Specifying Diagram Animation with Rewrite Systems

Bernd Meyer and Kim Marriott

June 8, 1997

Abstract

The complete specification of visual languages often requires to capture not only the static aspects of syntax and semantics but also to describe dynamic aspects like the animation or execution of visual expressions or user interaction with such notations. Formal specification frameworks for visual languages have so far mainly been targeted for static language aspects. We present a rewrite framework for the formal specification of diagram animation which is based on the combination of graph rewriting and constraint solving. Since it can be regarded as an extension to constraint-based multidimensional grammar frameworks, it is suitable for the integration with previous approaches to the formal specification of visual language syntax and parsing.

1 Motivation

Many applications of visual languages involve dynamically changing diagrams. Such change mainly arise from two sources: (a) in interactive languages change may be caused by a user interaction or (b) the task itself requires graphical transformations. Such graphical transformations can, for example, describe valid editing actions in a syntax-based diagram editor or they can be used to formalize so-called "imagery" reasoning processes, i.e., processes in which the reasoning steps themselves are given by graphical transformations of images [GNC95]. A prime example for such "imagery" reasoning are visual mathematical notations in which the evaluation rules or the consequence relations are given by visual transformations. Examples of such notations are Shin's extended Venn systems [Shi95], Hammer's heterogeneous relation representations [HD93, Ham93], VEX [CHZ95], Lambda Birds [Kee95], boundary arithmetics and logic [Bri88], and Harel's state charts [Har88]. The same principle is at the heart of some "completely visual" programming languages, like Pictorial Janus [KS90] and VIPR [CDZ94] whose execution semantics is fully defined by visual transformations.

Unfortunately there has been relatively little work into how to formalize or specify dynamically changing diagrams. Formal specifications of visual or diagrammatic languages have mainly considered static notations and have mostly been constructed to support one-step processing tasks like parsing or syntax-directed translation. As a consequence, most existing visual language formalisms are based on grammars. Indeed, a common argument against formal approaches to visual languages is that such static formalizations are essentially misguided, because most visual languages are used in interactive applications and are thus dynamic. While in our opinion grammar-based methods do merit a central place in visual language theory, it has to be admitted that current approaches do not sufficiently support dynamic notations. On the other hand, the advantages of grammar-based approaches are obvious: they provide a clear and well-formalized foundation of a theory of visual languages and they open the way for automated construction of visual language environments from declarative specifications. In this paper we address how to extend grammar-based approaches to the specification of dynamically changing diagrams.

To define and animate the execution of such notations in a well-defined manner we need to be able to specify visual diagram transformations in exact, unambiguous ways. Currently there are two main approaches to the grammatical specification of static visual languages. The first is based on graph rewriting and relies on representing a diagram by a graph whose nodes are the objects in the diagram

and the edges are the relations, such as *left_of* or *above*, between these nodes. The second is based on attributed multiset grammars. In this approach diagrams are regarded as a multiset of objects with attributes which detail their appearence and position. We will show that neither of these formalisms is appropriate for specification of diagram transformation. Rather a combination of the two approaches is required.

It has to be noted that not even a simple combination of these methods is always fully sufficient if cognitive aspects or aesthetic criteria have to be taken into account. An often cited rule for dynamic layout, for example, is to minmimize the "moment of surprise" for the viewer or, in other words, to preserve the viewer's "mental map". One technical requirement that can be derived from this rule is that the global arrangement (for example, object positions) should change as little as possible between subsequent phases of an animation. Such global layout criteria are often difficult to achieve in grammatic or rule-based frameworks, since these rely on the manipulation of local structures. A solution to this problem can be found in the integration of specialized layout modules, such as stochastic methods for graph layout, into dedicated output phases of the animation process.

The remainder of the paper is structured as follows: The rewrite formalism will be introduced in four steps: Section 2 presents the basic variant that allows to specify simple animations like the execution of state transition diagrams. In Section 3 we define an extension that allows the modification of spatial relations with unbounded degree. Section 4 examines the problem of concrete geometric layouts and defines how to handle them by constraint solving techniques. Finally, the problem of geometric inconsistencies that can arise from the usage of concrete geometries is analysed in Section 5 and a final version of the formalism that allows to handle such inconsistencies is defined. The basis for the implementation of our model is discussed in Section 6. Section 7 concludes.

2 Specifying Simple Animations: ARS_1

Consider the two approaches to grammatical specification of static visual languages. One approach is graph based. In this view we regard a picture as constructed of objects and spatial relationships according to some vocabulary (OT, RT) of object types and relation types. A picture is therefore represented by a graph with typed object nodes and typed relationship edges. Figure 1 shows some picture and its corresponding graph assuming the vocabulary $(\{circle, line, label\}, \{attached, touches\})$.

