*.

2.3 User-Defined Constraints

The drawbacks of the use of only numerical domains and constraints in the prototype discussed above lead to our current prototype, which implements an extra layer of abstraction, based on qualitative constraints. Unlike the generic numerical constraints, the qualitative constraints used are specific to multimedia presentation applications. In ECLiPSe, qualitative, or user-defined, constraints have no associated built-in library which specifies how to deal with these types of constraints. Instead, the application needs to provide the rules which the system can use to reduce the domains of the associated variables. For example, we need a transitive rule which states that if image A is left of image B and B is left of image C then A is left of C as well. We also need a symmetric rule which states that if A is left of B then B is right of A, etc.

Together, these domain reduction rules can be seen as a formalization of the implicit "common knowledge" human authors have about spatial and temporal relations. A complete specification of these rules, however, is not trivial and is in fact a programming task in itself. One way of specifying such rules is with a language that is designed for this purpose: Constraint Handling Rules (CHR), a declarative language extension of constraint-based systems such as ECLiPSe.

A CHR program consists of several rules, where each rule consists of a name, head, type, guard and body. Basically there are two types of rules which are *simplification rules* and *propagation rules*[1]. For example, the symmetry relation between left and right mentioned above might be specified using a simplification rule ($\Leftrightarrow$) named *sym*1, defined as:

```
% Name     Head             Type  Guard             Body
  sym1   @ A rgt B          <=>   visual([A,B]) |  B lft A.
```

Note the use of the guard that prevents this rule to be tried for non-visual items (such as audio fragments). The transitive nature of the left of relation could be specified using a propagation ($\Rightarrow$) rule named *trans*1:

```
% Name     Head               Type  Guard             Body
  trans1 @ A lft B, B lft C  ==>   A##B, B##C   |   A lft C.
```

These CHR rules can be applied as follows: suppose we have specified the constraints "image 1 should be placed to the right of image 2" and "image 1 should be left of image 3" then initially the working set of constraints (also called the *constraint store*) is:

{ img1 rgt img2, img1 lft img3 }

The CHR-rules all have a name which is only there for debugging reasons — it is convenient to know which rule is applied when. They also have a head, which 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 "img1 rgt img2" matches the head of the rule *sym*1. This is a simplification rule as the $\Leftrightarrow$ operator shows. Simplification rules replace the head by the constraint which is defined by the body of the rule. But before this replacement can proceed, we have to pass the guard. A guard is a final test which should succeed before the rule is applied. In our case the guard states that A or B should be visual items. Since we only have images here the test will succeed and we pass to the body. The body defines a new constraint and replaces "A rgt B" for "B lft A". So the constraint store was

[1]Strictly speaking there is a third type, called the *simpagation rule* which is a combination of the first two.

{ img1 rgt img2, img1 lft img3 }

and after the rule *sym*1 is applied the constraint store is

{ img2 lft img1, img1 lft img3 }

This set of constraints matches the head of rule *trans*1 which is a propagation rule. This means that the constraint store is now extended with the new constructed constraint from the body. So after the rule *trans*1 is applied the constraint store will look like:

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

Note that unlike Prolog clauses, CHR rules have no order. The next rule that fires is determined by the CHR-compiler, and the order in which the rules are specified does not influence the behavior of the program. If there is no applicable rule then the constraint-processing delays until a variable in one of the constraints changes. The rules whose head match the altered constraint will then be re-applied.

2.4 Automatic Rule Generation

User-defined constraints eventually need to be mapped onto built-in constraints, such as inequality ($\leq, \geq$) or equality ($=$), which will evaluate to either true or false. Since inequality constraints are only defined for numerical values, we have to enumerate all possibilities by means of equality constraints. In practice, the number of possibilities is often too large to allow manual enumeration, thus requiring some means of automatic rule generation.

