

Constraints for *Multi*media Presentation Generation

Joost Geurts

22nd January 2002

Universiteit van Amsterdam

Constraints for Multimedia Presentation Generation

Doctoraal scriptie Kunstmatige Intelligentie, specialisatie Logica

Begeleiders:

Prof.dr. K.R. Apt , CWI, Universiteit van Amsterdam

Prof.dr. H.L. Hardman, CWI, Technische Universiteit Eindhoven

Dr. J.R. van Ossenbruggen, CWI

door

Joost Geurts

Keywords: Multimedia, Constraints, CLP, CHR.

Acknowledgments

Many people contributed in one way or the other to the development of this thesis. In this section I'd like to specially thank a few.

First of all, Lynda Hardman for giving me the opportunity to work in a fascinating environment for the past three years and, moreover to stimulate me in developing my own thoughts and ideas (mostly about chocolate and juggling).

Krzysztof Apt for introducing me to the interesting subject of constraint programming and for his patience and valuable comments while writing the thesis.

Jacco van Ossenbruggen, without his help I could not have written the thesis as it is now. Through the numerous discussions we had, he taught me more about multimedia and research in general than I could ever have wished for.

My (other) colleagues at CWI Lloyd, Frank, Steven, Oscar, Karolina, Marcos and Stefano for their valuable comments and the 'gezellige' coffee-breaks. "Eetgezelschap de Diemerboyz" for a wonderful time and not letting me die of starvation.

The Rijksmuseum Amsterdam for the images. RTIPA and ToKeN2000 for funding the research.

Finally I'd like to thank my friends and family in Twente for their continuous support.

Joost Geurts, Amsterdam 2002

Abstract

Automatic multimedia presentation generation is applicable in a wide variety of circumstances because of its ability to adapt to different presentation contexts such as hardware platforms, user expertise and user interest.

The process of generating a multimedia presentation takes a semantic description of the content and transforms this through a number of steps into a multimedia presentation tailored to the requirements of the individual user. This is a complex process which requires a system able to make trade-offs between the different presentation dimensions.

This report shows that the use of constraints and constraint logic programming can be beneficial to the process of automatic presentation generation because of its ability to integrate different conceptual layers into one single execution environment. This approach enables an efficient revision of earlier made choices at any stage in the process. We introduce two types of constraints — quantitative and qualitative — and discuss what type of constraint can be beneficial at what stage.

Contents

1	Introduction	1
1.1	Problem statement	1
1.2	Scope and contribution	3
1.3	Outline	3
2	Background: Multimedia Presentation Generation	5
2.1	Introduction	5
2.2	The Cuypers Architecture	6
2.2.1	Levels of Abstractions in Cuypers	6
2.2.2	Example scenario	9
2.2.3	Media in Cuypers	12
2.2.4	Implementation	14
2.2.5	Related work	14
2.3	Conclusion	15
3	Background: Constraint Satisfaction Problems	16
3.1	Introduction	16
3.2	Constraint Programming	17
3.3	Constraint Propagation	18
3.4	Local Consistency Notions	19
3.5	Constraint Handling Rules	22
3.5.1	Example	23
3.6	Conclusion	23
4	Constraints for Multimedia	24
4.1	Introduction – example scenario	24
4.2	Quantitative constraints	26
4.2.1	Quantitative constraints and CLP	28
4.2.2	Drawbacks in quantitative constraint processing	30
4.3	Qualitative constraints	32
4.3.1	Reasoning with Qualitative constraints	34
4.4	Use of constraints within Cuypers	36
4.5	Discussion	39
4.5.1	Labeling — choice of candidate variable to label	39
4.5.2	Labeling — choice of candidate value	40
4.5.3	Labeling — best-first versus depth-first	41

4.5.4	Labeling — structural grouping	41
4.5.5	Redundancy of the Transitive reasoning rule	42
4.5.6	Allen’s temporal relations	43
4.5.7	Generalized Propagation — Propia	43
4.5.8	Performance	43
4.6	Conclusion	44
5	Conclusion	45
A	CHR rules	47
A.1	Domain Reduction Rules	47
A.2	Translation rules	47
B	Summary of Cuypers	49

Chapter 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 point of view, 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, the presentation should be conveyed in the way most suited to a particular user. Manually authoring all these different presentations is 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.

1.1 Problem statement

Automatic presentation generation is an alternative to manual authoring. Just as an author of a multimedia presentation faces problems concerning size, style, intended audience etc. so does a system automatically generating one. The system should be made aware of the different possibilities available and how certain choices can influence the number of open alternatives. We differentiate between a possible presentation generation space along three axes. Each of these axes represents a factor which influences the final presentation.

Semantic Multimedia presentations try to convey a message. The first stage of authoring a multimedia document is to identify this message and come up with a narrative structure to communicate to the user. Multimedia authors at this point typically make use of story boards where they make rough sketches of the different scenes in the presentation with the main purpose to clarify what and when information is presented. The result of this stage is a structure which abstractly defines the media used in the presentation and moreover in what order to produce a story which logically makes sense.

Design The design of a presentation is primarily intended to convey the identified semantic structure of the presentation by means of design constructs. Typically, the process of defining a narrative structure is intertwined with the design of a conceptual lay-out [6]. Because design is used to express semantics, consistency is an important aspect. Besides the semantical importance of design, aesthetic properties, such as style, are in practice an influential factor too.

Resources Within multimedia design the specific technical limitations of the target platform are of great importance. Nowadays, devices which are able to play multimedia

presentations range from cellular phones to fully equipped PCs. An author of a multimedia presentation has to take these specific limitations into account and adapt the presentation if the requirements cannot be met. Besides technical limitations, the user can influence the possible resources as well. For example, the user could have a limited amount of time or have specific abilities.

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 needs 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 within the screen dimensions of the user’s device and time-limits of the user. Solutions are possible, but only after potential contradictory requirements have been traded-off against one another. Examples of such trade-offs include:

Semantics — 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 — Resources 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 — Resources The message itself may also have to be compromised by the availability of resources. Whether this is caused by the temporal constraints of the user (the wish to view the message in as short a time as possible) or space constraints of the device, a consequence may be that the message itself has to be compressed. For example, by discarding media items that would have served to illustrate a particular concept, or even by removing the concept from those that were to be explained.

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 concerning 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.

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 in a automatic presentation generation system. From an implementor’s perspective, the system has to make the decisions an author would make, and also cope with the trade-offs that have to be made when chosen solutions fail in practice.

Our approach for creating the prototype multimedia presentation generation system, discussed in chapter 2, is to identify a number of different levels of abstraction 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.

1.2 Scope and contribution

Automatic multimedia generation is a broad topic. It integrates a number of disciplines including databases, document processing, real time network delivery, knowledge engineering and more. During the process the presentation is transformed from completely abstract, consisting only of high level semantic/rhetoric descriptions, to a concrete final form presentation.

We will argue that this is not a linear process, but will consist of going back and forth through different presentation abstraction layers. Because of this non-linearity we claim a declarative approach has advantages compared to traditional imperative techniques.

In this thesis we focus on the use of constraints and constraint logic programming in order to automatically generate a multimedia presentation. We will demonstrate that two types of constraints are needed. Quantitative constraints 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 have to be defined explicitly.

We implemented both types of constraints in Cuypers[28, 27], our prototype multimedia generation system. This resulted in specific insights concerning labeling of uninstantiated variables, grouping and the purpose of the transitive reasoning rules.

1.3 Outline

The remainder of this thesis is structured as follows:

Chapter 2 will explain multimedia in combination with multimedia presentation generation.

We show relations to previous work done in the field of document processing and AI presentation generation. Furthermore we outline the architecture of Cuypers, our prototype multimedia generation engine, which transforms a high-level semantic description of a multimedia presentation into a final form presentation.

Chapter 3 will describe the underlying theory of constraints, and constraint logic programming in particular. We explain different notions of consistency in constraint satisfaction problems. Furthermore we elaborate on the specification of user defined constraints.

Chapter 4 contains the results we found during our research. We show how multimedia presentation can be automatically generated by using the constraint logic programming paradigm. We introduce two types of constraints, quantitative and qualitative and discuss in what circumstances they are needed. Furthermore we discuss some of the problems we encountered during the development of Cuypers.

Chapter 2

Background: Multimedia Presentation Generation*

Multimedia is a term used to denote a collection of different types of media such as text, images, audio and video. A multimedia presentation is the run time manifestation of multiple media items where semantic relations are expressed through temporal and/or spatial synchronization. Multimedia presentations are represented by multimedia documents which encode the relations between the different media items. There are different languages to encode a presentation and, within a language, there are a number of ways to represent the presentation. The requirements for a multimedia document specification language is a topic which has been researched the past few years, resulting in a formal defined model for multimedia presentations: “the Amsterdam hypermedia model”([15]).¹ This model formed the basis of the Synchronized Multimedia Integration Language SMIL[29] which is a W3C recommendation for multimedia documents since 1998. Recently SMIL2.0[31] was released which extends the existing basic SMIL specification with a large number of multimedia features, such as animation and a more advanced event-driven timing model.

In this chapter we elaborate on the specific properties multimedia and multimedia presentations have. We focus on the differences between text based documents and multimedia presentations and argue that for multimedia transformation a more sophisticated transformation model is needed. Cuypers, our multimedia generation engine, addresses these issues and we explain the architecture and abstraction layers of it.

2.1 Introduction

In text based document models, such as HTML, the structure of the document is separated from the visual style. This way, a document can be adapted to be viewed on systems which have different properties or be displayed according to stylistic preferences of the user. This is possible since, for text, we have an established set of complicated but well understood algorithms [19] that can be used to automatically typeset a text according to the requirements of a given layout specification. To keep the style sheet itself as declarative as possible, the components implementing these relatively low level and detailed algorithms are typically

*adopted from van Ossenbruggen [28] and Geurts [14]

¹The Amsterdam hypermedia model is a model for hypermedia documents, hypermedia is multimedia extended with hyperlinking.

part of the style engine’s back-end application [16, 10]. These back-end components typically implement kerning, hyphenation, justification, and line and page breaking algorithms. Note that these algorithms are based on the linear structure of the underlying text. Since multimedia documents are not based on such a text flow, these algorithms do not suffice for formatting multimedia documents. In order to adapt a multimedia document to specific circumstances, other adaptation techniques and strategies need to be developed. The Cuypers system addresses these issues.

2.2 The Cuypers Architecture

Cuypers is a research prototype system, developed to experiment with the generation of Web-based presentations as an interface to semi-structured multimedia databases. Cuypers addresses the issue discussed in the introduction. First of all, it explores a set of abstractions, both on the document and on the presentation level, that are geared towards interactive, time-based and media centric presentations rather than presentations that are based on text-flow. Second, it uses a set of easily extensible transformation rules specified in Prolog, exploiting Prolog’s built-in support for backtracking. Finally, it facilitates easy feedback between the higher and lower level parts of the transformation process by executing both within the same environment. Instead of a strict separation between the transformation engine and the constraint solver, our system uses a constraint solver embedded in Prolog, so the system is able to backtrack when the transformation process generates a insolvable constraint.

Cuypers operates in the context of the environment depicted in Figure 2.1 on the facing page. It assumes a server-side environment containing a multimedia database management system, an intelligent multimedia IR retrieval system, the Cuypers generation engine itself, an off-the-shelf HTTP server, and — optionally — an off-the-shelf streaming media server. At the client-side, a standard Web client suffices. The focus of this section is the Cuypers generation engine. Given a rhetorical (or other type of semantic) structure and a set of design rules, the system generates a presentation that adheres to the limitations of the target platform and supports the user’s preferences.

2.2.1 Levels of Abstractions in Cuypers

The experience gained from the development of earlier prototypes (e.g. work done by Bailey [24]) however, proved that for most applications, the conceptual gap between an abstract, presentation-independent document structure and a fully-fledged, final-form multimedia presentation is too big to be specified by a single, direct transformation. Instead, we take an incremental approach, and define the total transformation in terms of smaller steps, which each perform a transformation to another level of abstraction. These levels are depicted in Figure 2.2 and include the semantic, communicative device, qualitative constraint, quantitative constraint and final-form presentation levels, respectively.

Below, we describe each abstraction level and why it is needed in the overall process. We take a bottom-up approach and start with the final-form presentation level, which is the level that describes the presentation as it is delivered to the client’s browser. This is also the level readers will be most familiar with, since it describes documents on the level of their encoding in, for example, HTML [30], SMIL [31] or SVG [12].

We subsequently add more abstraction levels, and end with the highest level, the ”semantic level”, which completely abstracts from all layout and style related information.

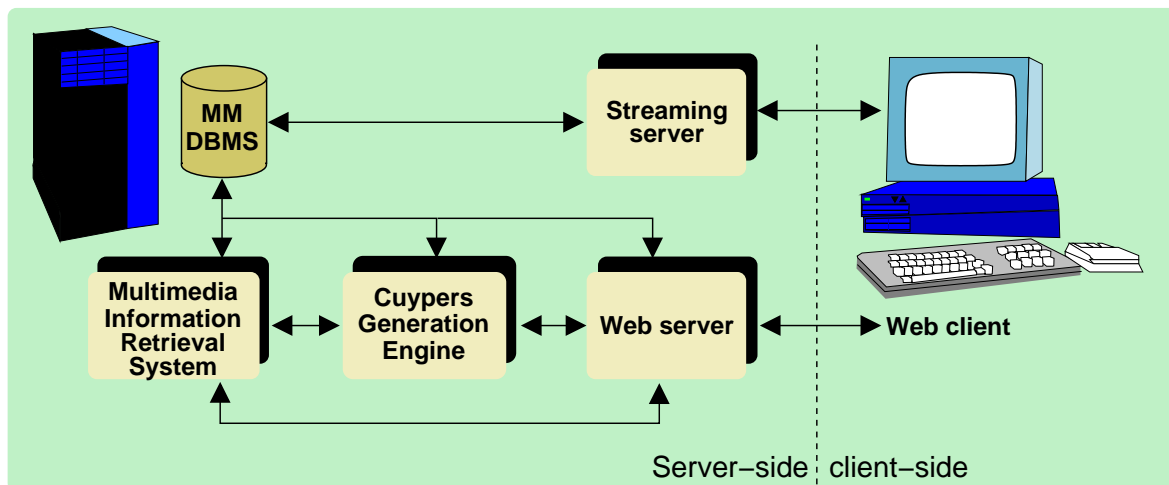


Figure 2.1: The environment of the Cuypers generation engine

1. **Final-form presentation level** At the lowest level of abstraction, we define the final-form presentation, which encodes the presentation in a document format that is readily playable by the end user’s Web browser or media player. Examples of such formats include, HTML, SVG, and — the focus of our current prototype — SMIL. This level is needed to make sure that the end-user’s Web-client remains independent of the abstractions we use internally in the Cuypers system, and to make sure that the end-user can use off-the-shelf Web clients to view the presentations generated by our system.
2. **Quantitative constraints level** To be able to generate presentations of the same information using different document formats, we need to abstract from the final-form presentation. On this level of abstraction, the desired temporal and spatial layout of the presentation is specified by a set of format-independent constraints, from which the final-form layout can be derived automatically (see chapter 4).
3. **Qualitative constraints level** An example of a qualitative constraint is “caption X is positioned below picture Y”, and backtracking to produce alternatives might involve trying right or above, etc. Some final-form formats allow specification of the document on this level. In these cases, the Cuypers system only generates and solves the associated numeric constraints to check whether the presentation can be realized at all, it subsequently discards the solution of the constraint solver and uses the qualitative constraints directly to generate the final form output (see chapter 4).