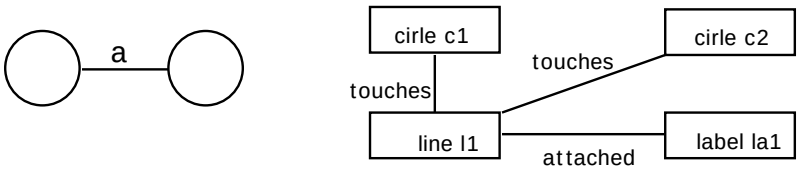

Figure 1: A Diagram and the Corresponding Graph Structure

Now consider a simple example of an animation specification, namely the execution of an NFA. The string to be accepted (or rejected) is given by a textual label inside of the circle representing the current state. A valid transition is thus informally described by the production in Figure 2, which has to be interpreted as "if the diagram contains a circle C_1 in which some string L_2 is located, and some arrow starts at C_1 and ends at another circle C_2 and is labeled with a string L_1 such that the first character of L_2 is given by L_1, then remove the first character from L_2 and move L_2 from C_1 into C_2.[1]

The problem with this approach is that we also need attributes which specify how to draw the new picture, and these attributes depend on those of the original picture. Thus every picture object

[1]This does obviously not describe loop transitions from C_1 to C_1.

Figure 2: An Example Production for NFA Transitions

needs an arbitrary number of attributes which carrying additional information for geometry handling or interpretation purposes.

The other approach to visual language specification relies on representing diagrams as a multiset of objects with attributes. In this approach we could formalize the above transformation as before but replace the tests for relationships by tests on the attributes of the objects. The rewriting can be done by just changing the coordinates of L_2 to move it from C_1 to C_2 without explicitly manipulating any relationship.

However this approach does not suffice for more complex rewriting. Imagine that we have a circle with an arbitrary number of circles attached to its perimeter. We wish the central circle to expand and for the attached circles to remain attached. Something very like this corresponds to procedure execution in Pictorial Janus. There is no easy way to formalize this transformation in the attribute based approach. We essentially have to provide a rule for each possible number of attached circles.

The attribute-based approach also presents another problem, which can informally be described as the "individual accessibility of spatial relations" and is related to only partially specified states of a diagram. It is obvious that no method which relies on fully instantiated values of attributes can capture partially specified states of a diagram. Attribute-based approaches therefore usually rely on constraint solving methods to describe partially specified geometries. The problem arises from the fact that in most constraint solving methods no constraint, once it has been asserted, can be accessed seperately from the other constraints with which it shares variables. In particular most arithmetic constraint solvers, which we need to handle geometric problems, maintain their constraint stores in some "solved form" which makes individual constraint inaccessible. Now consider the case where more than a single spatial relationship have been imposed on some object and constrain the value of some attribute $X.a$ say, $r(X.a, Y.b)$ and $q(X.a, Z.c)$. If we want to modify only one of them in some animation step, say $r(X.a, Y.b)$ and the constraints of $X.a$ are accessible only in solved form, then it is impossible to break $r(X.a, Y.b)$ while at the same time maintaining $q(X.a, Z.c)$. We need to have some symbolic representation of the individual constriants in order to be able to perform such a step.

Another reason for maintaining individual symbolic representations of constraints is that it is not possible to distinguish between *necessary* and *sufficient* spatial conditions in an attribute-based approach if the applicability of a production is only checked by testing whether the spatial conditions on the left-hand side of a production are consistent with the current diagram.

From the above we see that in order to be able to express an animation step we need to be able to modify objects as well as relations, i.e., it must be possible to delete or add spatial relations explicitely. We have also seen that it is neccessary to have some concrete geometrical interpretation of spatial relations which we will handle as artithmetic constraints on the objects' attributes. Let us defer the problem of spatial relation interpretation for now and turn to the basic rewrite model first. The concrete interpretation of relations and the integration of constraint solving will then be introduced in Section 5.

Since we need to manipulate objects as well as relations explicitely, the general form of a production is:

$$O_1, \ldots, O_n \text{ with } R_1, \ldots, R_m \diamond C_{before} \Rightarrow O'_1, \ldots, O'_{n'} \text{ with } R'_1, \ldots, R'_{m'} \diamond C_{after}$$

where O_i are the objects and R_i the relations to be rewritten, and O'_i and R'_i are the objects and relations, respectively, into which they are rewritten. C_{before} is an additional condition on the attributes of $O_1 \ldots O_n$ that has to be true for the production to be applicable. When the production is applied,

the LHS entities are removed from the picture, the RHS entities are inserted and the objects' attributes are updated according to C_{after}.

