ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Περιβάλλον σχεδίασης διαδραστικών κινούμενων κόσμων (χωροχρονικών συνθέσεων)»

Τεντζεράκη Μαρία
Μ3980003

ΑΘΗΝΑ, ΦΕΒΡΟΥΑΡΙΟΣ 2000

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Περιβάλλον σχεδίασης διαδραστικών κινούμενων κόσμων (χωροχρονικών συνθέσεων) »

Τεντζεράκη Μαρία
Μ3980003

Επιβλέπων Καθηγητής: Μ. Βαζιργιάννης
Εξωτερικός Κριτής: Καθηγητής Γ. Πολύζος

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΘΗΝΑ, ΦΕΒΡΟΥΑΡΙΟΣ 2000

Πρόλογος

Η παρούσα διπλωματική εργασία πραγματοποιήθηκε στα πλαίσια του Μεταπτυχιακού Προγράμματος Σπουδών στα Πληροφοριακά Συστήματα του Οικονομικού Πανεπιστημίου Αθηνών. Το θέμα που διαπραγματεύεται προτάθηκε σε συνεργασία με τα ερευνητικά ενδιαφέροντα τόσο του Οικονομικού Πανεπιστημίου, όσο και του Γαλλικού Πανεπιστημίου CNAM (Conservatoire Nationale des Arts et Metiers.) στα πλαίσια του προγράμματος ανταλλαγής φοιτητών Socrates/Erąsmus. Το γεγονός αυτό έκανε επιτακτική την ανάγκη, η εργασία να αναπτυχθεί στην αγγλική γλώσσα έτσι ώστε να υφίσταται μία κοινή βάση επικοινωνίας, συζήτησης και κριτικής. Στην ελληνική γλώσσα ωστόσο παραθέτουμε το executive summary και μία εκτενής περίληψη των κεφαλαίων που ακολουθούν, ελπίζοντας να διευκολύνεται έτσι το έργο του αναγνώστη.

Θα ήθελα προσωπικά να εκφράσω τις ευχαριστίες μου για τη φιλοξενία του γαλλικού πανεπιστημίου και τη βοήθεια των υπεύθυνων καθηγητών στην όσο το δυνατόν άνετη και χωρίς προβλήματα διαμονή μου στο Γαλλικό κράτος. Θα ήθελα τέλος να ευχαριστήσω τους επιβλέποντες καθηγητές μου και τον εξωτερικό κριτή για τις πολύτιμες παρατηρήσεις στη δομή, το περιεχόμενο και την παρουσίαση της διπλωματικής εργασίας.

Τεντζεράκη Μαρία
Φεβρουάριος 2000

Περιεχόμενα

Executive Summary

Στο executive summary θα προσπαθήσουμε να δώσουμε μία συνολική εικόνα των αποτελεσμάτων της εργασίας. Η εργασία αποτελεί το πρώτο βήμα για την υλοποίηση ενός ολοκληρωμένου συστήματος σχεδίασης, αποθήκευσης και ανάκτησης εγγράφων πολυμέσων που σαν βασικό τους χαρακτηριστικό έχουν την κίνηση (animation) και την αλληλεπίδραση (interaction) (Interactive Animations, IANs).

Οι γενικές απαιτήσεις από ένα τέτοιο σύστημα συνοψίζονται στα ακόλουθα:

- Θα πρέπει να παρέχεται στο χρήστη η δυνατότητα περιγραφής ενός Interactive Animation. Πιο συγκεκριμένα ο χρήστης θα πρέπει να μπορεί να ορίζει αντικείμενα όπως μία μπάλλα ή ένα αυτοκίνητο, να περιγράφει τους κανόνες με τους οποίους αυτά κινούνται, και να ορίζει τα γεγονότα που επηρεάζουν τα γραφικά χαρακτηριστικά και τη συμπεριφορά τους.
- Θα πρέπει ο χρήστης να έχει τη δυνατότητα να εκτελεί αυτές τις παρουσιάσεις και να βλέπει το οπτικό αποτέλεσμα της περιγραφής που έχει δώσει.
- Θα πρέπει ο χρήστης να έχει τη δυνατότητα αποθήκευσης τους σε βάση δεδομένων και τέλος
- Θα πρέπει ο χρήστης να έχει τη δυνατότητα επερωτήσεων (queries) για την ανάκτηση των παρουσιάσεων και για την άντληση ποικίλων πληροφοριών και κυρίως πληροφοριών που σχετίζονται με τη θέση ή την κατάσταση των αντικειμένων κατά τη χρονική διάρκεια των παρουσιάσεων (χωροχρονικές ερωτήσεις).