For example suppose we have a Boolean domain $\{false, true\}$ and a constraint $and(A, B, C)$ representing $A \wedge B = C$. The domain and constraint are not numeric, so we have to express the *and* constraint by means of equality constraints. While A, B and C can all be either true or false, once we know C is true we also know that A and B have to be true, etc. CHR rules to express this could 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={true,false}, C=false
  and(false,B,C)   ==> B={true,false}, C=false
```

These six rules completely define the ‘and’ constraint. This constraint uses a binary domain and is therefore quite straightforward. In practice, the domains for multimedia generation are often larger which result in more complex rules. For example, to fully define the transitive behavior of the 13 Allen temporal relations [1], it turns out that almost 500 CHR rules are needed. In addition, our constraint system also needs similar rules for the spatial dimensions. Clearly, manual specification of all these rules is not realistic.

Fortunately, the majority of these rules can be generated automatically. In particular, we used the rule generation algorithms and tools developed by Apt and Monfroy [3] to generate many of the needed rules automatically. Since the Allen domain consists of 13 values, there are $13 \times 13 = 169$ combinations which reduce the domain of the third variable. These combinations are completely defined by Allen ([1], the table in Figure 4 on page 836). The translation of this table to the corresponding CHR-rules is done automatically by the algorithm.

Apt and Monfroy define two versions of the algorithm. The first leads to a local consistency notion called *arc consistency*. Practically, this means that the domains associated with the three variables in a transitive constraint contain only values which participate in a solution. The other version of the algorithm leads to a local consistency notion called *rule consistency*. This notion excludes values which do not lead to a solution. Since rule consistency is a weaker notion than arc consistency, the arc consistent rules can exclude more values which results in faster solutions.

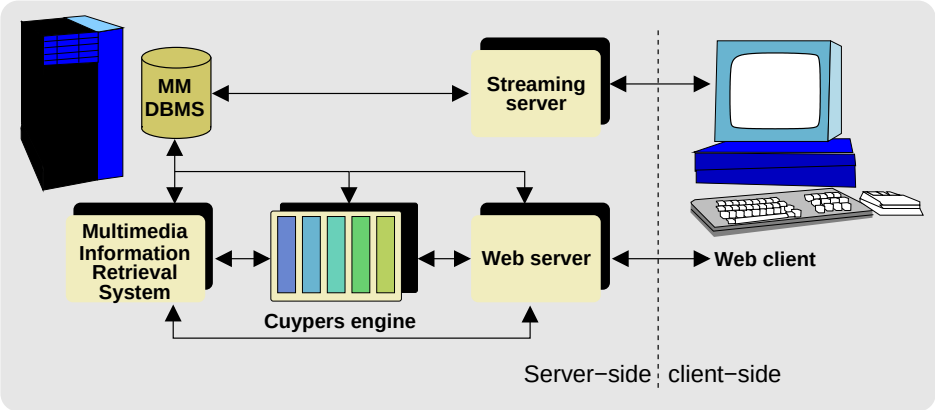

Figure 1: Overview of the Cuypers client/server architecture

The generation of all the arc consistent rules, however, takes about 5 days on a 500MHz PC, and results in more than 26000 rules. In contrast, the rule consistent algorithm generates only 498 rules in less than 6 seconds. Apart from the time it takes the algorithm to produce all the rules (which needs to be done only once), there is also a huge difference in the time needed to load the generated rules during the start-up of the presentation engine. So from a practical perspective, we choose to used the rule consistent version in our current prototype even though, theoretically, the arc consistent version is more efficient. In future versions of the system, when we have a more stable, continuously running presentation engine, the increase in run-time efficiency of the arc consistent rules might outweigh the extra overhead at start-up time. The next section describes the current prototype, the Cuypers multimedia presentation engine, in more detail.

3. Constraint Processing in Cuypers

Cuypers is our current prototype system for generating multimedia presentations. This section initially gives an overview of the system including how an abstract semantic structure is transformed into a playable multimedia presentation. It then discusses in detail the use of constraints in the different stages in the generation process.