The above NFA example production can thus be formalized as:

$$\begin{array}{l} C_1, C_2 : circle, A : arrow, L_1, L_2 : text \text{ with} \\ \quad inside(L_2, C_1), connects(A, C_1, C_2), attached(L_1, A) \\ \quad \diamond\ L_1.string = first(L_2.string) \\ \Rightarrow\ C_1, C_2 : circle, A : arrow, L_1, L_2 : text \text{ with} \\ \quad inside(L_2, C_2), connects(A, C_1, C_2), attached(L_1, A) \\ \quad \diamond\ L_2.string = rest(L_2.string) \end{array}$$

We summarize in the following definitions:

Definition 1 *A picture vocabulary is a tuple $V = (OT, RT)$ of a set OT of object types and a set RT of spatial relation types. A diagram over V is a tuple $D = (O, R)$ of a set O of objects and a set R of relations with types according to V.*

Definition 2 *An ARS_1 production system is a tuple (V, P) of a picture vocabulary V and a set P of production rules of the form*

$$O_1, \ldots, O_n \text{ with } R_1, \ldots, R_m \diamond C_{before} \Rightarrow O'_1, \ldots, O'_{n'} \text{ with } R'_1, \ldots, R'_{m'} \diamond C_{after}$$

where $O_1, \ldots, O_n, O'_1, \ldots, O'_{n'}$ are typed variables over OT; $R_1, \ldots, R_m, R'_1, \ldots, R'_{m'}$ are typed relations over RT; C_{before} is a boolean expression over the attributes of $O_1, \ldots, O_n$ and C_{after} is an assignment to the attributes of $O'_1, \ldots, O'_{n'}$.

Definition 3 *A production $P \equiv X$ with $Y \diamond C \Rightarrow X'$ with $Y' \diamond C'$ of an ARS_1 can be applied to a diagram $D = (O, R)$ if there is a variable assignment θ such that $O = X\theta \cup A$, $R = Y\theta \cup B$ and $C\theta$ evaluates to true. The application of P to D generates the diagram $D' = (O', R')$ with $O' = A \cup X'\theta$, $R' = B \cup Y'\theta$ in which the attributes of objects in O' are modified according to C'. Written $D \stackrel{P}{\rightarrow} D'$.*

3 Adding Relation Closures: ARS_2

The NFA transformation was a very simple example of a transformation and thus easy to specify. In particular it involved only changing a single objects (the text) and the number and type of relations in which this object can be involved where known beforehand (*inside*).

What happens, if we do not know the number and types of relations in which an object that has to be changed is involved? Consider, for example, the case where all nodes in the NFA (which can have arbitrary many arrows attached) have to be replaced by some other object (say, a rectangle). A simple rule like

$$X : circle \text{ with } startsat(X, Z) \diamond true \Rightarrow Y : rect \text{ with } startsat(Y, Z) \diamond nil$$

will not work for two reasons. (1) only a single instance of $startsat(X, Z)$ will be changed, but there may be arbitrarily many instances for other connections and (2) no other relation types (for example, *endsat* or *inside*) will be changed. Including these into the rule like in

$$X : circle \text{ with } endsat(X, Z1), startsat(X, Z2) \diamond true \Rightarrow Y : rect \text{ with } endsat(Y, Z1), startsat(Y, Z2) \diamond nil$$

would not change the situation: (1) Still the number of relation instances to be modified is fixed, (2) if some object does not have an *endsat* or a *startsat* relation, the rule is not applicable. The situation becomes even worse, if the types of relations in which an object may be involved are not known.

The solution to this problem is provided by an extension of the rewrite formalism which allows to indicate that some object Y is a replacement for an old object X. This will be indicated by an optional substitution $\{X/Y\}$ at the end of the production and is technically treated as a variable substitution that is applied to the relation set after the production application. Such we can resolve the above problem by writing a modified rule:

$X : circle$ with $\emptyset \diamond true \Rightarrow Y : rect$ with $\emptyset \diamond nil \diamond \{X/Y\}$

This rule will be applied in the usual way, thus removing a single circle X from the diagram and inserting a new rectangle Y. After this, the variable substitution $\{X/Y\}$ is applied to the relation set of the new diagram replacing all instances of X by Y. Thus every relation $r(X, Z)$ is rewritten as $r(Y, Z)$ so that after the production application Y participates in all relations in which X participated before.

4 Integrating Geometry with Constraints: ARS_3

Let us now turn back to our original example of animating an NFA. Assume we want to visualize such an animation by displaying the complete diagram after each rewrite step. The problem now is that we need a completely instantiated concrete geometry for the picture in order to be able to display it. So far we have only been handling abstract, symbolic objects and relations and have not looked at geometries. A naive approach to solve this problem would work in the following way: we assume that we start with a concrete diagram, i.e., with a diagram in which all relevant geometric attributes are instantiated with concrete values. In order to obtain a sufficiently instantiated diagram after each rewrite step, it is then only necessary to assign values to the attributes of new or changed objects. Applying this schema we would only have to change the above transformation rule to:

$C_1, C_2 : circle, A : arrow, L_1, L_2 : text$ with
$inside(L_2, C_1), connects(A, C_1, C_2), attached(L_1, A)$
$\diamond\ L_1.string = first(L_2.string)$
$\Rightarrow\ C_1, C_2 : circle, A : arrow, L_1, L_2 : text$ with
$inside(L_2, C_2), connects(A, C_1, C_2), attached(L_1, A)$
$\diamond\ L_2.string = rest(L_2.string), \mathbf{L_2.center = C_2.center}$

However, we have already argued above that such a simple approach which is based on fully instaniated attribute values is in general not a good idea, since we often need to be able to express partially specified geometries. In particular, it is not always possible to immediately determine the absolute coordinates of an object. In the above case, e.g., we might only know that L_2 has to be inside of C_2 but not where exactly it is located within C_2. Putting it simply into the center might generate a layout that conflicts with other objects which could be put into C_2 later. Therefore the ideal method is to integrate partially specified geometries by using constraint solving for the objects' attributes. To integrate this into the rewrite system we introduce *interpretations* of spatial relations which are defined as arithmetic equations over the objects' relevant attributes.

Not all symbolic relations have to be interpreted. We may, for example, have several ways of graphically expressing that some object is a subobject of another, but we may still want to capture the fact that some object O_1 is a subobject of O_2 during some intermediate phase in an abstract relationship $subobject(O_1, O_2)$ without immediately giving it a graphical interpretation. Thus every relation type in the vocabulary can either have an interpretation or be uninterpreted (in which case it is handled as a purely abstract relation object like above).

An interpretation is defined as an arithmetic constraint over attributes of the involved objects. For example, the $inside : label \times circle$ relation can be interpreted as:

$$inside(L, C) \equiv dist(L.topleft, C.center) < C.radius \wedge dist(L.bottomright, C.center) < C.radius$$

This interpretation of *inside* enforces the fact that the label is completely contained in the circle, but does not give it a definite position. To integrate the interpretations into the rewrite mechanism we need to employ an arithmetic constraint solver. Every diagram now consists of three components: as before, it consists of a set O of objects and a set R of relations. Additionally we now have a set C of constraints attached to the attributes of objects in O. Let I_r denote the interpretation of a relation r, then $C = \wedge_{r \in R} I_r$.

Now given some interpreted relations we must find an assignment of values which satisfy those relations viewed as constraints. If the constraints have a unique solution we are done. However, the constraints may have many solutions or no solutions at all. The case when there are no solutions, because the attributes are "over constrained" is crucial in the case of diagram animation, since intermediate animation steps often refer to geometrically infeasible diagrams. A solution to this problem is presented in the next section. The case when there is more than one solution arises because our constraint system is "under specified". The same problem often occurs in constraint based graphics layout. If a diagram which is under specified has to be displayed there are basically three ways to arrive at a concrete layout: (a) The constraint solver can be forced into an arbitrary feasible solution. This can, for example, be achieved by interval splitting methods if the solver is based on interval arithmetics. However, this approach is not a particularly attractive solution since there is very little control over the final layout. (b) A specialized layout solver (which is different from the main solver used during the rewrite phase and may, for example, use complex stochastic methods) can be used to generate a concrete layout from an under specified system. Still, it is difficult to ensure that two slight variants of the same under constrained diagram which may occur in subsequent animation steps have similar layouts. (c) The best solution is to employ hierarchical constraints. In this approach the user specifies additional constraints with weighted preferences, which do not influence the semantics of the diagram but only its layout. Semantically such constraints can be regarded as redundant and they ususally specify criteria like "an attribute should remain as it is" or "the distance between related objects should be minimal". The solver uses these constraints to arrive at a unique solution, but is free to ignore less preferred constraints if they cause an over specified system that would not have a solution. In this way general layout styles can be specified, but the animation specification can still focus on the semantically relevant steps of a diagram transformation.

A constraint-based production application consists of three steps: (1) The LHS of the production is matched against the current picture. (2) If the match succeeds, all LHS objects and relations are removed from the picture. (3) All interpretations of relations on the LHS are removed from the constraint store. (4) The RHS objects and relations are added to the current diagram. (5) All interpretations of relations on the RHS are added to the constraint store. In this model, a rule has to be applied tentatively, because it can, of course, only be applicable if the new constraint store generated by the rule application is consistent. Otherwise the effects of the rule application have to be undone and another rule must tried.

We will give a formal definition of this process in Section 5 after we have made a final extension to our rewrite formalism which allows us to handle over constrained diagrams.