Για την ικανοποίηση των παραπάνω απαιτήσεων στα πλαίσια της εργασίας προτείνουμε μία νέα γλώσσα δημιουργίας Interactive Animation. Κατόπιν υλοποιούμε ένα μέρος των δυνατοτήτων που πρέπει να παρέχει αυτή η γλώσσα, δίνοντας στο χρήστη τη δυνατότητα να δημιουργήσει παρουσιάσεις και να τις εκτελέσει στην οθόνη του υπολογιστή του. Στα πλαίσια της εργασίας σχεδιάζουμε επίσης το σχήμα της βάσης δεδομένων έτσι ώστε ο χρήστης να έχει τη δυνατότητα να αποθηκεύσει τις παρουσιάσεις που δημιούργησε και τέλος συντάσσουμε τις εντολές αναζήτησης για την ανάκτηση πληροφοριών από τις παρουσιάσεις που αποθηκεύονται στη βάση. Στο σημείο αυτό και για την καλύτερη κατανόηση της σημασίας του συστήματος θα δώσουμε ένα πλήρες παράδειγμα της λειτουργίας του.

Ας υποθέσουμε ότι ο χρήστης θέλει να δημιουργήσει 2 αντικείμενα: ένα τετράγωνο χρώματος κίτρινου και συγκεκριμένων διαστάσεων έστω 1, 1 και μία έλλειψη χρώματος κόκκινου και διάστάσεων 2, 2. Θέλει επίσης να ορίσει τις αρχικές τους θέσεις στα σημεία 3, 4 και 7, 8 αντίστοιχα, ενώ η κίνησή τους προσδιορίζεται με βάση τους κανόνες $2t$, και $5t + 1$ για τον άξονα των x αντίστοιχα και $4t + 7$ και $5t + 5$ για τον άξονα των y.

Με βάση τη γλώσσα που δημιουργήσαμε (και που τη γραμματική της μπορεί κανείς να μελετήσει στο κεφάλαιο 3 της εργασίας) προκειμένου να περιγράψουμε την παραπάνω παρουσίαση θα πρέπει να δώσουμε την ακόλουθη περιγραφή:

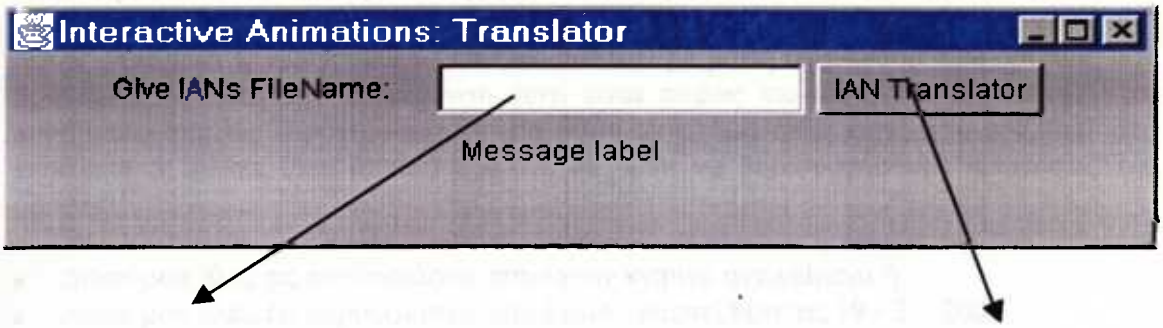
```
OBJECT SampleRectangle
{
   FORM
   {
      TYPE RECTANGLE
      POINTS (1,1)
   }
   COLOUR 1 1 0
}

OBJECT SampleEllipsis
{
   FORM
   {
      TYPE ELLIPSE
      POINTS (2, 2)
   }
   COLOUR 1 0 0
}

EVENT
  START
ACTIONS
  SEQ
   ( SampleRectangle.START((3, 4), TRUE ) 0
     SampleEllipsis.START((7, 8), TRUE ) 0
     SampleRectangle.C_TRANSLATION(2 TIME + 0, 4 TIME + 7) 0
     SampleEllipsis.C_TRANSLATION(5 TIME + 1, 5 TIME + 5) 0)
```

Την περιγραφή αφού την αποθηκεύσουμε σε αρχείο ASCII (έστω στο C:\twoObjects.txt) τη μεταφράζουμε με την χρήση του εργαλείου μας και το αποτέλεσμα της μετάφρασης μπορούμε να το δούμε σε οποιαδήποτε browser που υποστηρίζει VRML (το τελικό προιόν της μετάφρασης είναι ένα αρχείο VRML).