The Cuypers system is designed to operate in the context of a client/server architecture, as depicted in Figure 1. The figure shows two different servers: one dedicated server to deliver streaming media (such as audio and video fragments) to the client, and another, off-the-shelf, Web server to deliver the other data, including the generated multimedia presentation itself (we currently use SMIL [13] for this purpose). The core of the Cuypers system receives, from a multimedia information retrieval system, a semantic description of the multimedia document as its input, and sends the generated presentation to the server, for further delivery to the client (see [12] for more details).

The core generation engine, as depicted in Figure 2, is built around the five conceptual abstraction layers discussed in the previous section.

1. **Semantic structure** The semantic structure is a high-level semantic description of the presentation. It completely abstracts from the design and layout of the presentation. The designer provides a set of transformation rules to transform this structure into a specification based on communicative devices.

2. **Communicative devices** Communicative devices are abstract constructs that specify how the information should be conveyed to the user. Communicative devices are device independent, and can be used to convey the similar presentations for different devices. An example of a

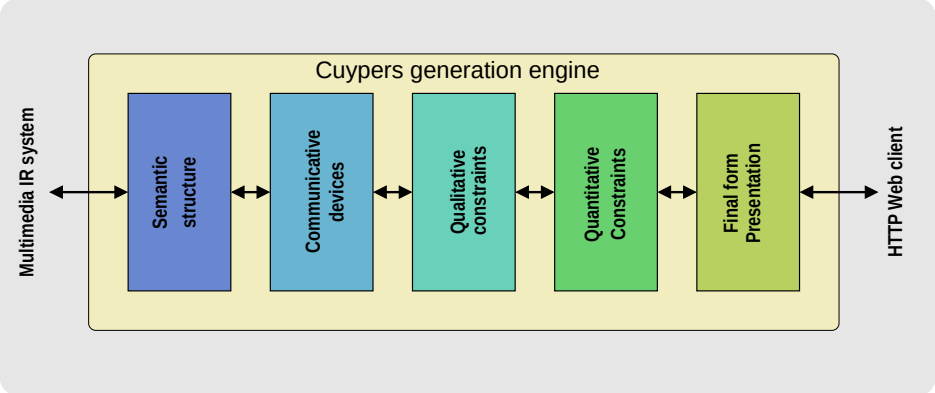

Figure 2: Levels of abstraction in the Cuypers generation engine

communicative device given in [9] is the "bookshelf", which ensures that a sequence of media items is presented in a particular order. The layout that is chosen to achieve this goal may depend on the capabilities of the target device, the preferences of the user, etc. This layout is specified using a set of qualitative constraints.

3. **Qualitative constraints** Qualitative constraints are used to specify communicative devices at a high level, as described in section 2.

4. **Quantitative constraints** Quantitative constraints make the transformation from qualitative constraints to numeric finite-domain constraints. At this level one can determine the exact position of the different media items in the presentation and whether they will fit on the screen, do not exceed a given time-limit, etc.

5. **Final form presentation** The last step is finalizing the presentation to a format suitable for the player or browser on the target device. Examples of such formats include XHTML, SVG and SMIL. Cuypers currently focuses on generating SMIL [13].

In the remainder of this section, we describe the design of the two constraint layers. For more details about the other layers see [9, 12]. As we discussed in section 2, the use of both qualitative and quantitative constraints within a system requires the translation of the qualitative constraints into quantitative constraints. To provide a clear understanding of how this translation should be done, we need to describe the domains that are involved in multimedia generation, the data structures that are needed to process the associated constraints and, most importantly, the rules that are needed to reason with them. These are detailed in the following.

3.1 Dimensions in Multimedia Generation

Multimedia presentations are built up from different media types which use different presentation dimensions:

- visual media, such as images, text and video use the two common spatial x and y dimensions;
- visual media also have a z dimension, describing the layering order of (overlapping) items;
- continuous media such as audio and video also use a temporal t dimension;
- in addition to these spatio-temporal dimensions, parts of the presentation can be connected by hyperlinks. Automatic link generation is, however, beyond the scope of this paper.