5 The Problem of Transient Inconsistencies: ARS_4

With the addition of interpretations we have introduced a new reason why a rule application can fail (or rather, why a rule is not applicable): even if the LHS matches correctly, the rule application can attempt to construct an inconsistent constraint store. In our current model such rules can not fire. On one hand, this behaviour is desirable, because given complete and correct geometric interpretations it guarantees that every diagram generated is geometrically feasible. For example, whereas a diagram in our first model could well have contained the relations $inside(X, Y) \wedge outside(X, Y)$ at the same time, this is not possible if both relations are interpreted. However, when we are looking at animations, there are certain cases of inconsistencies that are difficult to avoid and, in fact, do not have to be avoided. These inconsistencies occur only *temporarily* and a consistent state is automatically restored after the end of a transformation or some part of the transformation. Two main reasons exist, why such inconsistencies occur: (1) an atomic animation step, i.e., a visual transformation that is conceptually regarded as a single instantaneous transition, often has to be split into several rule applications, because the number of objects involved in not fixed. (2) The design of our rule model aims at maintaining contextual relations automatically, even when object are modified. While this is desirable, it is not always possible to automatically decide which relations have to be maintained and which have to be deleted if a diagram modification causes two contextual relations to conflict.

Let us clarify the problem by means of a simple example language, called *boxes*. The sentences of *boxes* consist of rectangles and circles. Circles are connected to a single "parent" rectangle by a line, and circles sharing the same parent may be interconnected by arrows. The desired transformation of *boxes* is to move all circles into their parent boxes while maintaining their interconnections. Thus the picture on the left hand side of Figure 3 is a sentence of *boxes* and the right hand side is the desired transformation of this sentence. Like above a naive approach to formalize the transformation consists only of a single production. Assume the vocabulary is defined as $(\{circle, rect, arrow, line\}, \{startsat : arrow \times circle, endsat : arrow \times circle, inside : O_1 \times rect, outside : O_1 \times rect, attached : line \times O_2\})$, where $O_1 = circle \cup arrow \cup line, O_2 = rect \cup circle$ together with appropriate interpretations. We might try to use the rule:

$C : circle, L : line, R : rect$ with
$attached(L, C), attached(L, R), outside(C, R)$
$\diamond\, true$
$\Rightarrow\ C : circle, L : line, R : rect$ with
$attached(L, C), attached(L, R), inside(C, R)$
$\diamond\, nil$

Since context relations are maintained automatically we do not need to worry about handling the arrows at all. Just swapping the boxes inside of their parent box, the arrows are following them automatically, because their *startsat* and *endsat* relations are maintained. However, exactly because of this automatic maintenance of contextual relations our rules generate a geometric conflict. When moving the circles one by one the first rule application generates the state given in Figure 4. Thus the picture contains the original relations $outside(A, R) \wedge endsat(A, C)$ and the new relation $inside(C, R)$ which are contradictory.

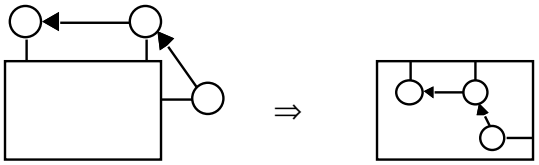

Figure 3: The Example Language *boxes*

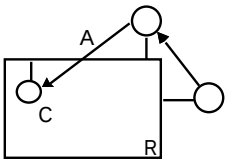

Figure 4: Intermediate State in the *boxes* Transformation

Conceptually, the intended effect is, of course, to delete the $outside(A, R)$ relation in favour of the $startsat(A, C)$ or $endsat(A, C)$ relation when some *circle* C is moved. However, it is not possible to decide automatically, which relation is preferred and should thus be maintained. These relations therefore have to be changed explicitly by the production set at a later derivation stage. However, it is still desirable that the production is applicable despite of the conflict, so that this later derivation stage can be reached in the first place.

The way to accommodate such conflict generating productions is to make the rule application mechanism detect the conflict, remove the conflicting relations from the diagram so that a consistent diagram is restored and to yield the conflicting relations that have been removed explicitly as a conflict set. This conflict set can later be processed by other rules in order to reconstruct a consistent global diagram state after a transformation phase has been finished that generates temporary conflicts. We are thus introducing a transaction-like concept into the rule mechanism: A group of productions can temporarily produce conflicts, but all conflicts have to be resolved at the end of this transaction. Of course, it is required that there is some way to check whether a conflict set is actually empty.

These ideas can be formalized by associating an explicit conflict set with each diagrams and letting the productions process these conflict sets explicitly. A production application now proceeds in the following steps: (1) The LHS of the production is matched just like before with the addition of matching the conflict set, (2) the LHS objects and relations are removed from the current diagram and all constraints attached to such removed relations are deleted from the constraint store. (3) the RHS objects and relations are added to the current diagram. (4) The interpretations of each relation in the new diagram is tested for consistency with the current constraint store. If it is consistent, it is added to the constraint store. If it is not consistent, the relation is removed from the current diagram and added to the conflict set. (5) The conflict set is updated according to the RHS.