While qualitative constraints solve many of the problems associated with quantitative constraints, they are still not suited for dealing with the relatively large differences in layout. Therefore, another level of abstraction is introduced: the communicative device.

4. **Communicative device level** The highest level of abstraction describing the presentation’s layout makes use of *communicative devices* [25]. These are similar to the patterns of multimedia and hypermedia interface design described by [23] in that they describe the presentation in terms of well known spatial, temporal and hyperlink presentation constructs. An example of a communicative device described in [25] is the

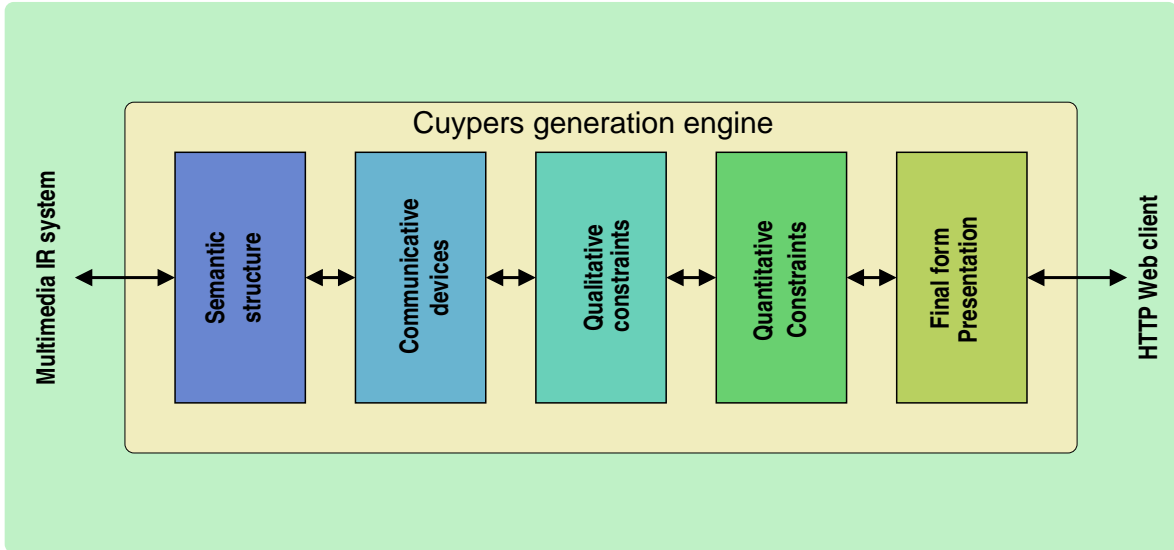


Figure 2.2: The layers of the Cuypers generation engine.

bookshelf. This device can be effectively used in multimedia presentations to present a sequence of media items, especially when it is important to communicate the *order* of the media items in the sequence. How the bookshelf determines the precise layout of a given presentation in terms of lower level constraints can depend on a number of issues. For example, depending on the cultural background of the user, it may order a sequence of images from left to right, top to bottom or *vice versa*. Also its *overflow strategy*, that is, what to do if there are too many images to fit on the screen, may depend on the preferences of the user and/or author of the document. It may decide to add a "More info" hyperlink to the remaining content in HTML, alternatively, it could split the presentation up in multiple scenes that are sequentially scheduled over time in SMIL.

Note that communicative devices, including the one described above, typically deal with layout strategies that involve multiple dimensions (including space, time and linking), while the constraints discussed above typically do not cross the boundaries of a single dimension. Constraints using variables along more than one dimension are called cross-dimensional constraints, and have previously been discussed in [24].

While the communicative device level is a very high-level description of the presentation, we still need a bridge from the domain-dependent semantics as stored in the multimedia information retrieval system to the high-level hypermedia presentation devices. To solve this problem, we introduce one last level of abstraction: the semantic structure level.

5. **Semantic structure level** This level completely abstracts from the presentation's layout and hyperlink navigation structure and describes the presentation purely in terms of higher-level, "semantic" relationships. Typical examples include the presentation's narrative structure or rhetorical structure. In the current Cuypers system, we focus on the rhetorical aspects of the presentation, because it applies well to the application domains we are currently building prototypes for which (e.g. generating multimedia

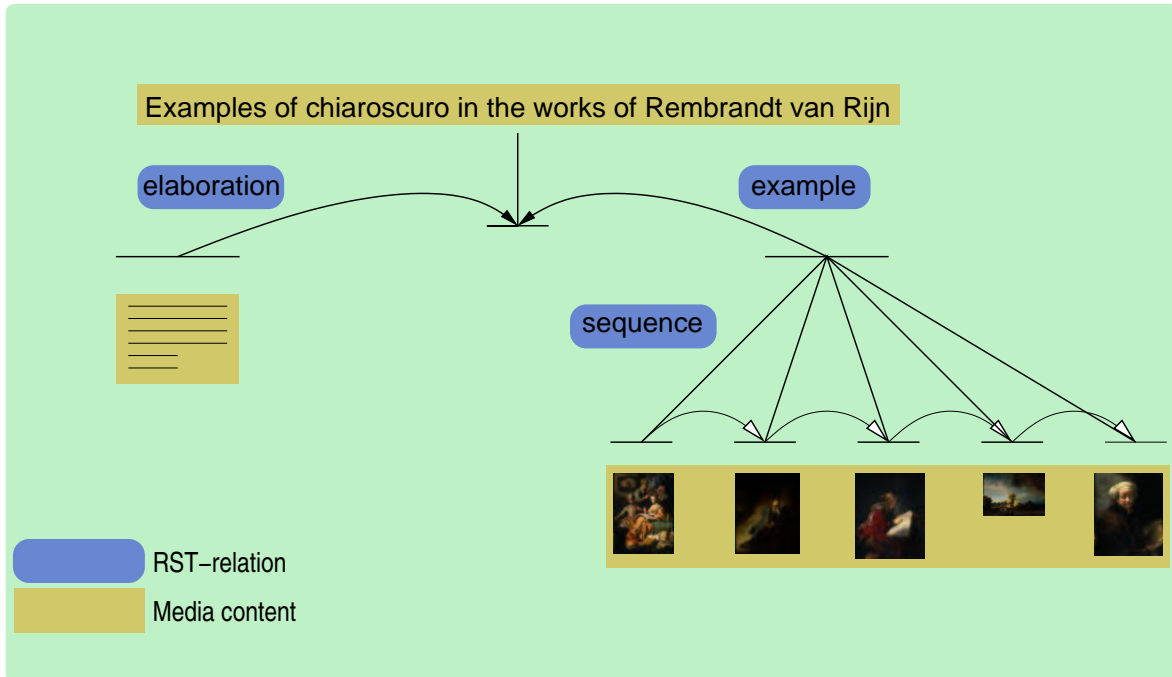


Figure 2.3: RST tree representation of the input.

descriptions of artwork for educational purposes).

The subdivision of the generation process in Cuypers is based on these different levels, with an extensible set of transformation rules to move from one level to another (see the section on implementation below). In practice, however, the transformations work by backtracking up and down different levels, and transformation rules may have access to information from several steps earlier. To explain the different abstraction levels and the associated transformation steps, the next section uses an example scenario to illustrate all levels.

2.2.2 Example scenario

In this section, we use an example scenario where the user (studying art history) just asked the system to explain the use of the *chiaroscuro* technique (strong contrast of light and dark shading) in the paintings of Rembrandt van Rijn. The system's multimedia information retrieval back-end queried its annotated multimedia database system and retrieved five images of paintings that are annotated as using this technique, the accompanying titles and a general textual description of the term *chiaroscuro*.

Semantic level: rhetorical structure

A presentation is constructed around the concept "Examples of Chiaroscuro in the works of Rembrandt van Rijn", using the images as *examples* of the the core concept, and the text as an *elaboration* of the core concept. Additionally, to preserve the ordering of the time the picture was made, the five images are shown in a *sequence* relation. The resulting

```

<!DOCTYPE presentation PUBLIC "-//CWI/DTD Rhetorics 1.0//EN" "rhetoric.dtd">
<presentation xmlns="http://www.cwi.nl/~media/ns/cuyppers/rhetoric">
  <media id="title" ... refs to content/metadata database .../>
  <media id="img1" ... />
  <media id="img2" ... />
  <media id="img3" ... />
  <media id="chiaroscuro" ... />
  <nsRelation>
    <nucleus>
      <mediaref idref="title"/>
    </nucleus>
    <satellite type="example">
      <mnRelation type="sequence">
        <nucleus>
          <mediaref idref="img1"/>
        </nucleus>
        ... ..
        <nucleus>
          <mediaref idref="img5"/>
        </nucleus>
      </mnRelation>
    </satellite>
    <satellite type="elaboration">
      <mediaref idref="chiaroscuro"/>
    </satellite>
  </nsRelation>
</presentation>

```

Figure 2.4: XML encoding of the presentation's rhetorical structure

tree structure, using the notation common in Rhetorical Structure Theory [21] is shown in Figure 2.3 (the titles of the individual paintings have been omitted for brevity). Note that while the generation of this structure requires advanced knowledge of the domain, the organization of the multimedia database, the user and this task, this generation process is carried out by the server's multimedia information retrieval system, which is considered to be beyond the scope of this paper. Here, we focus on the Cuyppers presentation engine, and assume the RST structure as the input given to the engine. The input is encoded using a simple XML Schema to represent RST nucleus/satellite relations. The associated XML fragment is shown in Figure 2.4.

High-level presentation semantics: communicative devices

Note that the rhetorical structure given in Figure 2.3 contains no information about the spatio-temporal layout of the final-form presentation. This information is incrementally added by the Cuyppers system, based on general design knowledge, combined with knowledge about the underlying domain (e.g. "17th century painting"), the task and preferences of the end-user and the capabilities of the device that is used to access the Web. In the first step, the input is matched against a set of rules designed to convert the input to a communicative device hierarchy. Note that this is purely a design decision: in practice, designers of a particular application will need to extend the default rule set that comes with the Cuyppers system.

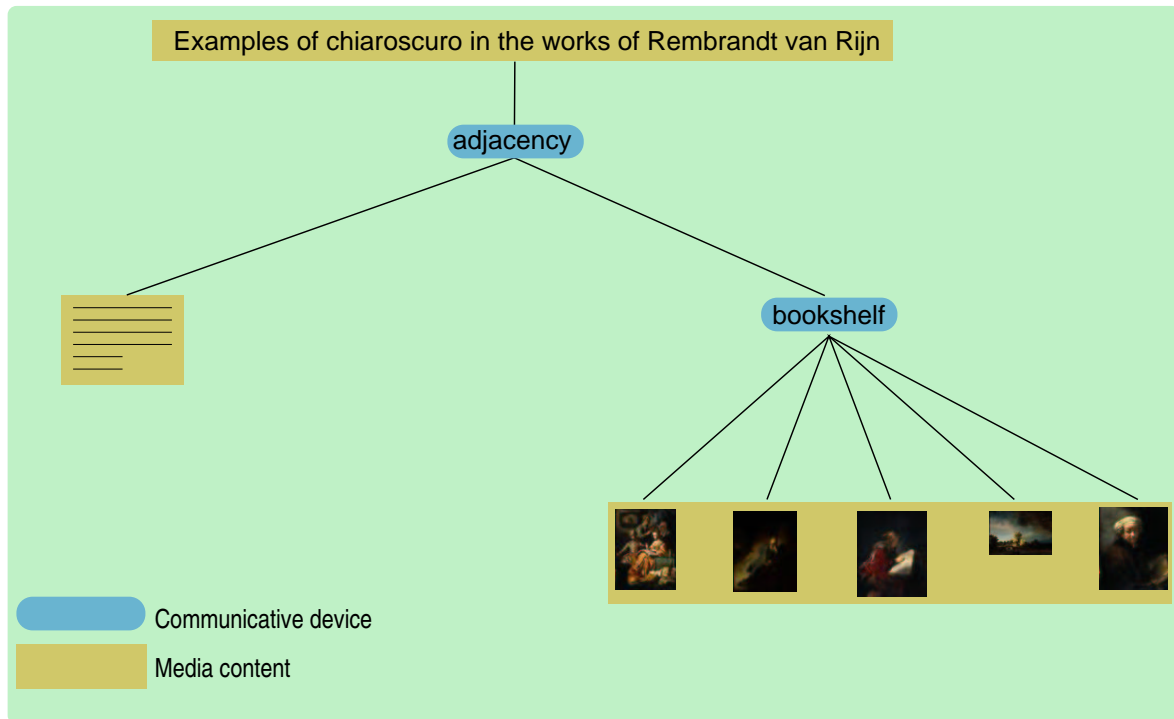


Figure 2.5: Example communicative device hierarchy.

In the example above, the rules that match the input RST structure could, for example, map the root nucleus (the label "Chiaroscuro by Rembrandt van Rijn") to the title of the presentation. In addition, the rules determine that the title, elaborative text and the example section should be visible at the same time, as close to another as possible. This is used by grouping these elements in a communicative device named *spatial adjacency* [25]. Because the example section itself consisted of a sequence of images of which the ordering should be preserved, the sequence is mapped to a communicative device named *bookshelf*. The bookshelf's layout strategy is parameterized, in this case the strategy is to try to achieve a left-to-right, top-to-bottom ordering first, and to use a temporal overflow strategy when it proves impossible to fit all images on a single screen. The resulting hierarchy is sketched in Figure 2.5.

Constraint level

While the communicative device hierarchy described above reflects the most basic design decisions about the way the document should be communicated to the user, the mutual relationships among the media items have not been established. How this could be realised is the topic of chapter 4.

Final-form generation

In the last step, the information accumulated in the previous steps is used to generate the final presentation in SMIL. A snapshot of the result is shown in Figure 2.6.

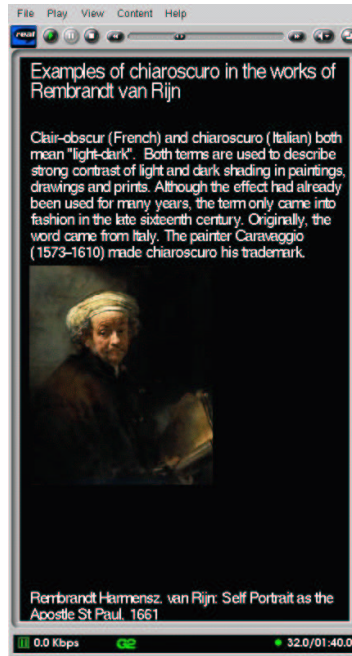


Figure 2.6: Snapshot of the resulting SMIL presentation (RealPlayer 7)

The resulting SMIL markup is listed in Figure 2.8. As one can see, the encoding used for the layout specification in the head is rather low level, and these are indeed the direct values generated by solving the set of quantitative constraints. In contrast, the temporal hierarchy in the body has been generated on the basis of qualitative (Allen) constraints, realizing **during** constraints with `<par>` elements in SMIL, and **after** constraints with `<seq>` elements.