The "traditional" x, y and t dimensions are described using both quantitative and qualitative constraints. For the quantitative constraints, we use integer domains. As described in section 2, the qualitative constraints for the temporal dimension make use of the 13 interval relations specified by Allen. Each of the spatial x and y dimensions is treated similarly to the temporal dimension, so Allen's interval relations are also used for the spatial dimensions. Note that this "bounding box" approach limits the ability to model, for example, the rotation of visual items. The z dimension could, in theory, also be modeled qualitatively. There are, however, only three possible relations (A *in front of* B, A *behind* B and A *on the same level as* B), and these map directly onto the equivalent numerical constraints ($A < B, B > A, A = B$). Therefore, we currently only use numerical constraints for the z dimension.

3.2 A Data-Structure for Qualitative Multimedia Constraints

To be able to describe a multimedia presentation by a set of high level qualitative constraints, and to be able to reason with these constraints, the constraints need to be organized into an adequate datastructure that provides more information than just the constraints that apply between the media items. For example, it is often convenient to be able to explicitly model the hierarchical structure of a multimedia presentation. While this suggests the use of a tree-based datastructure, lesson learned from earlier prototypes indicate that this is too restrictive. For example, when modeling grid structures, it is convenient to be able to group the same item both vertically and horizontally. Because this is not possible in a strict tree, our data structure is based on the notion of a fully connected graph, consisting of nodes and edges.

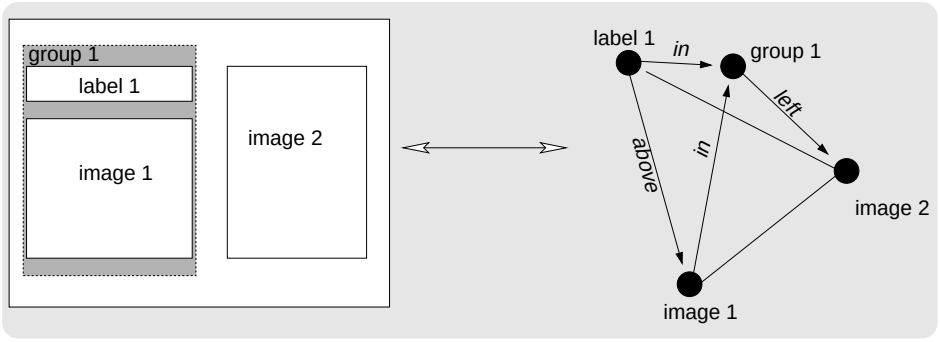

Figure 3: Graph datastructure for multimedia constraints.

In Figure 3, an example fragment of a presentation is displayed on the left, with the associated graph structure on the right. Note that only a single relation is displayed on the edges. The reverse relation and the relations representing the other dimensions have been left out for clarity. Initially all media items can have any relationship with each other. Constraints can be explicitly set, in the example the rules associated with the communicative device used have decided to group image 1 and label 1 together, to position this group left of image 2, and to position label 1 above image 1. This information can then be used to reduce the number of possibilities for the other edges in an intelligent way. The reasoning rules required for this reduction are specified using CHR. Once all reduction rules are applied and the system has not detect any inconsistencies, it picks an arbitrary value for a variable from its associated domain. The new binding of this variable and its value can trigger reduction rules once again. If the constraint solver detects an inconsistency, it backtracks over the available choices and picks another value, after which the process of applying reduction rules repeats itself. Once an inconsistency is found and there are no alternatives left, the set of constraints fails. This triggers backtracking on the Prolog level, which might result in an alternative set of constraints. As soon as

the system succeeds in binding every variable to a suitable value, these values are used to generate the associated multimedia presentation.

To build the graph structure, the transformation rules deploy two primitives: nodes and edges. In a later phase, nodes will also be used to represent the quantitative aspects of the presentation. The number of nodes for each media item depends on its type and the presentation dimensions involved. An image has 6 nodes $(x_1, y_1, x_2, y_2, t_1, t_2)$ and an audio fragment has 2 (t_1, t_2). A node can be represented in ECLiPSe as:

```
% NodeId is a structure of the form 'Id:Dimension/Nr' e.g. 'img1:x/1')
% Value is then a variable or atom associated with the x/1 domain of img1
node(NodeId,Value).
```

Note that all media items are assumed to have a unique ID. An edge relates two IDs along a particular dimension, thus representing a qualitative constraint. The number of edges between two IDs is dependent on the number of presentation dimensions they share. Between two images, there are six edges (for the x, y, t dimensions and their inverses) while between an audio and image there are only two (t with an inverse). An edge is represented in Cuypers as:

```
% IdX are the Ids of the associated media items
% Dimension is the label of the edge x,y or t
% Value is a variable or an atom associated with the domain of Dimension
edge(IdA,IdB,Dimension,Value).
```

To simplify the building of a consistent graph, a set of CHR rules specifies the relationship between nodes and edges. Whenever an edge is asserted, the system looks for the associated nodes. When these are not yet defined, the system introduces them automatically, according to the associated CHR rules. For example, for the temporal dimension a CHR rule relating temporal nodes with edges may be specified as:

```
% Node introduction
edge(IdA,IdB,t, _Value)             % Temporal
==>
        node(IdA:t/1,VA1),          % introduce start-node t1
        node(IdA:t/2,VA2).          % introduce end-node t2
```

When the initial graph is built, typically the qualitative relationships of only a few edges have been fully determined. The set of CHR rules that allow the system to infer possible values for the remaining edges. will be defined below.

3.3 CHR Rules for Qualitative Multimedia Constraints

In order to reduce the domains of the variables in the remaining edges in an intelligent way, we need specific rules which state how and when these reductions should be invoked. These rules range from optimization rules to rules that are needed for reasoning over spatio-temporal dimensions. For example, both application of the Prolog transformation rules and application of the domain reduction rules might result in duplicate edges. Accordingly, we need a rule that specifies that when two edges are identical, one can be omitted:

```
% Equality
edge(IdA,IdB,Dimension,Value), edge(IdA,IdB,Dimension,Value)
<=>                                 % substitute for
    edge(IdA,IdB,Dimension,Value).
```

Another rule is needed to specify that if two edges between the same two nodes have the same dimension, then their values should also be the same:

```
% Unification
edge(IdA,IdB,Dimension,ValueA), edge(IdA,IdB,Dimension,ValueB)
==>                                 % implies
    ValueA = ValueB.
```

We also need to be able to deal with inverse relations. If there is a (directed) edge between two nodes A and B with a certain dimension and a certain value for a relation, then there is also an edge between nodes B and A with the same dimension but with an inverse relation:

```
% Inverse
edge(IdA,IdB,Dimension,Value)
==>                                % implies
    inverse(Value,RValue)          % only when relation has inverse
    |
    edge(IdB,IdA,Dimension,RValue).
```

Further rules are needed to specify the transitive behavior of many of the Allen relations. For example, if there is an edge between node A and node B (on dimension x, y or t) and there is an edge between node B and C with value 'b' (before) then there is also an edge between A and C with value 'b':

```
% Transitivity
edge(IdA,IdB,Dimension,b), edge(IdB,IdC,Dimension,b)
==>                                % implies
    allenDim(Dimension)            % only for x,y and t domain
    |
    edge(IdA,IdC,Dimension,b).
```

As discussed in section 2, it is not practical to specify the transitive behavior of all 13×13 combinations in this manner. Therefore, the rule below defers the implementation of the transitivity rules to the predicate `tr`, and rule generation tools of Apt and Monfroy have been used to define the truth values of all possible combinations of variables of this predicate.