Here a careful analysis of when exactly inconsistencies can occur has to be given. The basic idea is to always maintain a diagram with a consistent geometric interpretation plus an explicitly given set of conflict relations. Removing objects or relations from such a diagram can not cause new conflicts, since it only relaxes constraints in the diagram. We require that the isolated set of RHS objects and relations does not contain any conflicts, i.e., that the RHS of a production which represents some new subpicture is consistent in itself as long as it is not combined with another diagram. This restriction is sensible, since it only means that no production can explicitly add conflicts to the diagram. Conflicts can therefore only arise when relations are interpreted of which one argument is a RHS object and the other argument is an object of the current diagram. These relations are exactly what we have termed contextual relations before. Consequently, if a contextual relation conflicts with some relation in the current diagram or a RHS relation, the contextual relation will be moved to the conflict set. Still, the idea of a conflict has to be made more precise. What happens if a conflict arises between two relations, both of which are contextual? This was exactly the case in our example of the *boxes* transformation where a conflict between the two contextual relations $startsat(A, C)$ and $outside(A, R)$ arises in the intermediate state. The intuitive idea of how to resolve such a situation is that we have an order of relative importance for the relations in mind. In our *boxes* language, for example, *startsat* and *endsat* are the relations for *arrow*s that we are primarily interested in and the *inside* and *outside* relations for *arrow*s have to follow the *circle*s to which they are attached. To formalize this, it is sufficient to define a partial order $<$ over the relation types in the vocabulary such that r_1 is preferred over r_2 if $r_1 < r_2$. If two contextual relations r_1 and r_2 conflict, the preferred relation will be kept in the diagram and the less preferred one will be moved to the conflict set. For the purpose of our example it is then sufficient to define $startsat, endsat < inside, outside$.

We summarize all of the above extensions to the basic formalism in the subsequent definitions:

Definition 4 *A picture vocabulary is a quadruple $V = (OT, RT, I, <)$ of a set OT of object types and a set RT of spatial relation types, a set I of interpretations and a partial order $<$ for the elements of RT. Each element $f = I_R \in I$ is an interpretation attached to some relation type $R \in RT$ given as a constraint formula f over the attributes of the object types involved in R.*

A diagram over V is a quadruple $D = (O, R, CF, S)$ of a set O of objects, a set R of relations with types according to V, a conflict set CF and a constraint store S. The elements of CF are relations according to V. A diagram $D = (O, R, \emptyset, S)$ is called consistent *if S is consistent.*

Definition 5 *An ARS_4 production system is a tuple (V, P) of a picture vocabulary V and a set P of production rules of the form*

$O_1, \ldots, O_n$ with
$R_1, \ldots, R_m$
$\Downarrow\ CF_1, \ldots, CF_k$
$\Diamond\, C_{before}$
$\Rightarrow\ O'_1, \ldots, O'_{n'}$ with
$R'_1, \ldots, R'_{m'}$
$\Downarrow\ CF_{l_1}, \ldots, CF_{l_k}$
$\Diamond\, C_{after} \Diamond\, \Theta$

where $O_1, \ldots, O_n, O'_1, \ldots, O'_{n'}$ *are typed variables over* OT*;* $R_1, \ldots, R_m, R'_1, \ldots, R'_{m'}$ *and* $CF_1, \ldots, CF_k$ *are typed relations over* RT*;* C_{before} *is a boolean expression over the attributes of* $O_1, \ldots, O_n$ *and* C_{after} *is an assignment to the attributes of* $O'_1, \ldots, O'_{n'}$*.* $\{CF_{l_1}, \ldots, CF_{l_k}\} \subseteq \{CF_{l_1}, \ldots, CF_{l_k}\}$*.* Θ *is a sequence of substitutions* $\{X/Y\}$ *where* X *is a LHS object variable and* Y *is a RHS object variable.*

Definition 6 *A production* $P \equiv\ X$ with $Y\ \Downarrow\ CO \Diamond C\ \Rightarrow\ X'$ with $Y'\ \Downarrow\ CO' \Diamond C' \Diamond \Theta$ *of an* ARS_4 *production system can be applied to a diagram* $D = (O, R, CF, S)$ *if there is a substitution* θ *such that* $O = X\theta \cup A$*,* $R = Y\theta \cup B$*,* $CF = CO\theta \cup D$ *and* $C\theta$ *evaluates to true. The application of* P *to* D *generates the diagram* $D' = (O', R', CF', S')$ *with* $O' = A \cup X'\theta$*,* $R' = B\Theta \cup Y'\theta - \Delta$*,* $CF' = D \cup CO'\theta \cup \Delta$ *in which the attributes of objects in* O' *are modified according to* $C'\theta$*.*

Let $S_0 = S - \{I_r \mid r \in Y\theta\} + \{I_r \mid r \in Y'\theta\}$*.* Δ *is determined by testing the interpretation* I_r *of each contextual relation* r *in* D' *in the order defined by* $<$ *for consistency with* S_0*. If* $S_0 \wedge r \models false$ *then* $\Delta = \Delta \cup \{r\}$ *otherwise* $S_0 = S_0 \cup \{r\}$*.* $S' = S_0 - \{I_r \mid r \in \Delta\}$*.*

With these extensions it is easy to define a production set that specifies the desired transformation of *boxes*:

$C : circle, L : line, R : rect$ with
$attached(L, C), attached(L, R), outside(C, R)$
$\Downarrow\ \emptyset$
$\Diamond\, true$
$\Rightarrow\ C : circle, L : line, R : rect$ with
$attached(L, C), attached(L, R), inside(C, R)$
$\Downarrow\ \emptyset$
$\Diamond\, nil \Diamond\, \{\}$
$C_1, C_2 : circle, A : arrow, R : rect$ with
$startsat(A, C_1), endsat(A, C_2), inside(C_1, R), inside(C_2, R)$
$\Downarrow\ outside(A, R)$
$\Diamond\, true$
$\Rightarrow\ C_1, C_2 : circle, A : arrow, R : rect$ with
$startsat(A, C_1), endsat(A, C_2), inside(C_1, R), inside(C_2, R), inside(A, R)$
$\Downarrow\ \emptyset$
$\Diamond\, nil \Diamond\, \{\}$

Repeated application of the first rule moves all circles into their corresponding parent while maintaining the attachment to arrows and accumulates a conflict set of all the $outside(A, R)$ relations for arrows whose attached circles have been moved into some rectangle R. After this rule has exhaustively been applied, the second rule explicitly manages the accumulated conflict set by changing all such conflicting *outside* relations into the corresponding *inside* relations, leaving an empty conflict set after it has exhaustively been applied. The entire transformation thus restores a conflict-free diagram with a consistent geometric interpretation.

With the three extensions of object replacement, constraint solving and temporary conflict resolution that we have made to the basic model, we are now able to define a broad range of diagram animations by ARS_4 rewrite specifications.

6 Towards an Implementation

So far the discussion has been focused on the theoretical requirements for formal animation specifications. A rather important question is whether we could build a usable system based on this formal model. Determining applicability of a rewrite production is simple and not different from other production systems. The real issue is whether it is possible to perform constraint solving quickly enough and to perform diagram layout quickly. This of course depends on the type of constraints, the required operations of the constraint solver, and the type of aesthetic criteria. In particular the dynamic deletion of constraints which are required for ARS_4 execution is a critical issue. Fortunately there has been recent work on contraint based graphics and graph layout which we can leverage from.

The constraint solver QOCA has been developed for exactly this type of problem. It handles linear arithmetic equality and inequality constraints. One of its most important features is that it allows incremental addition and deletion of constraints. This is exactly what is required when we perform rewriting. Its other important feature is that it provides convex quadratic optimization. This means that it can be used to find solutions to the constraints which minimize a sum of least squares formula. This can be used to find the solution which is as close as possible to the layout before the transformation was applied, thus ensuring small changes and smooth transitions between subsequent variations of the same diagram. The same functionality can also be used to model simple aesthetic criteria like "parents of children in a tree shold be as close as possible to the center". Earlier versions of QOCA, which were used to support interactive graphical editing [HHMV95], have already proven that the "minimum surprise" rule for interactive graphics can successfully be supported in this way.

Rather surprisingly, QOCA is fast. With systems of up to 1000 constraints, addition, deletion and optimization take significantly less than one second [BMSX97].

The model for constraint based diagram layout is based on the constraint based graph layout model introduced in [HM86].

7 Discussion and Conclusions

We have presented a rewrite formalism for the specification of diagram animations. Our analysis of the basic task has shown that a combination of constraint solving and symbolic relation rewriting is required to support non-trivial diagram animations. Such a system has been described and the basic methods for its implementation have been discussed.

The formalism presented can be considered as an extension to constraint-based multidimensional grammars. However, for the purpose of animation it was required to introduce a transaction concept which allows to handle transient diagram states that temporarily do not have a feasible geometric interpretation. This can not be achieved with normal constraint-based grammar models.

Other recent work on the formal specification of diagram animation includes [GC96, UD95, UD97, KK91, TMMY94, Fur90, Fur91, Fur92, Bel92, BL93]. In [GC96] a specification of the execution of the visual programming language Pictorial Janus is given. However, the approach is only capable of specifying each single transitions between two diagram states seperately. Since the formal model does not include any notion of state or change, it does not seem possible to entirely base a complete specification of diagram execution or animation on it. The approach to dynamic diagram specification presented in [UD95, UD97] is based on rewrite specifications, too, but is directly targeted for interactive languages, for example the specification of diagram editors.

Other work on the specification of diagram animation [KK91, TMMY94, Fur90, Fur91, Fur92, Bel92, BL93] considers the execution or animation of diagrams as a separate problem from their syntactic and semantic specification. In contrast we are aiming at integrating both aspects of diagram specification by extending well-known and proven methods for the visual syntax specification to the realm of dynamic diagram languages with the objective to formalize the syntax, semantics, and execution of visual languages within a single formalism. Related work which covers the interpretation and execution of visual mathematical expressions based on the integration of the presented model into a logic programming framework can be found in [Mey97].