2.2.3 Media in Cuypers

Within multimedia we can differentiate between two types of media, ‘continuous’ such as audio and video, and ‘static’ such as text and images. The difference between these two types is that continuous media have an inherent temporal dimension, typically expressed with a

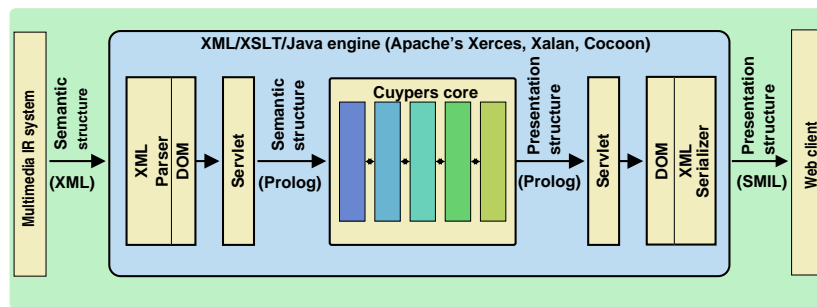


Figure 2.7: The core Cuypers architecture and its integration within the Apache HTTP server.

```

<?xml version="1.0"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 1.0//EN"
      "http://www.w3.org/TR/REC-smil/SMIL10.dtd">
<smil>
  <head>
    <meta name="generator" content="Cuypers 1.0"/>
    <layout>
      <root-layout id="root-layout" background-color="black" width="400" height="690"/>
      <region id="title" left="10" top="5" width="400" height="50" fit="meet"/>
      <region id="descr" left="10" top="55" width="400" height="200" fit="meet"/>
      <region id="img" left="10" top="255" width="400" height="400" fit="meet"/>
      <region id="ptitle" left="10" top="655" width="400" height="35" fit="meet"/>
    </layout>
  </head>
  <body>
    <par>
      <text region="title" src="...query to multimedia database..."/>
      <text region="descr" src="..."/>
      <seq>
        <par dur="10"> ... 1st painting+title ... </par>
        <par dur="10"> ... 2nd painting+title ... </par>
        <par dur="10"> ... 3rd painting+title ... </par>
        <par dur="10"> ... 4th painting+title ... </par>
        <par dur="10">
          
          <text region="ptitle" src="..."/>
        </par>
      </seq>
    </par>
  </body>
</smil>

```

Figure 2.8: SMIL encoding of the presentation shown in Figure 2.6.

duration attribute, while static media have none. Because a multimedia presentation itself is continuous, all media items within the presentation have a continuous property as well. Therefore we have to extend the static media elements with a duration attribute. Since we are automatically generating presentations we have to use a heuristic to define the duration for static media. Constraints allow us to define the duration of a media item as an interval, by specifying a minimum and a maximum duration. The duration of a media item is dependent on its role within the presentation. For example, a background image can be displayed during the whole presentation while, in order to keep the users attention, an image intended to be used as an example should only be shown for a limited amount of time. Moreover, a user needs more time to view a complex image than a simple one, therefore the minimal duration of an image should depend on its size. Textual media require, besides a minimal and maximal duration, also a spatial heuristic. The spatial size of a text is dependent on different factors, such as text length, font type and font size. If, for simplicity, we assume a rectangular text flow we can define a minimal and maximal interval in both spatial X and Y dimensions (which are related) ensuring a fixed surface. Note that the introduction of properties that are not inherent to the media type, introduces a flexibility which might be valuable for media which have inherent properties as well. For example images could benefit from an interval

specification to enable scaling. The original size could be used to define the aspect ratio.

2.2.4 Implementation

To exploit the possibilities offered by on demand multimedia presentation integration, we have integrated the Cuypers core presentation generation engine with an off-the-shelf HTTP server (Apache), as depicted in Figure 2.7 on page 12.

The server parses XML input as shown in Figure 2.4, using the XML parser of Apache's Xerces framework. The result is, via the DOM interface, converted by a Cocoon Java servlet to an equivalent Prolog term. This Prolog term is the actual input taken by the core of the presentation engine, which consists of a number of transformations written in ECLⁱPS^e. ECLⁱPS^e allows the transformations to combine, within a single runtime environment, standard Prolog rule-matching and back-tracking with high-level constraint solving techniques. This allows high level transformation rules to generate alternative layouts using lower-level sets of constraints. Layouts with constraints that prove to be insolvable automatically evaluate to false and cause the system to backtrack, trying alternative layout strategies. In addition, the layout rules can exploit Prolog's unification mechanism as a powerful and extensible selector mechanism, without the need to implement a special purpose selector language such as XPath [11]. When the constraints for a given layout can be solved by ECLⁱPS^e, this solution is returned back to the servlet. The servlet converts the result back to XML (in this case SMIL), again using Cocoon's DOM interface.

The example described above focuses on the use of rhetorical structures as the main technique for describing the input, and on SMIL for describing the final-form output. The core of the Cuypers presentation engine, however, is independent of these formats. Any input that can be transformed to a set of communicative devices can be supported by plugging in a rule set that transforms the input to a set of communicative devices. The same applies to the output format, which can be modified by adapting the lower-level rules that use the (solved) constraints to generate the final form output.

2.2.5 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 [17]. 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) [7], and the relation between SRM-IMMPS with a previous prototype of the system presented here has been described in [26].

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 runtime 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.

2.3 Conclusion

In order to understand the context and contribution of the thesis we explained in this chapter the architecture of Cuypers. We explained the different concepts and presentation abstractions needed to automatically generate a presentation. In chapter 4 we elaborate on the quantitative and qualitative constraints layers and use the notions introduced in this chapter.

In the next chapter we focus on the key notions of constraints and constraint programming.

Chapter 3

Background: Constraint Satisfaction Problems*

Constraints occur often in our daily lives. Whenever we make an appointment we check our agenda to make sure the appointment does not coincide with an earlier made appointment. When we are shopping, we have to make sure we have enough cash with us to pay for the items we would like to have. Constraints are all about restricting possible values. Within constraint programming one tries to solve problems which are defined in terms of restrictions.

In this chapter we will focus on the constraint programming paradigm. We give some theoretical background on constraint solving and domain reduction rules, which constraint programming uses to reduce the search-space. Furthermore, we introduce user-defined constraints, and elaborate on the CHR language which is used for the specification of user-defined constraints. In chapter 4 we employ constraint specifications to transform a semantic structure in terms of communicative devices (see chapter 2) to the final-form presentation.

3.1 Introduction

Within computer science, problems are traditionally solved by dividing the problem into smaller sub-problems which are easier to solve. Solutions to the smaller sub-problems will eventually lead to the solution of the whole problem. Programming languages such as C, Pascal and Java are well suited for writing programs which work in this way. They are called *imperative* languages and try to approach the problem from its solution. That is, the specific features of a solution are known in advance and one tries to reach this state by applying problem-specific transformations on the initial situation.

Declarative programming, in contrast, concentrates on the specification of the problem. Built-in, theoretically based constructs are used then to find a solution. Languages such as Prolog, which uses first order logic as a theoretical basis, and Lisp, which is based upon lambda calculus are typical examples of declarative languages. Declarative programming in particular suits problems where there is no obvious strategy to find a solution.

Constraint programming is also a declarative technique. In order to find a solution, the user states specific requirements on the variables of the problem. The *constraint solver* then tries to reduce the search-space by applying domain reduction rules.

*adopted from Apt [3] and Frühwirth [13]

Problem specifications in terms of constraints are, in particular, useful for complex problems involving a large number of variables with large associated domains. Moreover, for some problems, including combinatorial problems, resource planning and scheduling, constraints provide an intuitive way of problem specification.

3.2 Constraint Programming

A *Constraint Satisfaction Problem*(CSP) is defined as a finite set of relations over some domains. In practice, this means a finite sequence of variables with associated domains and a finite set of constraints, each defined on a sub sequence of the variables. *Constraint programming* is about making CSPs efficiently computable. Within constraint programming there are two phases:

- First, modeling the problem in terms of constraints. This phase is problem and domain dependent. Just as in other modeling tasks, the success of this phase is highly dependent on the skills of the programmer to make sensible abstractions and define efficient data structures.
- Second, applying a procedure which at every step simplifies the CSP. The general architecture of this procedure is equal for every CSP, however, the rules which simplify the CSP are typically dependent of the type of domains in use.
- Finally, the CSP can be in a form which is neither solved, nor failed, but there are no more domain reductions possible. Labeling, that is, assigning a value to a variable from its respective domain, can trigger new domain reductions, and, eventually lead to the success or failure of the CSP. The labeling process is similar to traditional search techniques adopted from computer science.

In this chapter we will mainly focus on the second phase of constraint programming: solving a defined CSP.

The following puzzle can be modeled as a CSP. Every letter represents a unique digit ranging from 0 to 9. The goal is to assign a value to each letter in such a way that they make up a correct addition.

$$\begin{array}{r} SEND \\ + MORE \\ \hline MONEY \end{array}$$

There are different ways to model this problem in terms of constraints. One way is by considering all letters as variables, which have domains [0..9]. We can then setup the equation:

$$\begin{aligned} & 1000 \cdot S + 100 \cdot E + 10 \cdot N + D \\ + & 1000 \cdot M + 100 \cdot O + 10 \cdot R + E \\ = & 10000 \cdot M + 1000 \cdot O + 100 \cdot N + 10 \cdot E + Y \end{aligned}$$

The solution to this equation, CSP and the puzzle is:

$$S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2$$

The paradigm of constraint programming has been implemented as Constraint Solvers in a number of different programming languages. An example is the Cassowary constraint solving

toolkit [5] which is implemented in Smalltalk, C++, Java, and Python. Cassowary works with arithmetic domains and can be incorporated in a user application. Another example is JSolver [9] written in Java, which is able to deal with arbitrary domains. Constraint solvers are also included in Logic Programming system, such as Prolog, resulting in Constraint Logic Programming (CLP) systems. ECLⁱPS^e [32] is an example of such a system. These systems combine the features of Logic Programming (backtracking and unification), with the domain reduction properties of constraint programming. The advantage of CLP in comparison with other constraint solvers is that the specification of constraints is more dynamic. Due to the backtrack mechanism of Logic Programming, alternative constraints can be specified once a previously stated constraint caused a failure. The drawback of CLP, however, is that for a fixed set of constraints the computation time is in general longer than for traditional constraint solvers [8].

3.3 Constraint Propagation

The CSP is processed by a general constraint solving procedure. This procedure transforms the CSP into a simpler, yet equivalent one by applying *domain reduction rules* and/or reducing constraints, which we will explain in more detail later. If there are no more reductions possible, the simplified CSP is processed by a general search procedure².

If a CSP has a solution, we call it *consistent*, if not it is called *inconsistent*. The process of solving this CSP can end in a few possible states:

A solution has been found All variables are associated with a value from their respective domains and they do not conflict with the defined constraints.

All solutions have been found All possible solutions are found.

The best solution is found A solution has been found which maximizes a certain, problem dependent, fitness function which evaluates the quality of a solution.

Inconsistency is detected At least one of the variable domains is empty, which means the variable cannot be associated with a value and therefore the CSP has no solution.

As long as none of these states has been reached the CSP is processed by a constraint propagation procedure. This procedure takes a CSP and returns a simpler, yet equivalent (with respect to the solutions) version of the CSP. Simpler in this sense means the domain of at least one variable or constraint is reduced. When the state is reached that there are no more domain reductions possible (and the CSP is not solved) one can ‘label’ a variable, that is, assign a value to a variable from its respective domain. This can result in additional domain reductions. For example within the CSP

$$(x < y; x \in [3..6], y \in [4..8])$$

²There exist different search techniques adopted from computer science such as depth-first search, breadth-first search and best-first search.

There are optimization’s possible if we combine the domain reduction procedure and the search procedure, since an instantiation of a variable can lead to more possible reductions.

the domain of y can be further reduced if we assign the value 6 to the variable x .

$$(x < y, x = 6; y \in [7..8])$$

The heuristic that determines which variable is assigned which value is dependent on the problem and the domains involved.

Within the constraint propagation procedure there are domain dependent reduction rules and domain independent reduction rules. An example of a domain independent rule is to remove from a domain all values which do not participate in a solution.³

$$\frac{(x = y; x \in [a, b, c], y \in [c, d])}{(x = y; x \in [c], y \in [c])}$$

An example of a domain dependent rule is to remove all values smaller than three from a domain consisting of the integer values 1 up to 10 if the variable associated with this domain participates in a constraint which states that all values should be at least 3 or greater.

$$\frac{(x \geq 3; x \in [0..10])}{(x \geq 3; x \in [3..10])}$$

In addition to reduction in domains, constraints can also be reduced. Reduction of constraints in fact means addition of a constraint, since an added constraint strengthens the CSP and thus limits the possible values. For example:

$$\frac{(x < y, y < z; x \in D_x, y \in D_y, z \in D_z)}{(x < y, y < z, x < z; x \in D_x, y \in D_y, z \in D_z)}$$

The rule introduces a constraint $x < z$ which possibly reduces the domains of both x and z .

3.4 Local Consistency Notions

Both the domain reduction rules and the constraint reduction rules are used to solve a CSP. The constraint propagation procedure is responsible for scheduling which reduction rule to apply. The goal of the procedure is to reach a local consistency. That is, the CSP has reached a state where sub parts of the CSP are consistent and thus have a solution. Because there are no generally applicable, efficient algorithms to solve a CSP, local consistency is an important notion within constraint programming. Determining whether a CSP is consistent in total is then easier and, in some cases, reaching a certain type of local consistency can already be the solution of the CSP.

There are different types of local consistencies, the most common are:

Hyper-arc consistency A CSP is called hyper-arc consistent if all domains only contain values which can participate in a solution. For example the CSP:

$$(x = y; x \in [1, 2], y \in [1, 2])$$

is hyper-arc consistent since the values of both domains can participate in a solution. In contrast, the CSP:

$$(x = y; x \in [1, 2], y \in [1, 2, 3])$$

³The format of a reduction rule is: $\frac{\varphi}{\psi}$, where φ and ψ are CSPs. ψ is the result after a domain in φ has been reduced.

is not hyper-arc consistent since the value 3 does not participate in any solution.

A weaker notion of hyper-arc consistency is *directional arc consistency*. This means the consistency considers a specified direction. The second example is thus directional arc consistent if we consider the direction from y to x . The domain of x consists of the values 1 and 2 which both can participate in a solution.

However, from the direction x to y the CSP is not directional arc consistent. Within the domain of y there is a value 3, which does not participate in any solution.