```
% Transitivity (dim x,y,t)
edge(IdA,IdB,Dimension,ValueAB), edge(IdB,IdC,Dimension,ValueBC)
==>                                % implies
    allenDim(label),               % only for x,y and t domain
    A \= B, A \= C, B \= C         % A, B and C are different nodes
    |
    tr(ValueAB,ValueBC,ValueAC),   % rule automatically generated
                                   % by Apt et al.
    edge(A,C, Dimension ,ValueAC).  % add the deduced constraint
```

Using these CHR rules we can, on the basis of the edges explicitly defined by the transformation rules, reduce the domains of the remaining edges. This allows us to define higher level constraints. For example, we can define a "meta-constraint" that specifies that if a label is positioned above an image, then all other labels should also be placed consistently above their corresponding images. However, a solved set of qualitative constraints in itself is insufficient. Quantitative constraints are needed to check whether a certain set of qualitative constraints does not violate some hard quantitative requirements such as the target platform's screen size or the maximum duration of the presentation. Below, we define a set of CHR rules that maps the qualitative constraint graph to a set of quantitative constraints.

3.4 CHR Rules for Translating Qualitative to Quantitative Constraints

Most of the quantitative constraints can be generated by mapping the qualitative Allen relations to numerical constraints. The qualitative Allen relations can, for each of the three dimensions x, y, and t, be mapped to one or more corresponding numeric constraints. Figure 4 on the following page defines this mapping for the x dimension, using X_1 as the left coordinate and X_2 as the right. Similar mappings can be defined for the other dimensions. Given these mappings, we can define CHR rules to map all qualitative (Allen) constraints into numeric constraints that can be solved by ECLiPSe built-in libraries. For example, we can define the translation rules for *meet* ("b") in CHR as follows:[2]

[2]Note the use of the hashmark to discriminate equality constraints (# =) from the usual equality operator in Prolog (=).

Allen relation	Quantitative constraints
A before B	$X_2^A < X_1^B$
A during B	$X_1^A > X_1^B,\ X_2^A < X_2^B$
A overlaps B	$X_1^A < X_2^B,\ X_2^A > X_1^B,\ X_2^A < X_2^B$
A meets B	$X_2^A = X_1^B$
A starts B	$X_1^A = X_1^B,\ X_2^A - X_1^A < X_2^B - X_1^B$
A finishes B	$X_2^A = X_2^B,\ X_2^A - X_1^A < X_2^B - X_1^B$
A equals B	$X_1^A = X_1^B,\ X_2^A = X_2^B$

Figure 4: Allen's relations expressed using quantitative constraints.

```
% transform qualitative to quantitative
edge(IdA,IdB,Dimension,m),                  % A meets B
node(IdA:Dimension/2,CoordinateA),          % A.coordinate2
node(IdB:Dimension/1,CoordinateB)           % B.coordinate1.
==>
meets(CoordinateA, CoordinateB).


% meets -> the second coordinate of A should be equal to the
%      first coordinate of B
meets(CoordinateA2,CoordinateB1)
==>                                         % implies
    integers([CoordinateA2,CoordinateB]1), % only integers
    |
    CoordinateA2 #= CoordinateB1            % A2 equals B1
```

The Allen relations *before*, *during* and *overlap* require an extra input parameter to define the corresponding quantitative constraints. The relation between two objects A and B is not entirely defined by its X_1 and X_2 coordinates. The missing parameter is the padding-distance. This is the distance between two of the coordinates of the A and B object. The definition depends on the type of relation. The padding between a *before* relation is defined to be the difference of its X_1^B and X_2^A coordinates. The padding between a *during* relation is defined to be the difference between X_1^A and X_1^B and the padding for the overlap relation is defined to be the difference between X_2^A and X_1^B.