Το εργαλείο παρέχει ένα πολύ απλό interface. Αρκεί να δώσουμε ως παράμετρο το μονοπάτι του αρχείου με την περιγραφή της παρουσίασης στη γλώσσα που εμείς έχουμε ορίσει (script file) και να ενεργοποιήσουμε κατόπιν τον μεταφραστή μέσα από το κατάλληλο πλήκτρο. Η διεπαφή που χρησιμοποιούμε φαίνεται στο σχήμα που ακολουθεί:

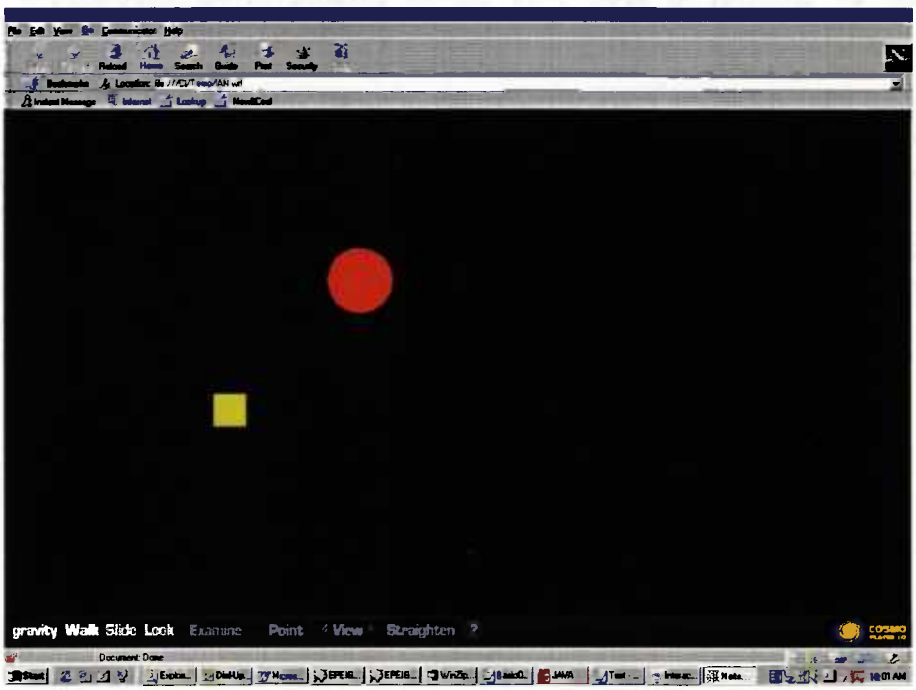


Στο σημείο αυτό θα πρέπει να δώσουμε ως παράμετρο το όνομα του αρχείου

Αυτό είναι το πλήκτρο ενεργοποίησης της μετάφρασης

Σχήμα 1: Η κεντρική διεπαφή του συστήματος μετάφρασης

Η ενεργοποίηση του εργαλείου γίνεται δίνοντας στη γραμμή εντολών του dos την εντολή: `java Translator`. Το αρχείο VRML που δημιουργείται παίρνει το προκαθορισμένο όνομα IAN.wrl και τοποθετείται αυτόματα στον κατάλογο C:\temp. Ένα στιγμιότυπο του σεναρίου, στην αρχική έστω χρονική στιγμή παρουσιάζεται στην οθόνη που ακολουθεί:



Σχήμα 2: Αποτέλεσμα μετάφρασης

Ας υποθέσουμε τώρα ότι ο χρήστης έχει δημιουργήσει με τη χρήση του εργαλείου ένα μεγάλο αριθμό παρόμοιων παρουσιάσεων και θέλει να αντλήσει συγκεντρωτικά στοιχεία από αυτές. Έτσι προκύπτει η ανάγκη για αποθήκευση των παρουσιάσεων στη βάση.