Path consistency A normalized⁴ CSP is called path consistent if for each subset $\{x, y, z\}$ of its variables the following holds:

If $(a, c) \in C_{x,z}$ where C_X denotes⁵ the unique constraint C on the subsequence of variables X . (Note the arity of X is here 2 because of a binary constraint.) And, (x, y) denotes an instantiation which does not violate the constraint C_X , then there exists a b such that $(a, b) \in C_{x,y}$ and $(b, c) \in C_{y,z}$.

Informally this means that a transitive relation cannot lead to an inconsistency. For example the CSP:

$$(x < y, y < z, x < z); x \in [0..4], y \in [1..5], z \in [6..10]$$

is path-consistent since for every possible values for x and z , which satisfies the constraint $x < z$, we can choose a value for y which satisfies both $x < y$ and $y < z$.

In contrast, the CSP:

$$(x < y, y < z, x < z); x \in [0..4], y \in [1..5], z \in [5..10]$$

is not path consistent since the assignment $x = 4$ and $z = 5$, which satisfies the constraint $x < z$, leaves no possible value for y to satisfy $x < y$ and $y < z$.

A weaker notion of path consistency is directional path consistency which differs from path consistency in a similar way that directional arc consistency differs from hyper-arc consistency.

K -Consistency We call a CSP 1-consistent if every unary constraint on a variable x is satisfied by all values in the domain of x .

We call a CSP k -consistent, where $k > 1$, if for every $(k - 1)$ -consistent instantiation,⁶ there is, for every uninstantiated variable x , a value in its associated domain which is legal and, thus makes the CSP k -consistent.

The following CSP is 3-consistent:

$$(x \neq y, x \neq z, y \neq z; x \in [1, 2], y \in [2, 3], z \in [1, 3])$$

For every possible instantiation of two variables, there exist a value in the domain of the third variable which is consistent. The CSP:

$$(x \neq y, x \neq z, y \neq z; x \in [1, 2, 3], y \in [1, 2, 3], z \in [1, 2])$$

⁴A CSP is normalized if for each sub sequence of its variables there is at most one constraint defined. For every CSP there is a normalized equivalent (see [3] pag. 52).

⁵For example the CSP $(x < y; x \in [0..2], y \in [0..2])$ C_{XY} denotes $\{(0, 1), (0, 2), (1, 2)\}$.

⁶Assigning a value to a variable from its associated domain

is not 3-consistent. If we consider the 2-consistent instantiation $x = 1, y = 2$ there is no value for z which satisfies the constraints.

K -consistency has a stronger notion called strong k -consistency. We call a CSP strong k -consistent if for every value $i \in [1..k]$ it is i -consistent.

The following CSP is strong 3-consistent, since it is 1-consistent, 2-consistent and 3-consistent

$$(x \neq y, x \neq z, y \neq z; x \in [1, 2, 3], y \in [1, 2, 3], z \in [1, 2, 3])$$

In contrast, the following CSP is not *strong* 3-consistent despite the fact that it is 3-consistent. Consider the assignment $y = 1$, there is no value in the domain of z which is 2-consistent, therefore the CSP is not 2-consistent and not strong 3-consistent.

$$(x \neq y, x \neq z, y \neq z; x \in [1, 2, 3], y \in [1, 2, 3], z \in [1])$$

CSPs often work with problems which can be represented by numerical domains. However, CSPs are not limited to numerical domains, and in principle any type of domain can be used. Constraint programming is successfully applied in problems where the types of domains are Boolean, real and symbolic. In order to make use of other, non built-in types of domains the domain solving rules should be specified. For example if we consider a Boolean domain and the conjunction $x \wedge y = z$. The table consisting of the possible domains for x, y and z contains 54 entries.⁷ However the following rules encode the same information:

$$\begin{aligned} x \wedge y = z, x = 1, y = 1 &\rightarrow z = 1 \\ x \wedge y = z, x = 1, z = 0 &\rightarrow y = 0 \\ x \wedge y = z, y = 1, z = 0 &\rightarrow x = 0 \\ x \wedge y = z, x = 0 &\rightarrow z = 0 \\ x \wedge y = z, y = 0 &\rightarrow z = 0 \\ x \wedge y = z, z = 1 &\rightarrow x = 1, y = 1 \end{aligned}$$

If, subsequently solving rules for equality(=), disjunction(\vee) and negation(\neg) are added the Boolean system is complete. The system is arc consistent, that is, the domains of a CSP in this system contain only values which can participate in a solution.

The complete Boolean system contains 20 solving rules. This number grows considerably if the size of the domains increases. For example, if we consider the transitive relation of the 13 temporal relations defined by Allen [1] the number of solving rules which lead to an arc consistent CSP contains over 26000 rules. This is, obviously, too large to generate by hand. Apt and Monfroy[4] describe an algorithm which automatically generates these rules for small finite domains. In the case of the Allen relations, the table provided by Allen ([1] page 836) is used as input for the algorithm. The algorithm takes about 5 days on a 500 MHz Pentium III PC to generate 26000 rules. In practice, 26000 rules require too much overhead in terms of memory and processor resources to be of practical use. Apt and Monfroy introduce a new consistency notion: *Rule Consistency* which is weaker than arc consistency but takes fewer resources to compute. The algorithm generates 500 rules which lead to rule consistency in less than 6 seconds. We use these generated rules as part of the reasoning rules in the Cuyper system, described in section 4.3.1.

⁷3 variables(x, y, z) with 3 possible values ($\{\{0\}, \{1\}, \{0, 1\}\}$) make $3^3 = 27$ combinations. Each of these combination can be either true or false.

3.5 Constraint Handling Rules

ECLⁱPS^e has a number of commonly used, predefined solvers for numerical domains, interval domains and Boolean domains. These built-in solvers, however, are not in all cases sufficient to suit the specific needs of the user. For non-typical domains such as temporal logic, three-value logic etc. the domain reduction rules have to be explicitly defined by the user. Constraint Handling Rules(CHR) [13] is a language developed to specify these rules.

In a nutshell, a CHR program consists of constraint definitions and constraint handling rules. The constraint definition makes the system aware of the *new* user-defined constraint by specifying its name, and the number of arguments the constraint takes (e.g. `constraints minimum/3.`). Constraint handling rules are used to transform or extend the specified constraints (constraint store). That is, by applying a *simplification rule* a constraint is replaced by a simpler, yet equivalent one (e.g. $X > Y, Y > X \rightarrow fail$). A *propagation rule* in contrast extends the constraint store with logical redundant constraints which can cause further simplification (e.g. $X > Y, Y > Z \rightarrow X > Z$). Repeatedly applying these rules simplifies the CSP and eventually solves it.

In more detail, a constraint handling rule consists of the following components:

Name A rule can have a name. Although this is optional and does not influence the behavior of the program it is advised to name the rules within a CHR program for debugging reasons. Knowledge about which rule fired is often valuable information for tracking errors.

Head Every CHR rule has a head. A head contains one or more constraints⁸ and they determine whether the rule is applicable. That is, if the head of the rule matches constraints from the constraint store then the rule is applicable.

Type The type of a rule defines whether the rule is a *simplification rule* or a *propagation rule*⁹

Guard Optionally a rule has a guard. A guard is an additional check as to whether the rule should be applied after the head of the rule is matched. If the guard succeeds the body of the rule is executed. If the guard fails the rule is delayed.

Body If the head is matched and the guard is passed the body of a rule is executed. In ECLⁱPS^e the body can contain constraint specifications and Prolog goals. Dependent on the type of rule the body extends the constraint store (propagation) or replaces the matched head of the constraint store (simplification).

Within a CHR program, the application of a rule modifies the constraint store. Because of this, other rules might become applicable as well. The general rule is that, whenever a variable in the head of a rule is altered, that is, its domain is reduced, the rule is adopted for application again. If there are multiple rules applicable the first specified rule by the programmer is executed first, then the second, etc.

footnoteIn older versions of ECLⁱPS^e the CHR compiler decided which rule should be applied. When there are no more rules applicable the CSP is either solved, in which case all domains of all variables contain one value, or there are no reductions possible but the domains have not reduced to one. In this case, labeling the variables can lead to a solution.

⁸In ECLⁱPS^e a maximum of 2 constraints in the head is fully supported.

⁹Strictly speaking there is a third type, called the *simpagation rule* which is a combination of the first two.

3.5.1 Example

Consider media items which can be positioned left or right from each other. The inverse relation can be specified using a simplification rule (\Leftrightarrow) named `inv1`, defined as:

```
% Name      Head          Type  Guard          Body
   inv1     @ A rgt B      <=>  visual([A,B]) | B lft A.
```

Note the use of the guard that prevents this rule from being tried for non-visual items (such as audio fragments). The transitive nature of the *left of* relation `lft` could be specified using a propagation (\Rightarrow) rule named `trans1`:

```
% 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 }
```

In our example the constraint “img1 rgt img2” matches the head of the rule `inv1`. 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. Before this replacement can proceed, however, the guard has to succeed. In our case the guard states that A or B should be visual items. Since the example deals only with images, the test will succeed and pass to the body. The body defines a new constraint and replaces “A rgt B” for “B lft A”. So the constraint store was

```
{ img1 rgt img2, img1 lft img3 }
```

and after the rule `inv1` is applied the constraint store is:

```
{ 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 constraint store is now extended with the new constructed constraint from the body. After the rule `trans1` is applied the constraint store is updated:

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

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.

3.6 Conclusion

To provide context to the application of constraint programming to automatic multimedia presentation generation we introduced, in this chapter, the most important concepts of constraints and constraint programming.

We explained that constraints are declarative and particularly suited to problems which are complex in the sense that the variables are numerous and the associated domains are large. Furthermore, we showed that constraints can be applied to both numerical and to non-numerical domains and that the CHR language can be specified to define the domain reduction rules.

In the next chapter we elaborate on the application of the constraint programming paradigm in the context of automatic multimedia presentation generation.

Chapter 4

Constraints for Multimedia

We define a multimedia presentation as a collection of media items which are logically ordered to convey a certain message. In chapter 2 we showed that the presentation of a media item is influenced by three factors: semantics, design and resources. These factors are not orthogonal, and during presentation generation a trade-off has to be made between these factors. Because the validity of a choice is, in some cases, not known until the end of the process, the generation is not linear. Instead, the process consists for a considerable part of revising earlier made choices.

In this chapter we apply the paradigm of constraints and constraint logic programming as discussed in chapter 3 to multimedia presentation generation. We will show how constraints provide a natural way of specifying the previously mentioned problems. We argue that constraint logic programming has several advantages compared with other techniques which make it better suited to address these issues efficiently. Furthermore, the typical domains in multimedia are both quantitative and qualitative. We will show how qualitative domains in combination with quantitative domains can be successfully applied within multimedia generation. Finally, we show how the labeling of uninstantiated variables becomes an important issue when both quantitative and qualitative domains are used.

4.1 Introduction – example scenario

In this chapter we will focus on the constraint layers within the Cuypers system architecture, as introduced in chapter 2. To illustrate the specific problems and proposed solutions within these layers we make use of a running example throughout this chapter. The example is based on section 2.2.2 where a user is interested in the work of Rembrandt van Rijn. For simplicity, we assume the system retrieved four images of examples of the work of Rembrandt which have to be presented to the user (in any order). Furthermore the author of the presentation has stated the presentation should not exceed 20 seconds. Finally, the user profile specified a maximum screen size.

In our example we assume minimal and maximal duration of discrete media items (see chapter 2) which leads to the following input:

A type:image, width:130, height:25, minimal duration:10, maximal duration:20

B type:image, width:150, height:75, minimal duration:10, maximal duration:20

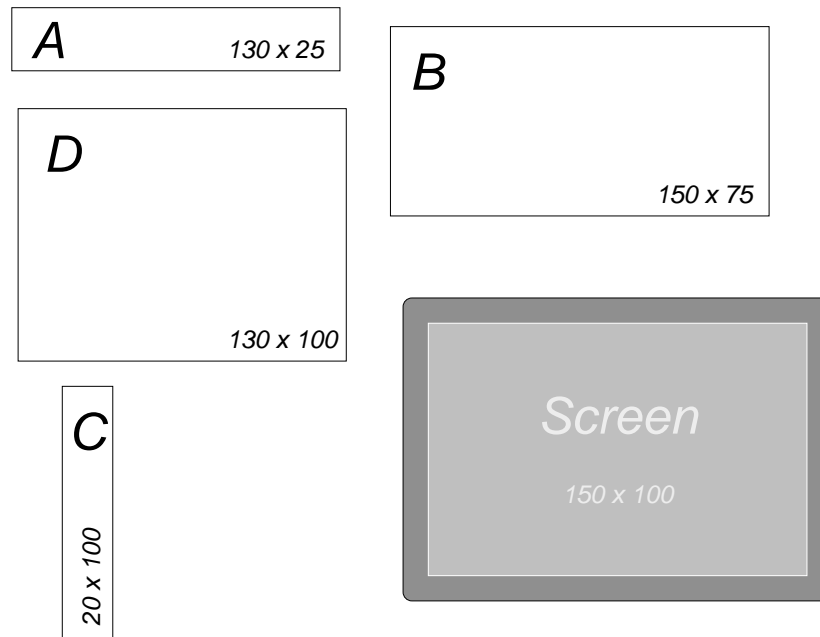


Figure 4.1: Goal of the system is to display all 4 media items within 20 seconds.

C type:image, width:20, height:100, minimal duration:10, maximal duration:20

D type:image, width:130, height:100, minimal duration:10, maximal duration:20

Screen width:150, height:100

Duration < 20 sec.

The task of the system is to find a configuration which satisfies these constraints (see figure 4.1). As shown in chapter 2, the placement of spatial media in multimedia presentations is less flexible than in static/textual documents such as HTML. For example, in SMIL, spatial media are placed within a box whose sizes are specified by the author. This box can either be absolute, in which case the position is determined by coordinates, or relative in which case the size of the box is determined in percentages, relative to the total screen size. In both ways the position of a media item is fixed in relation to the other media. Therefore determining the position of a spatial media item is equivalent to finding its coordinates (which can be mapped to relative percentages if needed). The temporal dimension within SMIL documents is more flexible in the sense that you can specify whether two items should be played in parallel or sequentially. However, to detect, for example, whether a presentation is conflicting with its maximum duration the system should be aware of the total duration of the presentation. Therefore the temporal coordinates of a media item are needed as well.

4.2 Quantitative constraints

Within Cuypers, we model media items as three-dimensional rectangular boxes¹ which cannot be rotated. Because of these simplifications we can consider the spatial and temporal dimension orthogonal, which means there are no dependencies between the X, Y, T -dimension at the level of constraint specifications. Moreover, it allows us to model every media item by at most 6 coordinates. Although this simplification may seem a limitation, in practice most multimedia output formats (including SMIL) also support only non-rotated rectangular media items.

So far, we have distinguished three dimensions within a multimedia presentation.

Temporal Multimedia presentations are structured around temporal synchronization. Therefore all media use the T dimension. The temporal dimension corresponds to the duration of the presentation and is typically measured in seconds or milliseconds. Temporal media, just as spatial media, are represented by an interval. Two dependent variables T_1 and T_2 are needed to model the interval. The domain of these variables consists of the values on the T dimension. The length of the interval represents the duration of the media element.

Spatial Visual media such as images, text and video use both spatial dimensions X and Y . The X and Y dimensions correspond to the window of the user's application where the sizes are typically measured in number of pixels or other discrete units. Therefore the values on the X and Y dimension can be modeled by positive integers. The X interval is modeled by two dependent variables X_1 and X_2 where the domains consist of the values of the X dimension. The length of the interval represents the width of the media item. The Y interval is modeled in a similar way by using the variables Y_1 and Y_2 . The length of the interval represents the height of the media item.

Stacking order Visual media have a Z dimension which describes the stacking order of items. This can be represented by an integer value. All media items which have a lower Z value are ordered on top of media items which have a higher value. When two items have the same Z value the stacking order is undefined.

Hyperlinks Within hypermedia documents, media items can be connected by hyperlinks which introduces a link dimension. However, in this thesis we focus on multimedia and hyperlinks are thus beyond our scope.

We define the set of variables associated with their domains as a multimedia Constraint Satisfaction Problem (CSP). For simplicity, we consider in the example only the temporal and spatial dimensions. Within the example we use four media items, A, B, C, and D (see figure 4.1). Every bounding box corresponds to a media item from the example. Each box is defined by 6 variables, of which X_1, X_2, Y_1 and Y_2 define the spatial dimensions, while T_1 and T_2 define the temporal dimension. All variables are defined as positive integers.

¹Rectangular boxes have some drawbacks in the sense that all media items (except audio, which has no spatial dimension) are generalized to rectangular boxes. Circular or polygon shaped media items are thus represented by their bounding boxes which are defined by the minimal and maximal coordinate in all three dimensions, covering the complete shape. In addition to the generalization of media items to rectangular shaped boxes, we do not allow rotation of these boxes.

After initialization the four boxes are still completely unconstrained, which means that every possible assignment of a domain value to a corresponding variable is a valid one. For each box, we can constrain the variables by stating that the variable with index 1 is always smaller than the variable with index 2. Moreover, we know that all boxes have a width, height and duration so we can set the difference of the two variables within a dimension with index 1 and 2 to respectively the height, width or duration of the image. We already mentioned the non-inherent duration attribute for images in chapter two, within our example we assume a duration interval which is specified by minimal and maximal duration attributes. For simplicity we do not allow a variable width and height as suggested in chapter two, we assume a single height and width attribute. Finally, the maximum screen size and the maximum duration is known in advance. This constrains the upper bound of all 6 variables. The following excerpt of ECLⁱPS^e code initializes a given box identified by its Id. For each dimension two variables are created and associated with a positive integer domain.^{2 3}