```
before(CoordinateA2,CoordinateB1, Padding)
==>                                         % implies
    integers([CoordinateA2,CoordinateB1, Padding]),   % only integers
    |
    CoordinateA2 #< CoordinateB1            % A before B
    Padding #= CoordinateB1 - CoordinateA2 % P = B1 - A2
```

The padding between two objects is added as a constraint to the constraint store. This enables a convenient way of constraining distances between media items in a consistent way, *e.g.*, ensuring the padding between an enumeration of media items to be the same:

```
edge(A,B,Dim,b),
node(A:Dim/2,AD2),
node(B:Dim/1,BD1)
==>                                         % implies
    BD1 - AD2 #= P,                         % calculate padding
    padding(A,B,P),                         % add padding constraint
    before(AD2,BD1,P).                      % -> enough information to specify
```

```
                                        %    qualitative constraint
```

Relations, such as *during*, *starts* and $finishes$, require certain constraints on the length of the items. There is an optimization possible if we enable a combination of qualitative and quantitative domains. We know, for example, that an "A during B" constraint will never be met if the length of A exceeds the length of B:

```
edge(A,B,x,d), width(A,WA), width(B,WB)    % suppose widths are known
<=>
    ground(WA),ground(WB),                 % WA and WB are known
    WA >= WB                               % WA >= WB
    |
    false.                                 % this constraint is not feasible.