Οπωσδήποτε το να αποθηκεύσουμε τον παραγόμενο κώδικα σε VRML δεν ωφελεί ιδιαίτερα, καθώς δεν είναι τόσο εύκολο να αντλήσουμε τις απαντήσεις μας μέσα από αυτόν. Αντίθετα η περιγραφή των παρουσιάσεων με βάση τη γλώσσα που ορίσαμε μας βοηθάει προς αυτή την κατεύθυνση γιατί είναι σαφώς πιο απλή και δομημένη. Έτσι αποθηκεύοντας τις παρουσιάσεις σε μία βάση με σχήμα όπως αυτό που ορίζουμε στο κεφάλαιο 5 (βλέπε data model) είμαστε σε θέση να δημιουργήσουμε ερωτήσεις της μορφής:

- Δώσε μου όλες τις παρουσιάσεις που έχουν κίτρινα αντικείμενα ή
- Δώσε μου όλες τις παρουσιάσεις που έχουν γραφτεί πριν τις 19 - 2 – 2000.

Ο χρήστης ωστόσο θέλει να αντλήσει από τις παρουσιάσεις του περισσότερο εξειδικευμένες πληροφορίες όπως:

- Δώσε μου τη θέση που είχε το αντικείμενο α στην παρουσίαση x κατά τη χρονική στιγμή 4 ή
- Δώσε μου τη χρονική στιγμή κατά την οποία δύο αντικείμενα σε μία παρουσίαση συναντώνται ενώ ταυτόχρονα έχουν κόκκινο χρώμα.

Αυτού του είδους οι χωροχρονικές ερωτήσεις παρουσιάζουν κάποιες ιδιαιτερότητες και απαιτούν την παραπέρα μελέτη γύρω από αυτό το χώρο. Στο πέμπτο κεφάλαιο της εργασίας προτείνουμε συγκεκριμένες λύσεις έτσι ώστε ο χρήστης να μπορεί να δημιουργήσει τέτοιου είδους ερωτήσεις.

Ο ρόλος και η χρησιμότητα ενός τέτοιου συστήματος είναι προφανής εάν θεωρήσουμε ότι αυτές οι παρουσιάσεις μπορεί να αφορούν τις τροχιές αεροπλάνων ή πλοίων κλπ. Η προσπάθεια αυτή είναι πραγματικά καινοτομική καθώς δεν έχει επιχειρηθεί στο παρελθόν η δημιουργία γλώσσας για Interactive Animations ενώ παράλληλα τα υπόλοιπα μοντέλα που έχουν προταθεί στο χώρο αυτό δεν έχουν τον προσανατολισμό που εμείς επιθυμούμε για να ολοκληρώσουμε την προσπάθειά μας μέσα από την σύνδεση με τον χώρο των βάσεων δεδομένων.

Σύντομη παρουσίαση

Μετά το executive summary κρίνουμε σκόπιμο να παρουσιάσουμε μία εκτενή περίληψη των κεφαλαίων της εργασίας στα ελληνικά έτσι ώστε να βοηθήσουμε τον αναγνώστη να κατανοήσει καλύτερα το κείμενο που ακολουθεί. Παραθέτουμε επίσης μία σύντομη περίληψη στα αγγλικά.

Σκοπός και στόχοι της εργασίας

Η εργασία εντάσσεται στα πλαίσια δύο διαφορετικών επιστημονικών περιοχών, των πολυμέσων και των βάσεων δεδομένων. Σκοπός της εργασίας είναι να διερευνήσει το κατά πόσο οι δύο αυτοί διαφορετικοί τομείς είναι δυνατόν να συσχετιστούν, επικεντρώνοντας στις παρουσιάσεις πολυμέσων (multimedia presentations) με βασικά τους χαρακτηριστικά, την κίνηση (animation) των αντικειμένων που συμμετέχουν σ' αυτές και την αλληλεπίδραση, τόσο μεταξύ τους, όσο και με τον χρήστη (interaction). Εξαιτίας αυτών των δύο βασικών γνωρισμάτων ονομάζουμε τις παρουσιάσεις που θα αποτελέσουν επίκεντρο της έρευνάς μας ως Interactive Animations, ή εν συντομία IANs

Για το συσχετισμό των δύο αυτών περιοχών οριοθετούμε τρεις επιμέρους βασικούς στόχους:

- Τη δημιουργία γλώσσας για την περιγραφή Interactive Animations. Για το σκοπό αυτό προχωρούμε στο σχεδιασμό και την παραγωγή της κατάλληλης γραμματικής
- Την υλοποίηση των βασικών χαρακτηριστικών της γλώσσας, έτσι ώστε ο χρήστης να έχει ένα πραγματικό εργαλείο δημιουργίας τέτοιων παρουσιάσεων (authoring tool ή Translator) και τέλος
- Τη σχεδίαση ερωτήσεων (queries) πάνω στις παρουσιάσεις (IAN) που παράγονται από τα προηγούμενα βήματα.