```
% csp(+Media, -Boxes)
% generate a CSP
csp(Media, Boxes) :-
    initBoxes(Media, Boxes).

% initialize all media
initBoxes(Media, Boxes) :-
    ...

% initBox(+Id,-Box)
% initializes a bounding box for visual media item Id
initBox(Id,Box) :-
    Box = box(Id,[x/1:X1,x/2:X2,y/1:Y1,y/2:Y2,t/1:T1,t/2:T2]),

    % X dimension
    screenX(MaxX), % get screen boundary value X
    [X1,X2] :: [0..MaxX], % set domain
    attribute(width, Id, Width), % get Width of media Id
    X2 #= X1 + Width, % set Width of media Id

    % Y dimension
    screenY(MaxY),
    [Y1,Y2] :: [0..MaxY],
    attribute(height, Id, Height),
    Y2 #= Y1 + Height,

    % T dimension
    duration(MaxT),
    [T1,T2] :: [0..MaxT],
    attribute(min_duration, Id, MinDur), % get minimal duration
    attribute(max_duration, Id, MaxDur), % get maximal duration
    T2 #>= T1 + MinDur, % set minimal duration
    T2 #<= T1 + MaxDur. % set maximal duration
```

The dimensions in a multimedia CSP are orthogonal. Therefore, no variable participates in

²‘+’ means input variable, ‘-’ means output variable, ‘?’ can be used both as input or output variable.

³ As mentioned before, audio fragments do not have a spatial dimension. Therefore, boxes representing audio fragments contain only T_1 and T_2 variables. (This is not shown in the code example). The initialization of an audio fragment is different in the sense that initialization for the spatial dimension is skipped. Moreover, in the current implementation the duration of an audio fragment is fixed.

multiple dimensions. We can consequently consider the CSP as three independent sub-CSPs. The whole CSP is solved once all sub CSPs are solved.

Within a sub-CSP, two variables can be either dependent or independent of each other. Dependency in this sense means that once the domain of a variable is modified, all dependent variables are candidates for a revision of their domains as well. Dependencies between variables are created by letting them participate in a constraint. During the initialization of a CSP dependencies are created between the variables which define the width, height and duration of a media item.

All variables are associated with a domain. In order to generate a presentation the variables of the CSP need to be assigned a value from their domain. If there are multiple values available, a choice has to be made. Once a value is assigned to a variable the domains of dependent variables are candidates for domain reduction. If a conflict rises between the newly instantiated variable and these dependent variables, the choice for a value is revised. This labeling process is repeated until there are no more conflicts.⁴

The initialized CSP (as constructed by the ECLⁱPS^e code above) generates all possible placements of media items within the stated physical boundaries, screen size and maximum duration. Therefore this CSP represents the complete search space of a multimedia presentation which consists of the four example images.

The search space, however, is still very large and contains, within the scope of multimedia presentations, unwanted solutions. Therefore we have to constrain the possible solutions further. Extra constraints are generated by *communicative devices*, as described in chapter 2, which implement a semantic construct by adding constraints on the respective media items. For example, the communicative device “follows”, which is used to make sure the user is first made aware of A before he/she is made aware of B, can be modeled by the temporal constraint that item A should immediately be followed by item B. In terms of constraints: the T_2 variable of item A should be equal to the T_1 variable of item B. We represent this as a graph where the nodes represent the variable coordinates of the box and the edges represent the relation in terms of arithmetic (in)equality constraints (see figure 4.2).

In ECLⁱPS^e code:

```
% follows(+IdA,+IdB,+Dim)
% adds a constraint that A is followed by B
follows(IdA,IdB,Dim) :-
    node(IdA,Dim/2,VA2),    % 'get' variable IdA with index 2 (eg. T2)
    node(IdB,Dim/1,VB1),   % 'get' variable IdB with index 1 (eg. T1)
    VA2 #= VB1.            % set constraint T2 #= T1
```

4.2.1 Quantitative constraints and CLP

In the example we retrieved four images. The goal of the system is to show the images to the user within a timespan of 20 seconds. We assume the images should be completely visible and are therefore are not allowed to overlap both spatially and temporally.

The ‘non-overlap’ constraint cannot be expressed by a single arithmetic constraint, as shown by Kautz [18].⁵ In order to model non-overlap constraints we need to generate two al-

⁴The default heuristic to label a variable with a domain value is to pick the smallest value from the domain. This can however be modified and we will discuss alternative heuristics later.

⁵Arithmetic constraints lack an *or*-operator which is required to specify non-overlap between media items.

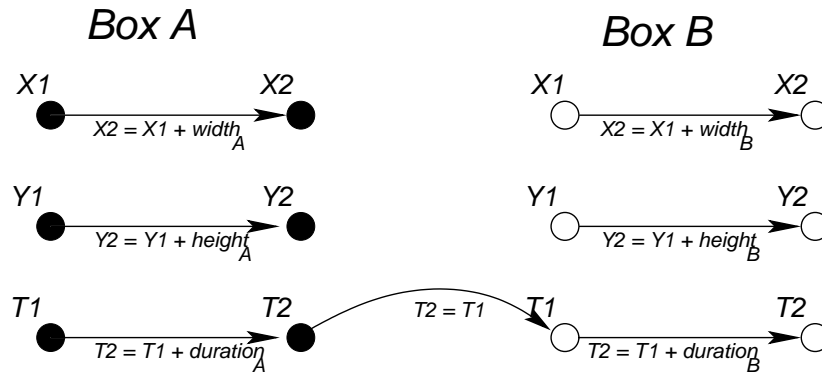


Figure 4.2: CSP between two media items A and B

ternative CSP's where the non-overlap (in one dimension) constraint is modeled by expressing both non-overlapping possible placements.

There are two ways of placing one image next to the other in the X dimension. This also holds for the Y and T dimension, which results in $2! \times 2! \times 2! = 8$ distinct ways of combining 2 items. In our example we use 4 images which will result in $4! \times 4! \times 4! = 13824$ distinct CSPs. Note that this is a simplified case, we considered only two possible relations for the X , Y and T dimensions while in practice there are more⁶. Furthermore, we only considered three dimensions, ignoring the Z dimension.

Before finding out that a set of constraints has no solution, all generated CSPs have to be tried. Because of the combinatorial explosion, this is inefficient and therefore not a realistic option. For this reason, constraint solvers embedded in imperative programs cannot be used.

Constraint Logic Programming addresses this problem because of its ability to backtrack over insolvable constraint definitions. This way, alternative constraints can be specified immediately on failure.

We define the spatial and temporal relation between media items A and B as 'A left of B' or 'B left of A', 'A above B' or 'B above A' and 'A before B' or 'B before A'. These relationships are defined in terms of arithmetic constraints. If the first one fails, the second one is tried. If this one fails as well, the the third is tried etc. This behavior is more efficient since a failure in constraints is addressed immediately, while the former approach could only discover a failure after specification of all constraints. The next excerpt of code illustrates the initialization of media items with the constraint that two media items should not overlap. The Prolog backtrack-engine makes sure that, once the first rule fails, the second one is tried.⁷

```
% generate a CSP
csp(Media, Boxes) :-
    initBoxes(Media, Boxes),
    relateN(x,Boxes),
    relateN(y,Boxes),
    relateN(t,Boxes).

% relate all media items
relateN(Dim, [B1|[B2|Boxes]]) :-
```

⁶In Cuypers we use 13 different relations.

⁷Note that Prolog in fact extends the system with the, previously mentioned, missing 'or' operator.

```

...
relate(Dim,B1,B2),
...

relate(Dim,Box1,Box2) :-
    % get coordinates
    attribute(Dim/1, Box1, B1_D1),
    attribute(Dim/2, Box1, B1_D2),
    attribute(Dim/1, Box2, B2_D1),
    attribute(Dim/2, Box2, B2_D2),

    % box1 left of/above/before box2
    (      B1_D2 #< B2_D1

    % or, if this is not possible try
    ;

    % box2 left of/above/before box1
      B2_D2 #< B1_D1
    ).

```

Since we also consider a temporal dimension in our example a relation between two media items consists of 8 different options. Each option is expressed by means of inequality constraints as shown above. Inequality constraints cause the domains of associated variables to shrink. However, after all have been applied, the domains of the variables can still have multiple values. Reduction to intervals is insufficient since the placement of media items in a multimedia presentation requires fixed coordinates. Therefore labeling of the variables with domains larger than one is needed.

4.2.2 Drawbacks in quantitative constraint processing

Although the combination of CLP and quantitative constraints improves the efficiency in multimedia presentation generation, there are also drawbacks:

Inefficiency Consider the example with the following configuration: ‘A left of C’, ‘D after A’ and ‘D after C’ (see figure 4.3). These positions are all legal within the specified constraints, still there is no valid placement for B to satisfy all constraints and therefore this configuration will not succeed. Nevertheless, the domains of spatial variables X_1^D, X_2^D will contain multiple values (to be more precise the domains will be $X_1^D \in [0..20], X_2^D \in [130..150]$.) The labeling procedure picks one of these values for variable X_1^D and finds out image B does not fit, so it tries another value which fails for the same reason. In fact all 20 values are tried before it concludes that this does not lead to a solution. This holds for all variables with domain sizes larger than 1 and all alternatives will fail for the same reason. In our example the X_1^D, X_2^D variables define the width of image D and are therefore dependent on each other. This results in 40 variable - value assignments. Variables $Y_1^A \in [0..75]$ and $X_1^D \in [0..20]$ are independent. This will result in 1500 different labelings. In practice this turns out to be too inefficient to be of practical use. Moreover, when we are looking for alternative solutions, the system will generate solutions which only differ one unit (pixel, millisecond) but are essentially the same from a user perspective.

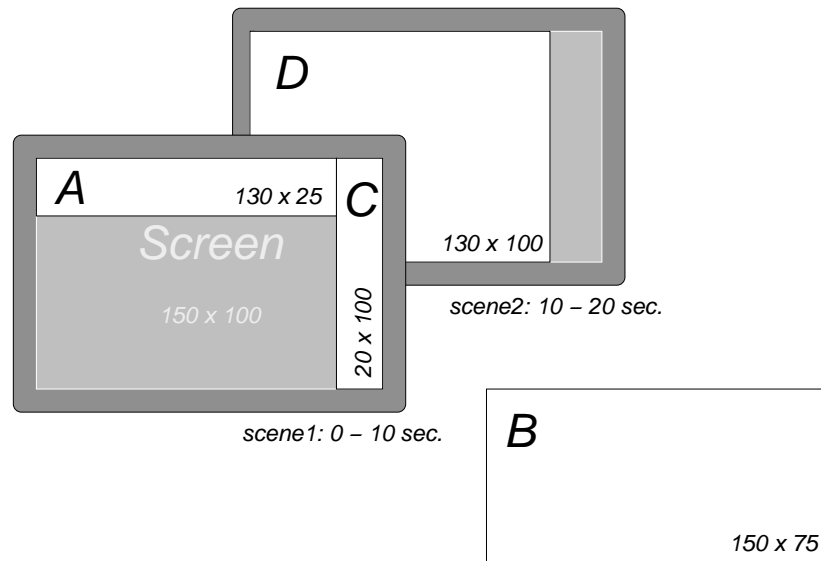


Figure 4.3: Wrong configuration

Insufficiently expressive Besides inefficiency, the use of only quantitative constraints lacks some specific features which are useful in multimedia presentation generation. Inequality relations allow 5 different relations between variables ($<$, $>$, $=$, \leq and \geq). These are rather limited and force a relative choice for the position of an item with respect to another item. This yields, for example, that within the constraint language it is not possible to state that an item should not overlap with another item. In order to get the desired result you have to rely on the Prolog backtrack engine by stating alternative rules for media placement. This results in loss of performance because of the less efficient behavior of the backtrack engine in comparison with the constraint engine.

Loss of higher level control Multimedia presentation languages such as SMIL allow high level synchronization specification of the media items, such as ‘item A should be directly followed by item B’. This type of specification allows the system to deal with, for example, network delays that can occur during the playing of the presentation. The system makes sure two media items which are intended to play in parallel are really played in parallel. Because quantitative constraints work on integer numbers this type of information is lost and media items are assigned numerical values when they should start and end. This way the system cannot take into account network delays.

Non-intuitive Finally, the constraint engine should be usable by third party designers who write their formatting rules in terms of constraints (see chapter 2 for more details). The interface to the constraint specifications should, for this reason be, as user friendly as possible. The use of quantitative constraints in this sense is not very intuitive. Instead, a higher level specification would be more appropriate.

These drawbacks show that the use of only quantitative constraints is not sufficiently efficient, not intuitive and too low level. In the next section we introduce qualitative constraints which can specifically be tailored for multimedia presentation generation and overcome the

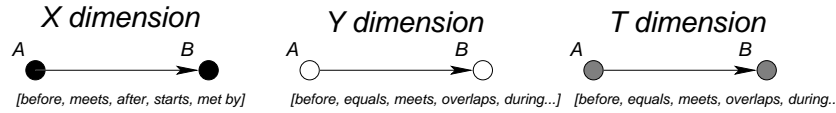


Figure 4.4: qualitative graph

problems of quantitative constraints.

4.3 Qualitative constraints

The previous section showed that quantitative constraints on their own are insufficient for multimedia presentation generation. In this section we show that constraint specification in qualitative terms provides solutions to these problems. Qualitative constraints have smaller domains, are more intuitive and more expressive. We argue that Allen's 13 temporal relations can be used to model both the temporal and spatial dimensions of a multimedia presentation in qualitative terms. Because we define our own type of domains, we have to specify rules which enable the constraint solver to make efficient domain reductions. Although qualitative constraints have advantages in comparison to quantitative constraints, we still need quantitative domains to cope with numerical properties, such as screen size and time limits. Therefore, we need rules which translate qualitative constraints into quantitative constraints.

In the example we have 4 images of different sizes, the goal is to display them within a time span of 20 seconds. We identified 4 different dimensions within a multimedia presentation. The X, Y dimension set up the spatial layout and the T dimension defines the temporal layout. Finally the Z dimension specifies the stacking order of the spatial media items. So far we modeled all these dimensions in terms of quantitative constraints on integers. This setup, however, caused complexity problems and therefore we have to use other domains. Allen [1] showed that within the temporal dimension there are 13 distinct relations possible between two media items. Although these relations are primary intended to be used as temporal relationships, they can also be used to model the spatial X, Y dimensions.

Within the Z dimension it is not necessary to model the possible relations in qualitative terms since they are already covered by the arithmetic inequality and equality relations. For example $A > B$ models A is stacked on top of B and, A and B are on the same stacking layer is coded by $A = B$.

Modeling these relationships as qualitative constraints does not provide additional functionality.

Within qualitative domains we make the same assumptions as we did with the quantitative domains. We represent the media items as rectangular boxes which cannot be rotated. Just as in the quantitative situation, the dimensions are orthogonal (see figure 4.4). Different however, in comparison to the quantitative situation, is the way we represent the relations in graph terms. In the quantitative situation we needed to find the coordinates of the media items and the (arithmetic) relations were assumed to be known in advance.

In the qualitative situation, however, we know that every pair of media items is related but we do not know what the relation is. In terms of a graph representation, this means we have a complete graph where the nodes in the graph represent media items and the edges represent variable relations with other media items.

The complete search space is defined by three compactly connected orthogonal graphs. There are 13 distinct relations two media items can have. The size of the complete search space is thus $3 \times \left(\frac{n^2-n}{2}\right)^{13}$ where n is the number of media items. Within our example we use 4 media items, the size of the complete qualitative search space is considerably smaller than the quantitative search space.

The following excerpt of code initializes the complete qualitative search space.