edge(A,B,x,d), width(A,WA), width(B,WB)
==>
    (nonground(WA);nonground(WB))          % either WA or WB is unknown
    |
    WA #< WB.                              % constraint: length of A should be
                                           % smaller than the length of B
```

Finally, grouping enables the user to define constraints on a group of nodes in a convenient way. Instead of stating every relation between a target object and the members of the group they can be stated by defining a group containing all those members and a constraint between the group and the target object. In practice a group is a node which is not bound to a media item. All its members are placed within this node by means of 'during' constraints in the x,y and t dimensions. A group forms a 'bounding box' around its members. The exact width, height, duration and position of a group are not known until its members are sufficiently instantiated.

4. Related Work

Generation of synchronized multimedia presentations from higher level descriptions is not novel in itself. Spatial and temporal constraints for specifying multimedia are used, for example, in the Madeus system [6]. The common architecture of a number of model-based systems for multimedia presentations developed within the AI community resulted in the Standard Reference Model for Intelligent Multimedia Presentation Systems (SRM-IMMPS) [5], and the relation between SRM-IMMPS with a previous prototype of the system presented here has been described in [10].

Our work is also closely related to the work of Elisabeth André, who described the use of AI planning techniques in combination with constraint solvers in her WIP and PPP systems [2]. The main contribution of our approach is that it integrates the several processing steps into a single run time environment so that the system can freely backtrack across the different levels. This allows high-level presentation decisions to be re-evaluated as a result of constraints that turn out to be insolvable at the lower levels (*e.g.* pixel level). Nevertheless, the individual levels remain conceptually separated, which allows the definition of small, declarative design rules instead of the single hierarchy of planning operators used by André.

5. Conclusions and Future Work

This paper shows how constraints and constraint logic programming can be beneficial to the automatic generation of multimedia presentations. We demonstrated that the generation process consists of several conceptual layers transforming a high level semantic description of the presentation through different stages into a playable multimedia presentation. Constraints are applied to create an abstraction layer which simplifies the specification of the transformation rules while conserving the ability to output a fully specified final form multimedia presentation. We argued that both qualitative and quantitative constraints are needed in order to generate a presentation. Quantitative constraints form a first abstraction above the final-form presentation, and allow for (minor) adaptation of a presentation to different screen sizes etc. A solved set of constraints ensures that quantitative measures,

such as maximum screen size and maximum duration, are not violated. Qualitative constraints form a higher-level specification of the layout of a presentation. They allow for a wider range of adaptation, and also allow for reasoning over the results by specifying "meta constraints", such as consistency constraints. They also make the generation process more efficient, because of their small domains, and allow higher level structures to be used in the final form specification language. While the domain reduction rules needed to reason with qualitative constraints can be expressed in CHR, additional algorithms and tools support are needed to generate the large number of rules required.

The Cuypers system we are currently developing makes use of the techniques presented here. It chooses the first presentation which satisfies all constraints as the one that is presented to the user. In most cases this is, from an author's perspective, unlikely to be the best one. We are currently investigating methods of choosing among the possible alternatives on the basis of some predefined evaluation function that reflects the "quality" of the generated presentation. What are good quality measures for multimedia presentations and how can we formalize them are further points of research.

Acknowledgements
Part of the research described here has been funded by the European ITEA/RTIPA and the Dutch Dynamo and ToKeN2000 projects. Krzysztof Apt provided many valuable insights on the use of qualitative constraints, Eric Monfroy developed the rule generation software.

References

1. James F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–844, November 1983.
2. Elisabeth André. WIP and PPP: A Comparison of two Multimedia Presentation Systems in Terms of the Standard Reference Model. *Computer Standards & Interfaces*, 18(6-7):555–564, December 1997.
3. K.R. Apt and E. Monfroy. "Automatic generation of constraint propagation algorithms for small finite domains". In *Proc. of the 5th International Conference on Principles and Practice of Constraint Programming (CP'99)*, pages 58–72. Springer-Verlag, 1999. Lecture Notes in Computer Science 1713.
4. G.J. Badros and A. Borning. Cassowary: A Constraint Solving Toolkit, 1999.
5. M. Bordegoni, G. Faconti, M.T. Maybury, T. Rist, S. Ruggieri, P. Trahanias, and M. Wilson. A Standard Reference Model for Intelligent Multimedia Presentation Systems. *Computer Standards & Interfaces*, 18(6-7):477–496, December 1997.
6. M. Jourdan, N. Layaïda, C. Roisin, L. Sabry-Ismaïl, and L. Tardif. Madeus, an Authoring Environment for Interactive Multimedia Documents. In *Proceedings of ACM Multimedia '98*, Bristol, UK, 1998.
7. William C. Mann, Christian M. I. M. Matthiesen, and Sandara A. Thompson. Rhetorical Structure Theory and Text Analysis. Technical Report ISI/RR-89-242, Information Sciences Institute, University of Southern California, November 1989.
8. Lloyd Rutledge, Brian Bailey, Jacco van Ossenbruggen, Lynda Hardman, and Joost Geurts. Generating Presentation Constraints from Rhetorical Structure. In *Proceedings of the 11th ACM conference on Hypertext and Hypermedia*, pages 19–28, San Antonio, Texas, USA, May 30 – June 3, 2000. ACM.
9. Lloyd Rutledge, Jim Davis, Jacco van Ossenbruggen, and Lynda Hardman. Inter-dimensional Hypermedia Communicative Devices for Rhetorical Structure. In *Proceedings of International Conference on Multimedia Modeling 2000 (MMM00)*, pages 89–105, Nagano, Japan, November 13-15, 2000.
10. Lloyd Rutledge, Lynda Hardman, Jacco van Ossenbruggen, and Dick C.A. Bulterman. Implementing Adaptability in the Standard Reference Model for Intelligent Multimedia Presentation

Systems. In *The International Conference on Multimedia Modeling*, pages 12–20, 12-15 October 1998.

11. Jacco van Ossenbruggen. *Processing Structured Hypermedia — A Matter of Style*. PhD thesis, Vrije Universiteit, Amsterdam, The Netherlands, April 10, 2001. Also available on http://www.cwi.nl/∼jrvosse/thesis/.
12. Jacco van Ossenbruggen, Joost Geurts, Frank Cornelissen, Lloyd Rutledge, and Lynda Hardman. Towards Second and Third Generation Web-Based Multimedia. In *The Tenth International World Wide Web Conference*, pages 479–488, Hong Kong, May 1-5, 2001. IW3C2. This is a revised version of CWI technical report INS-R0025.
13. W3C. Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. W3C Recommendations are available at http://www.w3.org/TR/, June 15, 1998. Edited by Philipp Hoschka.
14. Mark Wallace, Stefano Novello, and Joachim Schimpf. ECLiPSe: A Platform for Constraint Logic Programming, 1997.