Για την υλοποίηση των στόχων βασιστήκαμε στο θεωρητικό μοντέλο περιγραφής χωροχρονικών συνθέσεων που προτάθηκε από τους Dan Vodislav και Μιχάλη Βαζιργιάννη στο άρθρο τους με τίτλο "Generic Object Modelling for Animation Contexts", καθώς είναι το μόνο που βοηθάει στη σχεδίαση γλώσσας για τη δημιουργία Interactive Animations ενώ παράλληλα έχει σαφή προσανατολισμό στη σύνδεση των παρουσιάσεων με τις βάσεις δεδομένων

Βασικές έννοιες – συνοπτική παρουσίαση του μοντέλου.

Πριν εξηγήσουμε τον τρόπο με τον οποίον θα αναπτύξουμε το σύστημά μας για την ικανοποίηση των παραπάνω στόχων κρίνουμε σκόπιμο να αναλύσουμε ορισμένα από τα βασικά χαρακτηριστικά του μοντέλου στο οποίο βασιζόμαστε.

Όπως έχουμε ήδη αναφέρει η κίνηση αποτελεί το χαρακτηριστικό που διαφοροποιεί τα IANs από τις ποικίλες παρουσιάσεις πολυμέσων. Η κίνηση (animation) κατά συνέπεια είναι και το βασικό συστατικό στοιχείο του μοντέλου. Έτσι οι παρουσιάσεις που παράγονται αποτελούνται από κινούμενα αντικείμενα (animated objects) που

αλληλεπιδρούν είτε μεταξύ τους είτε με τον ίδιο το χρήστη μέσα από το πληκτρολόγιο, το ποντίκι κλπ.

Τα κινούμενα αντικείμενα μπορεί να είναι είτε απλά είτε σύνθετα. Τα απλά αντικείμενα προσδιορίζονται από τη γεωμετρία τους (τετράγωνο, γραμμή, έλλειψη, πολύγωνο), τη θέση τους στο χώρο, το χρώμα τους και γενικότερα τα υπόλοιπα γραφικά τους χαρακτηριστικά, τους νόμους και τα είδη της συνεχούς κίνησής τους (translation, rotation, scaling) τον εσωτερικό τους χρόνο (πότε εμφανίζονται στην παρουσίαση, πότε σταματούν να είναι ενεργά κλπ.), την ορατότητά τους (visibility) και την κατάστασή τους αν δηλαδή είναι ενεργά ή αν η κίνησή τους έχει τερμάτιστεί ή αν έχει σταματήσει προσωρινά.

Κατά συνέπεια η γραμματική της γλώσσας που σκοπεύουμε να δημιουργήσουμε θα πρέπει να οριστεί με τρόπο ώστε να παρέχεται στο χρήστη η δυνατότητα να ορίζει τέτοιου είδους απλά αντικείμενα. Επιπλέον θα πρέπει να δίνεται η δυνατότητα ορισμού και σύνθετων αντικειμένων. Ένα σύνθετο αντικείμενο περιγράφεται με τα ίδια ακριβώς χαρακτηριστικά όπως και το απλό αλλά περιλαμβάνει επιπλέον μία λίστα με τα απλά αντικείμενα που το συνθέτουν.

Το μοντέλο τέλος κάνει αναφορά στη συμπεριφορά αυτών των αντικειμένων και προσδιορίζει τις βασικές τους λειτουργίες: start για την αρχικοποίηση των αντικειμένων στη θέση που ορίζει ο χρήστης, pause resume και stop για τον έλεγχο της κατάστασης των αντικειμένων, translation, rotation, scaling, color για την κίνηση και τη μεταβολή των γεωμετρικών ή γραφικών χαρακτηριστικών των αντικειμένων κλπ.