```
% init(+MediaIds, -Graph)
% setup a complete qualitative graph
init(MediaIds, [XGraph, YGraph, TGraph]) :-
    fullConnect(MediaIds, x, XGraph),
    fullConnect(MediaIds, y, YGraph),
    fullConnect(MediaIds, t, TGraph).

% fullConnect(+Ids, +Dim, -Graph)
% create a fully connected graph of Ids
fullConnect([IdA, IdB|Ids], Dim, [Edge|Edges]) :-
    edge(IdA, IdB, Dim, Edge),
    ...
    true.

% make_edge(+IdA, +IdB, +Dim, -Edge)
make_edge(IdA, IdB, Dim, edge(IdA, IdB, Dim, Value)) :-
    % define domain
    Value :: [b, d, o, m, s, f, e, 'b-', 'd-', 'o-', 'm-', 's-', 'f-'],
    edge(IdA, IdB, Dim, Value).
```

Within the example, the constraints need to express in qualitative terms that two media items should not overlap spatially if they overlap temporally. The temporal relations of Allen define ‘before’ and ‘meets’ as non-overlapping and ‘starts’, ‘during’, ‘overlaps’, ‘finishes’ and ‘equals’ as overlapping. The specification of a non-overlapping relation between two media items in a certain dimension yields that the overlapping relations are removed from the domain. To express that two media items should not overlap spatially we need to state that there should be no overlap on the x dimension or on the y dimension. If there is a temporal dimension aswell there should be no overlap in the x, y dimension if there is a temporal overlap. And vice-versa, if there is a spatial overlap there should not be a temporal overlap. The following code⁸ sets up thses non overlapping constraints between two media items. Note that the Prolog ‘or’ used here is different from the code on page 18 since it is used here to provide alternatives between dimensions, whereas in the previous case it was used for alternatives within a single dimension.

```
% No Overlap in T dimension
noOverlap(A, B) :-
    % define domains
    NoOverlap :: [m, 'm-', b, 'b-'],
    [AllX, AllY] :: [m, 'm-', b, 'b-', e, d, 'd-', s, 's-', f, 'f-', o, 'o-'],
    edge(A, B, x, AllX),
    edge(A, B, y, AllY),
    edge(A, B, t, NoOverlap).

% OR:
```

⁸Predicates with the same name can be read as *or* within prolog.

```

% No Overlap in Y dimension
noOverlap(A,B) :-
    NoOverlap::[m,'m-',b,'b-'],
    [AllX,AllT]::[m,'m-',b,'b-',e,d,'d-',s,'s-',f,'f-',o,'o-'],
    edge(A,B,x,AllX),
    edge(A,B,y,NoOverlap),
    edge(A,B,t,AllT).

% OR:
% No Overlap in X dimension
noOverlap(A,B) :-
    NoOverlap::[m,'m-',b,'b-'],
    [AllY,AllT]::[m,'m-',b,'b-',e,d,'d-',s,'s-',f,'f-',o,'o-'],
    edge(A,B,x,NoOverlap),
    edge(A,B,y,AllY),
    edge(A,B,t,AllT).

```

4.3.1 Reasoning with Qualitative constraints

Although the problem is set up in terms of constraints, the system is not capable of making domain reductions to the associated variables, because it does not ‘understand’ the nature of the Allen relations. In order to do so we provide the system with rules that define the specific features of the Allen relations and enable the system to reason with them and make domain reductions where possible. These rules(see appendix A.1) are specified in the CHR language which we described in chapter 3. We defined three types of rules which describe the basic properties of the 13 Allen relations:

inverse Within the Allen relations, 6 values are inverse values of other values. From a user’s perspective these 13 relationships are necessary for ease of use. From a system’s perspective, however, we can simplify the CSP by translating all inverse values to their normalized values. An edge from A to B with value V can be replaced by an edge from B to A with the inverse value of V. Every value (except equals) within the Alan domain has an inverse relation. To enable simplifications and prevent looping, however, we only apply the inverse rule in one direction.

transitivity An edge from A to B and an edge from B to C within the same dimension and $A \neq B \neq C$ can influence the possible values for the edge between A and C. For example if A ‘overlaps’ B and B is ‘during’ C then we know that A is either, ‘overlapping’, ‘during’ or ‘starts’ B. These transitive relations have been explored by Allen and provided in a table (see [1], page 836). The RULES GENERATION ALGORITHM [4] transforms this table into equality constraints which the system is able to process. (see chapter 3)

equality Two edges within a certain dimension that have equal start nodes and equal end nodes are the same edges and should therefore have the same values. Because the application of the inverse rule is only applied in one direction there is an additional rule needed which states that if there is an edge from A to B with value V and there is an edge from B to A then the value of this edge should be the reverse value of V.

These rules enable the system to detect inconsistencies in the specifications of qualitative constraints. They do not, however, account for numerical limitations such as exceeding the maximum screen size, duration or padding distances. These are quantitative properties which

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	$X_1^A < X_1^B, X_2^A > X_1^B, X_2^A < X_2^B$
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	$X_1^A = X_1^B, X_2^A = X_2^B$

Table 4.1: translating qualitative constraints to quantitative constraints

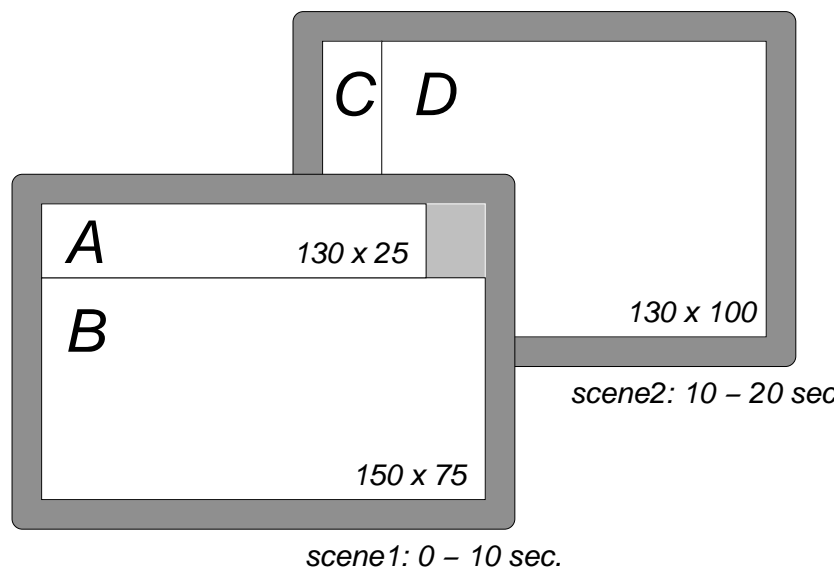


Figure 4.5: good configuration

cannot be expressed by qualitative constraints. Therefore we need a combination of both qualitative and quantitative constraints: Qualitative constraints to allow high level reasoning, and quantitative constraints to check the numerical limitations. Once we know the qualitative relation between two media items we are able to translate this qualitative constraint in terms of quantitative constraints (see table 4.1). If the quantitative constraints cause a failure, the system has to find an alternative value for the corresponding qualitative constraint.

The combination of quantitative and qualitative constraints, together with the Allen specific reasoning rules, are sufficient to solve our simple example, for which the output is shown in figure 4.5. Note that we only show one solution here, in reality there are multiple solutions.⁹

Typically, layout specifications of the presentation are specified in both qualitative and quantitative constraints. For example, alignment constraints, such as ‘meets’, ‘starts’, ‘finishes’ and ‘equals’ are completely defined by using solely qualitative constraints. However, ‘before’, ‘during’ and ‘overlaps’ constraints typically make use of an extra quantitative parameter which defines the (padding) distance between two items. This parameter can be used to

⁹On a 500 MHz Pentium III it takes approximately 220 seconds to find all 1536 solutions.

make sure two media items are centered but it can also be used to convey a visual grouping, when a group of items are ordered in such a way that the distance between all items is equal. In the later case the scope of the quantitative constraint is not solely between two media items but should range over a number of items. (Currently not implemented.)

```
% Note that the constraint variable Padding
% is used in both padding constraints.
foo(IdA,IdC,IdC) :-
    Padding :: [10..50]           % padding interval
    edge(IdA,IdB,x,b),
    setPadding(IdA,IdB,Padding),  % define padding constraint
                                   % between A and B
    edge(IdB,IdC,x,b),
    setPadding(IdB,IdC,Padding).  % define padding constraint
                                   % between B and C
```

Within the qualitative graph there are two types of relations. The first type is specified by the author. These relations are not necessarily completely determined but can have some variations, such as only allowing non-overlapping relations, in which case the domain consists of four values. The second type of relations are the implicitly determined values inferred by the system's reduction rules on Allen's temporal relations. Ideally, all domains are reduced to one value after all constraints are imposed, however, in practice this is almost never the case and therefore we have to choose a value for every variable with an associated domain larger than 1. There is, unfortunately, not an obvious heuristic to pick the most suitable value from its domain (see section 4.5 for more details), therefore the standard heuristic from ECLⁱPS^e is used, which picks values in alphabetical order.

From a designer's perspective, knowing the implicit relations between media items might be important information. For example, for exploring the possibilities within the specified constraints. We found out through experience that often implicit constraints should in fact be explicitly defined to gain the desired result. For this reason we developed a tool (see figure 4.6 and figure 4.7) intended for the designer. The tool gives a schematic overview by separately displaying the spatial and temporal dimensions of the presentation. The designer can exhaustively enumerate the possible alternatives within the defined constraints. This way, the designer can easily detect inappropriate behavior and add or adapt the constraint definition where needed.

4.4 Use of constraints within Cuypers

In previous sections we clarified the purpose of quantitative and qualitative constraints. In this section we elaborate on their practical use in the Cuypers system.

In chapter 2 we described the concept of a communicative device. This is a construct which describes, to a certain extent, how the information is presented, but leaves some degree of freedom to make it generally applicable. The *chiaroscuro* example in section 2.2.2 consists of two communicative devices. The 'adjacency device' which makes sure the items (or group of items) are displayed close to each other, and the 'bookshelf device', which displays all of its items ordered from left to right, and top to bottom. Ideally, an author of a presentation finds all of the communicative devices used in the presentation already implemented. However, for

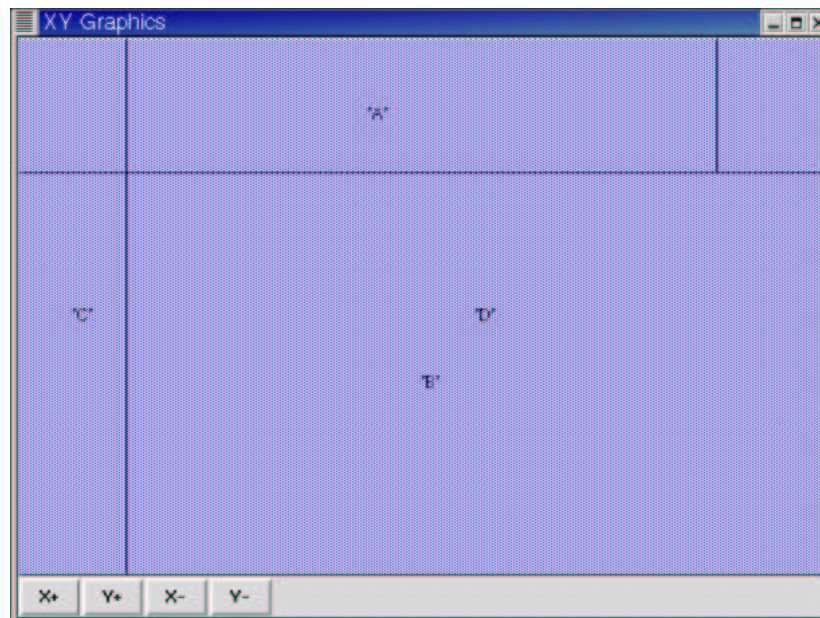


Figure 4.6: Spatial layout (X -axis from left to right, Y -axis from top to bottom.)



Figure 4.7: Temporal layout (T -axis from left to right.)

unimplemented devices the author has to specify in what way the items within this device, should be presented to the user.

Communicative devices are transformed into quantitative and qualitative constraints by formatting rules. A communicative device can have multiple formatting rules. This way, an alternative formatter can be applied once a previous formatter caused a failure. Formatters can be defined in such a way that they are capable of dealing with an arbitrary number of items which have unknown sizes. Consider, for example the ‘row’ formatter, which displays items ordered from left to right.