References

[Bel92] B. Bell. Chemtrains: A rule-based visual language for building graphical simulations. Technical Report CU-CS-529-92, US West Advanced Technologies, February 1992 1992.

[BL93] B. Bell and C. Lewis. Chemtrains: A language for creating behaving pictures. In *IEEE Workshop on Visual Languages*, pages 188–195, Bergen, 1993. IEEE Computer Society Press.

[BMSX97] Alan Borning, Kim Marriott, Peter Stuckey, and Yi Xiao. Solving linear arithmetic constraints for user interface applications. Submitted to the *1997 ACM Conference on User Interface Software and Technology*. Available from http://www.cs.washington.edu/research/constraints, April 1997.

[Bri88] William M. Bricken. An introduction to boundary logic with the losp deductive engine. Research report, University of Washington, 1988.

[CDZ94] W. Citrin, M. Doherty, and B. Zorn. Formal semantics of control in a completely visual programming language. In *IEEE Symposium on Visual Languages*, St. Louis/MO, 1994.

[CHZ95] Wayne Citrin, Richard Hall, and Benjamin Zorn. Programming with visual expressions. In *IEEE Workshop on Visual Languages*, pages 294 – 301, Darmstadt, Germany, 1995. IEEE Computer Society Press.

[Fur90] G.W. Furnas. Formal models for imaginal deduction. In *Twelfth Annual Conference of the Cognitive Science Society*, pages 662–669, Cambridge/MA, 1990. Lawrence Erlbaum.

[Fur91] G.W. Furnas. New graphical reasoning models for understanding graphical interfaces. In *Human Factors in Computing Systems ACM CHI 91*, pages 71–78, New Orleans, 1991.

[Fur92] G.W. Furnas. Reasoning with diagrams only. In *AAAI Spring Symposium on Reasoning with Diagrammatic Representations*, Stanford/CA, 1992.

[GC96] J.M. Gooday and A.G. Cohn. Using spatial logic to describe visual programming languages. *Artificial Intelligence Review*, 10:171 – 186, 1996.

[GNC95] J. Glasgow, N.H. Narayanan, and B. Chandrasekaran. *Diagrammatic Reasoning: Cognitive and Computational Perspectives*. AAAI Press and MIT Press, Menlo Park/CA and Cambrdige/MA, 1995.

[Ham93] Eric Hammer. Representing relations diagrammatically. In Gerard Allwein and Jon Barwise, editors, *Working Papers on Diagrams and Logic*. Indiana University, Bloomington, 1993.

[Har88] David Harel. On visual formalisms. *Communications of the ACM*, 31(5):514 – 530, 1988.

[HD93] Eric Hammer and Norman Danner. Towards a model theory of diagrams. In Gerard Allwein and Jon Barwise, editors, *Working Papers on Diagrams and Logic*. Indiana University, Bloomington, 1993.

[HHMV95] Richard Helm, Tien Huynh, Kim Marriott, and John Vlissides. An object-oriented architecture for constraint-based graphical editing. In *Object-Oriented Programming for Graphics*. Springer-Verlag, New York, 1995.

[HM86] W. He and K. Marriott. Constrained graph layout. In *Graph Drawing '96*, pages 217–232. Springer-Verlag (LNCS 1190), 1986.

[Kee95] David Keenan. To dissect a mockingbird: A graphical notation for the lambda calculus with animated reduction. Technical report, Smalltalk Computing, 1995.

[KK91] T. Kamada and S. Kawai. A general framework for visualizing abstract objects and relations. *ACM Transactions on Graphics*, 10(1), 1991.

[KS90] Kenneth M. Kahn and Vijay A. Sraswat. Complete visualizations of concurrent programs and their execution. In *IEEE Workshop on Visual Languages*, pages 7 – 15, Skokie/IL, 1990. IEEE Computer Society Press.

[Mey97] B. Meyer. Formalization of visual mathematical notations. In *AAAI Symposium on Diagrammatic Reasoning*, Boston/MA, forthcoming 1997. AAAI Press.

[Shi95] Sun-Joo Shin. *The Logical Status of Diagrams*. Cambridge University Press, Cambridge/MA, 1995.

[TMMY94] Shin Takahashi, Ken Miyashita, Satoshi Matsuoka, and Akinori Yonezawa. A framework for constructing animations via declarative mapping rules. In *IEEE Symposium on Visual Languages*, pages 314 – 322, St. Louis/MI, 1994.

[UD95] S. Üsküdarli and T.B. Dinesh. Towards a visual programming environment generator for algebraic specifications. In *IEEE Symposium on Visual Languages*, Darmstadt, Germany, 1995.

[UD97] S. Üsküdarli and T.B. Dinesh. Input and output for specified visual languages. In K. Marriott and B. Meyer, editors, *Theory of Visual Languages*. Springer, New York, forthcoming 1997.