Scenario modeling

Σαν επόμενο βήμα πριν τη δημιουργία της κατάλληλης γραμματικής είναι η επιλογή του τρόπου με τον οποίον οι χρήστες θα μπορούν να περιγράφουν όλες τις παραπάνω έννοιες και να τις συνδέουν μεταξύ τους για την παραγωγή παρουσιάσεων. Με λίγα λόγια θα πρέπει να αποφασίσουμε στη δομή της αφηρημένης περιγραφής που θα δίνει ο χρήστης. Γι' αυτό το σκοπό εισάγουμε την έννοια του σεναρίου. Το σενάριο βοηθάει τον χρήστη να περιγράψει τη συμπεριφορά των αντικειμένων που συμμετέχουν στην παρουσίαση και αποτελείται από επιμέρους εντολές (statements). Κάθε εντολή του σεναρίου έχει μία συγκεκριμένη δομή και αποτελείται από γεγονότα (events), συνθήκες (conditions) και διαδικασίες ή δράσεις ή πράξεις (actions). Με τη χρήση των γεγονότων και των συνθηκών μπορούμε πλέον να χειριστούμε αποτελεσματικά και την αλληλεπίδραση (interaction) που όπως προαναφέραμε αποτελεί το δεύτερο βασικό χαρακτηριστικό των παρουσιάσεων που διαπραγματευόμαστε.

Έτσι ένας χρήστης για να περιγράψει πλήρως μία παρουσίαση αρκεί να δώσει μία αφαιρετική και δομημένη περιγραφή των αντικειμένων (objects) που συμμετέχουν σ' αυτήν, των γεγονότων που συμβαίνουν, των συνθηκών που ισχύουν προκειμένου να ενεργοποιηθούν τα γεγονότα και τέλος των δράσεων, της συμπεριφοράς δηλαδή των αντικειμένων όταν κάποιο γεγονός συμβαίνει.

Γραμματική

Έχοντας μελετήσει σε βάθος τις έννοιες του μοντέλου και του σεναρίου, η παραγωγή της γραμματικής είναι το επόμενο βήμα. Στη γραμματική ορίζουμε ένα IAN ως το σύνολο των ορισμών των αντικειμένων που συμμετέχουν σ' αυτό και του σεναρίου που ορίζει τη συμπεριφορά τους.

```
IAN ::= objects scenario;
```

Η περιγραφή των αντικειμένων είναι ιδιαίτερα απλή και δομημένη, το ίδιο και η περιγραφή των γεγονότων των συνθηκών και των δράσεων.

Τα γεγονότα τα χωρίζουμε σε απλά και σύνθετα. Στα απλά γεγονότα ορίζουμε τα ακόλουθα:

- Γεγονότα που προκύπτουν από την αλληλεπίδραση με τον χρήστη (user interaction events)
- Γεγονότα που προκύπτουν από αλλαγή των χαρακτηριστικών ενός αντικειμένου όπως για παράδειγμα το χρώμα (intra object events)
- Γεγονότα που προκύπτουν από την αλληλεπίδραση μεταξύ των αντικειμένων όπως η συνάντησή τους (inter object events)
- Γεγονότα που εγείρονται με την πάροδο του χρόνου, π.χ γεγονότα που συμβαίνουν στο τρίτο λεπτό της παρουσίασης
- Τα σύνθετα γεγονότα που ορίζονται από τη γραμματική αποτελούνται από διάφορους συνδυασμούς των παραπάνω απλών γεγονότων

Με αντίστοιχο τρόπο η γραμματική ορίζει επίσης τις συνθήκες (conditions) που χωρίζονται σε απλές και σύνθετες. Οι απλές συνθήκες χωρίζονται σε:

- Συνθήκες κατάστασης (state conditions) όπου ελέγχεται αν τα χαρακτηριστικά γνωρίσματα ενός αντικειμένου είναι μεγαλύτερα, μικρότερα ή ίσα με μία τιμή
- Συνθήκες αλληλεπίδρασης αντικειμένων όπου ελέγχεται κατά πόσο δύο αντικείμενα ικανοποιούν ορισμένες χωρικές συνθήκες και τέλος
- Συνθήκες χρονικές όπου ελέγχεται η τιμή του χρόνου.

Οι σύνθετες συνθήκες αποτελούνται από τον συνδυασμό των απλών συνθηκών με τη χρήση λογικών τελεστών δηλαδή του OR και του AND.

Το τελευταίο μέρος των εντολών κάθε σεναρίου αποτελείται από τις δράσεις ή πραξεις (actions) που ουσιαστικά ορίζουν τη συμπεριφορά των αντικειμένων. Οι πράξεις που ορίζονται στη γραμματική αντικατοπτρίζουν αυτές που περιγράφονται στο μοντέλο όπως start, stop, hide, show, continuous transformation, continuous rotation κλπ και συνδυάζονται μεταξύ τους ως ακολουθία πράξεων με διαφορά κάποιου χρονικού διαστήματος.