```
% row(+Ids)
%     formatter which creates a left to right ordering of items
row([]).
row([IdA|IdB|Ids]) :-
    edge(IdA,IdB,x,b),      # create a ‘before’ relation in the X dimension
    ...
    row([IdB|Ids]).
```

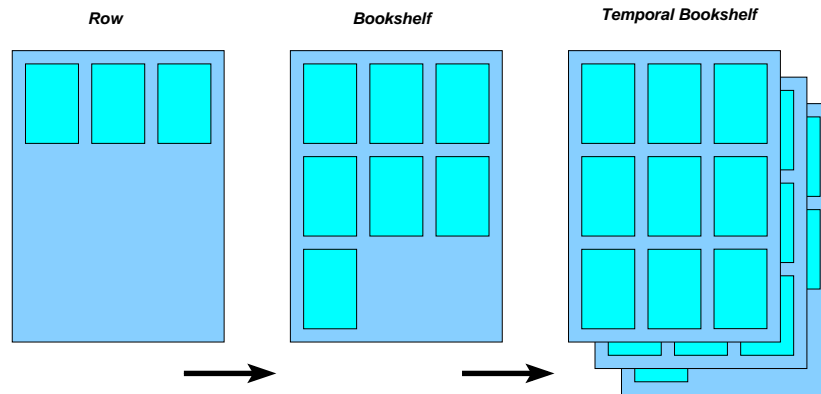


Figure 4.8: Bookshelf Communicative device

Within this formatter the qualitative constraint $\text{edge}(\text{IdA}, \text{IdB}, x, b)$ which states that the first item IdA is placed left of the second item IdB . The remaining items (Ids) are formatted recursively by employing the `row` formatter. Within a formatter, both qualitative and quantitative constraints can be specified. (Note that in the example above there are no quantitative constraints). If these constraints cause a failure this results in the failure of the formatter. The Prolog backtracking mechanism then triggers the application of an alternative formatter.

For example, if the items do not fit on a single row the formatter fails and an alternative formatter can be applied, which, for example, could order the items in time. A bookshelf formatter, used in the *'chiaroscuro'* example, uses the ability of a row formatter to fail to deal with an arbitrary number of media items. It uses Prolog backtracking to find out which items can be placed (in order) on a single row. The rest of the items are recursively formatted as a bookshelf and placed below the row (see figure 4.8).

```
% bookshelf(+Ids)
%   formatter creates a bookshelf ordering of items
bookshelf([]).
bookshelf(Ids) :-
    append2(Row, Rest, Ids), % Ids is the concatenation of Row and Rest,
                            % Row initially contains all items. If
                            % 'row' fails backtracking occurs in
                            % 'append2' which results in the
                            % transfer of the last item of Row to Rest

    row(Row),
    bookshelf(Rest),
    ...
    addNN(Row, Rest, y, b). % creates for every item in Row, on the
                            % Y dimension, a before constraint with
                            % every item in Rest
```

If, within the bookshelf formatter the media items do not fit on a single screen then the temporal bookshelf formatter can be used to order the items from left-to-right, top-to-bottom and eventually bookshelf-after-bookshelf. Note that this communicative device is very flexible: its formatters only fail if either one of the items is too large to fit on the entire screen, or the bookshelf exceeds the maximum duration of the presentation.

```

% temporal_bookshelf(+Ids)
%     formatter creates a bookshelf ordering of items and overflows
%     to the temporal dimension if needed.
temporal_bookshelf([]).
temporal_bookshelf(Ids) :-
    append2(Bookshelf,Rest,Ids),
    bookshelf(Bookshelf),
    temporal_bookshelf(Rest),
    ...
    addNN(Bookshelf, Rest, t, b). % creates for every item in Row a
                                % before constraint with every item in Rest

```

Although the recursive formatters are able to deal with an arbitrary number of items, the possible alternatives grow exponentially with the number of items. Formatters which are expected to deal with a large number of media items, such as the temporal bookshelf, consume many resources. The use of recursive formatters should thus be used with care.

4.5 Discussion

In the previous sections we gave an overview of the process of constraint solving and constraint logic programming in the context of multimedia presentation generation. We showed that the automatic generation of a multimedia presentation is feasible under these circumstances.

In this section we elaborate on some of the problems we encountered. We discuss some alternative solutions and the impact they have on the system. Finally we will discuss some flaws of the current system and how they can be improved.

4.5.1 Labeling — choice of candidate variable to label

Labeling is the mechanism used in constraint logic programming to choose a value from a domain of possible values and associate it with the domain variable. This is used to enable the system to make additional domain reductions, once there are no more possible reduction. Traditionally, a CSP problem is solved once all variables have a value which do not violate one of the defined constraints.

There are a number of different heuristics to choose a candidate variable. The ECLⁱPS^e default is to process variables in the order they are presented to the system. Other alternatives are to choose the variable which has the largest domain, the smallest domain or the variable which participates in the most constraints. The most suitable heuristic is dependent on the context of the CSP. Within the multimedia context we have two types of domains: qualitative and quantitative. In essence, the quantitative constraints represent the coordinates, and thus the position of the bounding boxes of the media items. The hierarchical ordering of the communicative device tree is reflected in the ordering of the bounding boxes. From this perspective, the ‘root’ bounding box (the whole presentation) is the best choice to label first since all other boxes are relative to this. For centering constraints in particular, which rely heavily on quantitative constraints, this decreases the search space. Relative positioning has a bad influence on this because the quantitative domains are absolute, which means the calculations of a centering have to be redone completely every time one of the involved items is assigned a new coordinate (due to a labeling). Still, the media items are represented by the leaf nodes in the communicate device tree which results in the media items being labeled last. This however is also not advised since the dimensions of the parent bounding box are only

known once the dimensions of its children are known. In order to detect whether the suggested presentation fits on the screen the dimensions of the child bounding boxes are essential. From this perspective, it is better to label the bounding boxes of the media items first.

Within the qualitative domain, there is no hierarchical ordering since the items are structured in a complete graph. From this perspective, choosing the most constrained variable would be an appropriate heuristic. Nevertheless, within the multimedia domain this can cause unexpected behavior. The complete graph consists of two types of relations, *explicit* which are defined by the author and *implicit* which are inferred by the systems reasoning engine. The labeling of an implicit variable (which has a domainsize > 1) can suggest a semantic relation which is in fact non-existing. For example a ‘starts’ relation aligns two media items which conveys a visual grouping. Subsequently, a separation between explicit and implicitly defined relations is advisory. First label the explicit defined relations according to the most-constrained-variable-first heuristic. This can reduce the domains of the implicit relations and therefore simplifies the choice for an implicit relation. There is no obvious heuristic to label implicit defined relations, in most cases a ‘meets’ relation will do for a first choice but this is definitely not true for all relations. Besides, the problem remains once there is no ‘starts’ in the domain. Alternatively, you can choose not to label implicit relations at all. This seems a plausible assumption, but if the implicit relation states there is no overlap (domain: [b,m,b-,m-]) between item A and B and there is no value assigned then this constraint is lost, since it cannot be translated to quantitative constraints. In this case overlapping items will be allowed, which is incorrect. For this reason the right answer seems to be a single constraint containing a translation for all qualitative domain values into quantitative constraints, separated by *or*’s. Since the *or* with the required properties is not in ECLⁱPS^e we did not implement this solution.

4.5.2 Labeling — choice of candidate value

Besides the choice of which variable is going to be labeled, the actual value from the domain associated with the variable is also determined by a heuristic. The default heuristics that ECLⁱPS^e uses for quantitative domains is to assign the smallest possible value to the variable. Within multimedia and left-to-right, top-to-bottom oriented document processing this is the desired default behavior; similarly for the temporal dimension. However, for right-to-left oriented document processing, such as in Arabic languages, a more appropriate behavior would be to start with the largest possible value. However, by consequently using the largest value from the domain the largest possible screen size is chosen by default. This can result in a waste of unused white space at the left side of the screen, which is not desirable. In this case, choosing the smallest possible value is a better heuristic for the ‘root’ bounding box. In fact, this argument also holds for the children of the ‘root’, and their children etc. In general, the size of a bounding box should be minimized and the labeling heuristic should be largest-values-first. In Cuypers, we use the default ECLⁱPS^e small-value-first heuristics for the quantitative labeling.

We briefly mentioned the problem of labeling implicit relations of qualitative variables previously. Implicitly defined relations should in principle not be labeled since the choice of a relation can unintentionally convey a semantic relation between media items.¹⁰ Ideally, after the labeling of explicitly defined relations, all the qualitative domain values should be

¹⁰One can argue that if labeling causes unwanted behaviour the CSP is under constrained.

translated into one single quantitative constraint. This way, no information is lost. We can then use the heuristics for quantitative labeling where there is no distinction made between implicitly and explicitly defined relations.

For the explicitly defined relations there is no specific order in a qualitative domain (besides, for example alphabetic, which does not relate to the context). The default heuristic of ECLⁱPS^e is alphabetic, which is used in Cuyper, but in principle any other heuristic is acceptable.

4.5.3 Labeling — best-first versus depth-first

The labeling procedure has a significant influence on the generated presentation. Although, there might not be an *a priori* preferred ordering in qualitative labeling, there might be one if we include design specific knowledge which takes into account the context of the to-be-labeled relation. For example, if we know that we are enumerating a list of items, and we know a list is preferably left-aligned, then a good heuristic is to first try a ‘starts’ relation which is used to left-align items. The search strategy changes then from a depth-first approach to a best-first approach. Note that the labeling procedure is not the only factor which influences the search strategy. The choice of rhetorical structure and communicative devices are, besides the labeling, the choice-points in the generation of a multimedia presentation. Typically a best-first approach chooses the best candidate by applying an evaluation function. We showed in chapter 2 that for multimedia presentation there is no “best” presentation. There is a trade-off between semantics, design and resources — what the optimal balance is between these factors is hard to specify, since it can differ for various contexts. An alternative approach can be the use of ‘global’ constraints, these are high-level constraints which are presentation independent. They encode good or bad features of a presentation. For example, having all labels below the corresponding media items would be desirable, and, for example, having more than 15 items on a screen would be discouraged. A numerical value for each constraint could be used to indicate how important it is. This value is comparable to the “badness” value in L^AT_EX [20]. The goal of the system is then to find an optimal value.

4.5.4 Labeling — structural grouping

Grouping is an important aspect in multimedia presentation generation. Within the definition of multimedia constraints, a grouping structure is therefore advisable. Often, media items are grouped for semantic reasons, but they are also used for design convenience (for example the ‘row’ grouping in the bookshelf communicative device in section 4.4). Grouping serves two purposes:

- Add specific features to the items within a group. For example: “all media items in the group are top-aligned.”
- Enable other media-items or groups to be positioned relative to this group. For example: “media item ‘A’ should be positioned above all media items within the group.”

Media items can be grouped in any number of dimensions within the presentation. For example, a grouping in the X, Y, T dimension creates a 3D-bounding box around all media items. Other media items or groups can be positioned relative to this group. Another example

is a grouping solely in the Y dimension, this way you can assure the media items are, for example top-aligned, no matter when they occur in the presentation.

Groups can be *implicitly* or *explicitly* defined. Explicit means that the box is treated as a (special) media item. Implicit means the box is merely the result of adding constraints between all of its members and another media item (or group). In principle, groups are “syntactic sugar” and for that reason they are not needed on the constraint level. Nevertheless, in some cases they are useful. An important reason is convenience for the author. The author of a presentation can, by adding a single constraint, position the group, whereas if there were no groups the author has to add the constraint for each individual media item within the group. Moreover, in some cases the members of a group are determined dynamically (eg. ‘row’ in a ‘bookshelf’), in which case the members of a group are typically unknown. Explicit definition of groups can for these reasons be beneficial.

Explicit groups can be defined at either the quantitative or qualitative level. If we consider the *qualitative* level, an explicit definition means the group is treated as a media item, and thus the `id` of the group is a node in the complete qualitative graph¹¹. All media items within the group have a ‘during’ relation with the group `id`. Because of the *transitive* reasoning rule all relations with respect to other media can be inferred.

On the *quantitative* level, groups are represented by bounding boxes. The dimensions of the box are determined by the minimal and maximal coordinates of the media items within the group. The coordinates of the bounding box are constraint variables (just as the coordinates of media items). The labeling of quantitative bounding boxes should be bottom-up, since the dimensions of a box are dependent on its children. Quantitative boxes can be used to, for example, center a group of media items.

4.5.5 Redundancy of the Transitive reasoning rule

Within the qualitative constraints domain the transitive reasoning rule is used to infer the implicit relations between the media items in the presentation. We argued that the labeling of implicit relations leads to undesired results (see section 4.5.1) and should, in principle be avoided. This means that the values of implicitly defined relations are not used in the generation process. Therefore, besides consistency checking, the transitive rule serves no purpose.¹² As shown by Kautz [18], qualitative specifications of the Allen temporal relations translate with minimal loss of information to quantitative specifications. A transitive relation on a qualitative level remains transitive in a quantitative translation. Inferring the transitive relation on a qualitative level and translating this to a quantitative level is thus redundant. The transitive rule, which is expensive in terms of performance can (under certain circumstances¹³) be omitted.

Note that this argument does not make the use of qualitative constraints as a whole obsolete. Besides the more intuitive specification of qualitative constraints, the domains are smaller and thus more efficient. Moreover, when in the future more extensive reasoning is needed to, for example, add global consistency constraints¹⁴ the transitive reasoning rule is

¹¹All media which are used in the presentation are known beforehand. The complete graph can therefore be created at initialization. The size of the graph is then known and cannot get larger. Groups, however, are dynamically created and are not known *a priori*. Therefore the complete graph cannot be created at initialization.

¹²If we use explicitly defined groups on a qualitative level, the transitive rule is required.

¹³When there are no explicit qualitative groupings.

¹⁴E.g. ensure all examples are presented in the same way.

required. Finally the redundancy of the transitive relation is mainly due to the data structure used, which is a complete graph. Although this structure increases expressibility,¹⁵ another less redundant datastructure could be used instead.

4.5.6 Allen’s temporal relations

Allen’s temporal relations are primarily intended for descriptive reasons. We found out through experience that the use of Allen’s temporal relations for generative purposes might not be the best choice.

All possible relative positions between two media items can be expressed by using the Allen relations. Besides the ‘before’, ‘meets’ and ‘overlaps’ relations the Allen relations contain information about the length of the media items as well. For example, the relation ‘A starts B’ requires that A is shorter than B. Within the use of the Allen relations in multimedia presentation generation the alignment of media items is important and not necessarily the length of the items. The Allen relations, however, force us to make a distinction on the basis of the length of the items which serves no purpose. For example, in order to specify a left-alignment relation between items A and B we need to distinguish three cases: A is smaller than B, A is larger than B and A and B have equal length. Since we do not know the lengths of A and B beforehand we have to specify all three cases, which, besides being inconvenient, is also inefficient.

Furthermore, the Allen relations do not provide the functionality of defining runtime dependency relations between media items. For example, the audio fragment which accompanies an image should only be played when the image is visible. In particular, dependency relations are required for allowing inter-activity.

4.5.7 Generalized Propagation — Propia

In addition to *Constraint Handling Rules (CHR)*, *Generalized Propagation (GP)* [22] (implemented in the Propia library of ECLⁱPS^e) can also be used to implement user defined constraints. The main idea behind GP is to use whatever constraints are available on the computation domain to express restrictions on the variables. Because of this, GP tends to be more efficient than CHR. Within this thesis we focused on the use of CHR. Further research is needed to find out to what extent GP (in combination with CHR) can be used in multimedia presentation generation.

4.5.8 Performance

Although the focus of this thesis¹⁶ is not to research how to create the most efficient system to generate a multimedia presentation, performance and efficiency is definitely an issue. Because of the computationally explosive character of positioning multimedia items there are some considerations which need special care. First of all, the number of media items influences the performance. Because of performance reasons, within Cuypers the number of media items did not exceed 20. For a serious presentation, the number of media items can easily grow to more than 100. Nevertheless, one cannot say that the performance relies only on the number

¹⁵A complete graph allows constraint specification between any pair of media items. Within, for example, a tree structure it is in general not possible to add constraints between leaves which are on different branches.

¹⁶see section 1.2

of items. There are often more influential parameters. In general, the level of generality consumes a lot of resources. For performance reasons it is thus a good thing to limit the number of possible presentations. An author can, for example, identify beforehand on which platforms the presentation can be played and tailor the alternatives for solely these specific platforms. Although the presentation is no longer completely device independent, the gain in performance might be of greater importance.

Another important factor which consumes resources is formatters which take an arbitrary number of media items. In particular, for large numbers of media items, the number of backtracks should be kept as small as possible (the ‘bookshelf’ formatter is thus not very efficient for this reason).

Finally, constraint based alignment relations, such as centering, can be inefficient. This particularly becomes an issue when groups of media items are being aligned.

The generation of the Chiaroscuro presentation from chapter 2 takes about 7 seconds on on a 500 MHz Pentium III PC. In comparison, the example we used in this chapter takes approximately 25 seconds on the same machine. The difference between the two example is the degree of freedom. The first example is quite static, in the sense that the temporal and spatial relations between the media items are mostly known beforehand. Within the second example however, these relations are not known and should be determined by the system.

4.6 Conclusion

In this chapter we showed that constraint logic programming can successfully be applied in multimedia presentation generation.

CLP combines the paradigms of constraint solving and prolog. This results in a system which provides an alternative for the inefficient generate-and-test approach often imposed by constraint solvers. By using the ability to backtrack we can, once we detect an inconsistency, immediately impose an alternative constraint without ‘resolving’ the entire set of constraints.

Furthermore, we showed that quantitative constraints on their own are insufficient because of their limited expressibility, large inefficient domains and their inability to allow an intuitive specification of multimedia constraints. Qualitative constraints use non-numerical domains and can therefore specifically be tailored for a multimedia domain. This requires, however, the explicit definition of domain reduction rules within the multimedia domain. We used three types of rules: equality, inverse and transitivity. Because multimedia presentations have both quantitative and qualitative properties we need a combination of both types of constraints. Qualitative constraints translate automatically into quantitative constraints and make the combination feasible. Because of redundancy between the quantitative and qualitative constraints the transitive rule becomes obsolete in cases where there are no explicit qualitative groupings.

The automatic generation of multimedia presentations is feasible by using constraint logic programming and a combination of both quantitative and qualitative constraints. Nevertheless, to improve performance and authoring convenience the grouping construct and labeling procedures need extra attention.

Chapter 5

Conclusion

In this thesis we focused on the use of constraints for automatic generation of multimedia presentations.

Authoring a multimedia presentation is not a linear process. This holds for a human author as well as for an automatic system. During the process, choices are made which, at a later stage, might need to be revised. The need for revision is inevitable because a multimedia presentation has to satisfy requirements of various natures, including, design, semantics and resources. Requirements often contradict each other and therefore trade-offs need to be made.

In contrast to manually authoring, an automatic system needs to abstract from media items in order to construct a generally applicable framework. Since retrieved media can be of any type or size, a document cannot be specified by using traditional *WYSIWYG* techniques, where the author exactly specifies the position of the media. Instead, higher level specifications in terms of semantics and rhetoric are needed to overcome this problem. Despite this, the transformation from high level semantic structure to final form presentation still has to be made by the system, once the media items are known.

The thesis focuses on the transformation of high level semantics to the final form presentation. That is, we researched the use of constraints and constraint logic programming to make this transformation efficiently computable and moreover create an intuitive and expressive interface for the author.

Our main results are that, besides the use of quantitative constraints to encode the numerical aspects of a presentation, qualitative constraints are needed to enable high-level reasoning about the structure of the presentation. Moreover, qualitative constraints are more intuitive to the author and more efficient to compute. Quantitative and qualitative constraints are dependent on each other which result in particular labeling heuristics. Furthermore we established that backtracking is needed in order to efficiently revise a constraint once it causes a failure. Finally, we argued that the transitive reasoning rule, in general, has no additional value and can therefore be removed from the reasoning engine.

Although this thesis shows that constraint logic programming can be successfully applied to multimedia presentation generation, the need to research this topic could not be considered finished. This mainly is due to newly gained insights about how design, rhetoric and semantics influence the presentation structure. These insights require additional features of the constraint specifications and may require a different structure and behavior of constraints.

Nevertheless, we argue that constraints and constraint logic programming have specific

characterizations, which make them well suited for automatic presentation generation

Appendix A

CHR rules

A.1 Domain Reduction Rules

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

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

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

% 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

    IdA \= IdB, IdA \= IdC, IdB \= IdC % 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
```

A.2 Translation rules

```
EdgeBefore ::=
    edge(IdA, IdB, Dim, b)
    ==>
        node(IdA:Dim/2,AD2),
        node(IdB:Dim/1,BD1),
        AD2 #< BD1.
```

```
EdgeDuring ::=
```

```

edge(IdA, IdB, Dim, d)
==>
  node(IdA:Dim/1,AD1),
  node(IdA:Dim/2,AD2),
  node(IdB:Dim/1,BD1),
  node(IdB:Dim/2,BD2),
  AD1 #>= BD1,
  AD2 #<= BD2.

EdgeOverlaps ::=
edge(IdA, IdB, Dim, o)
==>
  node(IdA:Dim/1,AD1),
  node(IdA:Dim/2,AD2),
  node(IdB:Dim/1,BD1),
  node(IdB:Dim/2,BD2),
  AD1 #< BD1,
  AD2 #> BD1,
  AD2 #< BD2.

EdgeMeets ::=
edge(IdA, IdB, Dim, m)
==>
  node(IdA:Dim/2,AD2),
  node(IdB:Dim/1,BD1),
  AD2 #= BD1.

EdgeStarts ::=
edge(IdA, IdB, Dim, s)
==>
  node(IdA:Dim/1,AD1),
  node(IdA:Dim/2,AD2),
  node(IdB:Dim/1,BD1),
  node(IdB:Dim/2,BD2),
  AD1 #= BD1,
  AD2 - AD1 #< BD2 - BD1.

EdgeFinishes ::=
edge(IdA, IdB, Dim, f)
==>
  node(IdA:Dim/1,AD1),
  node(IdA:Dim/2,AD2),
  node(IdB:Dim/1,BD1),
  node(IdB:Dim/2,BD2),
  AD2 #= BD2,
  AD2 - AD1 #< AD2 - BD1.

EdgeEquals ::=
edge(IdA, IdB, Dim, e)
==>
  node(IdA:Dim/1,AD1),
  node(IdA:Dim/2,AD2),
  node(IdB:Dim/1,BD1),
  node(IdB:Dim/2,BD2),
  AD1 #= BD1,
  AD2 #= BD2.

```

Appendix B

Summary of Cuypers

Cuypers is a multimedia presentation generation engine based on constraints. A semantic description, modeling the intended message of a presentation, is transformed automatically into a multimedia presentation. During this process the presentation is adapted to specific requirements which involve system hardware capabilities, user interests and user preferences.

Within the Cuypers engine there are presentation-independent rules, which include design rules, semantic rules and resource rules. Each rule generates a set of constraints on the available media items. Ideally, once all rules are applied and the constraints are solved, a presentation can be generated. In practice, however, these rules generate conflicting constraints. Therefore a trade-off between design, semantics and resources is needed.

A rule takes a rhetorical relation as input and generates a multimedia construct which conveys the intended relation. Typically, there are multiple ways to generate the same effect. For example the construct ‘sequence’ can be conveyed by a left to right ordering or by a temporal ordering where one is played after the other. During the generation process Cuypers invokes alternatives once it finds out it cannot satisfy a set of constraints.

The number of alternatives grows exponentially large as the number of media items grows. To cope with this problem we need a system which intelligently reduces the search space by omitting possible values which will never lead to a solution. Moreover, once it detects an inconsistency, it should trigger the rule which caused the inconsistency to generate an alternative. Constraint Logic Programming (CLP) addresses both these problems by combining the paradigm of constraint solving (domain reduction) with prolog unification and backtracking.

Modeling constraints for multimedia does not match traditional arithmetic constraints on integer numbers. For example, the constraint which states that two media items should not overlap is not expressible by a single arithmetic constraint. Moreover, the constraint ‘A immediately follows B’ is not entirely similar to ‘A plays from 1 to 5 sec.’ and ‘B plays from 5 to 10 sec.’ since a delay of A does not cause a rescheduling for B in the second case. For this reason we need to define our own type of multimedia constraints (qualitative) and specify how to reason with them.

Besides qualitative constraints we still need arithmetic constraints because media items have quantitative properties, such as width, height and duration. To check whether a configuration of media items fits on a screen or does not violate a certain time constraint we need to translate qualitative constraints into quantitative constraints.

Although Cuypers successfully generates a presentation conforming to the specified con-

straints it is still in an early phase of development and most processing layers could be improved. Currently, we use a depth-first approach were a best-first approach would be more beneficial. Furthermore, our research focuses on the identification of different levels of presentation abstractions and to what extent presentation independent rules can be applied. Finally domain knowledge is hardwired on different levels into the presentation engine. We should identify these and query a knowledge base or ontology instead. However the type of knowledge that is required to generate a presentation is a topic of further research.

Bibliography

- [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. Principles of Constraint Programming, January 2000. Lecture Notes for Constraint Logic Programming Course, University of Amsterdam.
- [4] 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.
- [5] G.J. Badros and A. Borning. Cassowary: A Constraint Solving Toolkit, 1999.
- [6] Brian P. Bailey, Joseph A. Konstan, and John V. Carlis. Supporting Multimedia Designers: Towards More Effective Design Tools. In *Proc. Multimedia Modeling: Modeling Multimedia Information and Systems (MMM2001)*, pages 267–286. Centrum voor Wiskunde en Informatica (CWI), 2001.
- [7] 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.
- [8] V. M. Calegario and I. C. Dutra. Performance Comparison between Conventional and Logic Programming Systems. Technical report, Federal University of Rio de Janeiro, 1998.
- [9] Andy Hon Wai Chun. Constraint Programming in Java with JSolver. In *Proceedings of PACLP99 The Practical Application of Constraint Technologies and Logic Programming*, April 1999.
- [10] James Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendations are available at <http://www.w3.org/TR/>, 16 November 1999.
- [11] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendations are available at <http://www.w3.org/TR/>, 16 November 1999.

- [12] Jon Ferraiolo. Scalable Vector Graphics (SVG) 1.0 Specification. W3C Candidate Recommendations are available at <http://www.w3.org/TR>, 2 November 2000.
- [13] Thom Frühwirth. Theory and Practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1-3), October 1998. Special Issue on Constraint Logic Programming, P. Stuckey and K. Marriot, Eds.
- [14] Joost Geurts, Jacco van Ossenbruggen, and Lynda Hardman. Application-Specific Constraints for Multimedia Presentation Generation. In *Proceedings of the International Conference on Multimedia Modeling 2001 (MMM01)*, pages 247–266, CWI, Amsterdam, The Netherlands, November 5-7, 2001.
- [15] Lynda Hardman. *Modelling and Authoring Hypermedia Documents*. PhD thesis, University of Amsterdam, 1998. ISBN: 90-74795-93-5, also available at <http://www.cwi.nl/~lynda/thesis/>.
- [16] International Organization for Standardization/International Electrotechnical Commission. Information technology — Processing languages — Document Style Semantics and Specification Language (DSSSL), 1996. International Standard ISO/IEC 10179:1996.
- [17] 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.
- [18] Henry A. Kautz and Peter B. Ladkin. Integrating Metric and Qualitative Temporal Reasoning. In *Ninth National Conference on Artificial Intelligence (AAAI-91)*, Anaheim, California, July 14-19, 1991.
- [19] Donald E. Knuth. *TeX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley Publishing Company, 1986.
- [20] Leslie Lamport. *LaTeX - A Document Preparation System*. Addison-Wesley Publishing Company, 1985.
- [21] 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.
- [22] Thierry Le Provost and Mark Wallace. Generalised Constraint Propagation Over the CLP Scheme. Technical report, ECRC, 1993.
- [23] Gustavo Rossi, Daniel Schwabe, and Alejandra Garrido. Design Reuse in Hypermedia Applications Development. In *The Proceedings of the Eighth ACM Conference on Hypertext and Hypermedia*, pages 57–66, Southampton, UK, April 1997. ACM, ACM Press.
- [24] 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.

- [25] Lloyd Rutledge, Jim Davis, Jacco van Ossenbruggen, and Lynda Hardman. Interdimensional Hypermedia Communicative Devices for Rhetorical Structure. In *Proceedings of the International Conference on Multimedia Modeling 2000 (MMM00)*, pages 89–105, Nagano, Japan, November 13-15, 2000.
- [26] 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.
- [27] Jacco van Ossenbruggen, Frank Cornelissen, Joost Geurts, Lloyd Rutledge, and Lynda Hardman. Cuypers: a semi-automatic hypermedia generation system. Technical Report INS-R0025, CWI, December 2000.
- [28] 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.
- [29] 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.
- [30] W3C. XHTML 1.0: The Extensible HyperText Markup Language: A Reformulation of HTML 4.0 in XML 1.0. W3C Recommendations are available at <http://www.w3.org/TR/>, January 26, 2000.
- [31] W3C. Synchronized Multimedia Integration Language (SMIL 2.0) Specification. W3C Recommendations are available at <http://www.w3.org/TR/>, August 7, 2001. Edited by Aaron Cohen.
- [32] Mark Wallace, Stefano Novello, and Joachim Schimpf. ECLiPSe: A Platform for Constraint Logic Programming, 1997.