Υλοποίηση του συστήματος

Ο επόμενος στόχος της εργασίας είναι η παραγωγή εργαλείου που να μπορεί να δέχεται την αφαιρετική και δομημένη περιγραφή μίας παρουσίασης από τον χρήστη και να παράγει μία εκτελέσιμη μορφή της περιγραφής αυτής.

Το εργαλείο αποτελείται από τρεις επιμέρους ενότητες, τη διεπαφή (user interface), το λεκτικό (lexical analyser) και τον συντακτικό αναλυτή (parser). Η είσοδος (input) σε αυτό το σύστημα είναι η περιγραφή της παρουσίασης από τον χρήστη σύμφωνα με τη γραμματική, σε ένα απλό αρχείο ASCII.

Το αποτέλεσμα (output) της επεξεργασίας είναι ένα VRML αρχείο που μπορεί να εκτελεστεί σε οποιονδήποτε browser υποστηρίζει αυτό το πρότυπο. Η VRML (virtual reality modelling language) κρίθηκε ως το πλέον κατάλληλο πρότυπο για τη μετάφραση της περιγραφής του χρήστη σε εκτελέσιμο format λόγω των δυνατοτήτων που παρέχει. Το αρχείο που δημιουργείται από το σύστημα έχει τον προκαθορισμένο τίτλο IAN και κατάληξη wrl, (IAN.wrl) και τοποθετείται στον κατάλογο temp του υπολογιστή.

Η διεπαφή σε αυτή την έκδοση του συστήματος δεν παρέχει σημαντικές ευκολίες στο χρήστη όπως για παράδειγμα τη δυνατότητα δήλωσης των αντικειμένων με drag and drop τεχνική, αλλά απλώς του επιτρέπει να δηλώσει το όνομα του αρχείου στο οποίο έχει αποθηκεύσει την περιγραφή του σεναρίου. Στη συνέχεια το σύστημα μεταφέρει τα περιεχόμενα του αρχείου στο λεκτικό αναλυτή όπου αναγνωρίζονται τα σύμβολα της γλώσσας και εντοπίζονται πιθανά συντακτικά λάθη. Στην περίπτωση της απουσίας συντακτικών λαθών το πρόγραμμα προχωρά στη σημασιολογική ανάλυση και για κάθε εντολή του χρήστη δημιουργείται ή κατάλληλη έξοδος στο αρχείο VRML. Η μετάφραση των εντολών του χρήστη σε VRML γίνεται με βάση συγκεκριμένους αλγορίθμους.

Στο κύριο μέρος της εργασίας και συγκεκριμένα στο τέταρτο κεφάλαιο που ασχολείται με την αρχιτεκτονική του συστήματος εξηγούμε με τη μορφή ψευδοκώδικα πως ακριβώς δουλεύει ο μεταφραστής (translator). Επιπλέον παραθέτουμε ένα πλήθος παραδειγμάτων που εξηγούν τη λογική μετάφρασης των εντολών του χρήστη σε VRML κόμβους. Πιο συγκεκριμένα παρουσιάζονται οι αλγόριθμοι δημιουργίας των αντικειμένων τύπου:

- παραλληλόγραμου (rectangle)
- έλλειψης (ellipsis)
- γραμμής (line)
- πολυγώνου (polygon) και
- ομάδας αντικειμένων (group)

Παράλληλα εξηγούνται πως ακριβώς δημιουργούνται οι κόμβοι στην περίπτωση του mouse click event, και πως υλοποιούνται οι πράξεις που συμπεριλαμβάνονται σ' αυτή την έκδοση του εργαλείου, οι πιο βασικές από τις οποίες είναι οι:

- start για την τοποθέτηση των αντικειμένων που έχουν ήδη δημιουργηθεί.
- Discrete Translation για τη διακριτή μετατόπιση των αντικειμένων
- Continuous Translation για τη συνεχή μετατόπιση και
- Continuous Rotation για τη συνεχή περιστροφή κλπ

Τις πράξεις (actions) μπορούμε να τις συνδέσουμε μεταξύ τους και να δημιουργήσουμε έτσι περισσότερο πολύπλοκα σενάρια. Στο τέλος του κεφαλαίου που ασχολείται με την αρχιτεκτονική του συστήματος δίνονται παραδείγματα υλοποίησης παρουσιάσεων όπως οι:

1. Δημιούργησε ένα παραλληλόγραμο μεγέθους 1, 4 και χρώματος κόκκινου. Όρισε την αρχική του θέση στο σημείο 3, 2 του χώρου που θα αρχικοποιηθεί. Όταν ο χρήστης χτυπήσει με το ποντίκι πάνω στο αντικείμενο, τότε το τετράγωνο θα πρέπει να μετατοπισθεί στην θέση 2, 1.
2. Δημιούργησε δύο παραλληλόγραμμα μεγέθους 1, 4 και 1, 3 και χρώματος κόκκινου και πράσινου αντίστοιχα. Αρχικοποίησέ τα στις θέσεις 3, 2 και 1,1 κι έπειτα όταν ο χρήστης κάνει click με το ποντίκι πάνω σ' αυτά τότε το κάθε ένα θα πρέπει να κινηθεί με βάση τον νόμο 2t + 5 για τον άξονα των x και t + 3 για τον άξονα των y.
3. Δημιούργησε ένα ορθογώνιο που μετά το mouse click να μετατοπίζεται με βάση τον κανόνα 2t + 5 για τον άξονα των x και t + 3 για τον άξονα των y ενώ παράλληλα να περιστρέφεται με βάση τον κανόνα 2t + 2.

Και στις τρεις παραπάνω περιπτώσεις οι παρουσιάσεις προκύπτουν από την σύνθεση των παραγόμενων κόμβων έτσι ώστε το αποτέλεσμα να είναι λειτουργικό. Παρόμοιου τύπου παρουσιάσεις είναι δυνατόν να κατασκευάσουμε ορίζοντας αντικειμένα διαφόρων τύπων και δίνοντας σε αυτά ποικίλες συμπεριφορές.

IANs και Βάσεις δεδομένων

Όπως ήδη έχουμε αναφέρει ο βασικός στόχος της εργασίας είναι να διερευνήσει τη δυνατότητα σύνδεσης των δύο επιστημονικών περιοχών (βάσεις δεδομένων και πολυμέσα) στην περίπτωση των Interactive Animations. Με την έννοια αυτή το τελευταίο κεφάλαιο της εργασίας είναι ίσως το πιο σημαντικό. Στην προσπάθεια αυτή ο στόχος είναι να εντοπίσουμε τις απαιτήσεις για ερωτήσεις πάνω σε τέτοιου είδους παρουσιάσεις. Η έννοια του σεναρίου μας βοηθάει προς αυτή την κατεύθυνση. Παρατηρούμε ότι μέσα από την περιγραφή του χρήστη μπορούμε να αντλήσουμε ποικίλες στατικές πληροφορίες όπως για παράδειγμα το χρώμα ενός αντικειμένου στην αρχή της παρουσίασης, την αρχική του θέση κλπ. Βλέπουμε όμως ότι για τέτοιου είδους παρουσιάσεις που τα δύο βασικά τους χαρακτηριστικά είναι η κίνηση και η αλληλεπίδραση υπάρχει επίσης ανάγκη για δυναμικές ερωτήσεις. Η μελέτη μας επομένως θα πρέπει να δώσει απαντήσεις στα ακόλουθα δύο ερωτήματα:

- Ποιες είναι οι στατικές επερωτήσεις (queries) που πρέπει να σχεδιάσουμε για την άντληση πληροφοριών από τις παρουσιάσεις.
- Ποιες είναι οι δυναμικές επερωτήσεις που θα πρέπει να σχεδιάσουμε για την άντληση πληροφοριών κατά την εκτέλεση των παρουσιάσεων.

Για τον εντοπισμό των στατικών επερωτήσεων βασιζόμαστε όπως είναι επακόλουθο στη γραμματική που σχεδιάστηκε σε προηγούμενο κεφάλαιο αλλά και σε ορισμένες έννοιες που συναντάμε στην θεωρία των χωροχρονικών βάσεων δεδομένων. Θεωρώντας αυτά τα

στοιχεία δημιουργούμε ένα μοντέλο δεδομένων (data model) βασιζόμενοι στον αντικειμενοστραφή σχεδιασμό (object oriented database theory).

Το μοντέλο που προκύπτει περιλαμβάνει μία σειρά από κλάσεις που συνδέονται μεταξύ τους σύμφωνα με τις σχέσεις που ορίζονται από την ίδια τη γραμματική. Οι πιο σημαντικές παραγόμενες κλάσεις είναι οι:

- Class IAN που περιγράφει τα γενικά χαρακτηριστικά της κάθε παρουσίασης, δηλαδή το όνομά, τον χρήστη που τη δημιούργησε, την ημερομηνία που δημιουργήθηκε, το σύνολο των αντικειμένων που συμμετέχουν σ' αυτήν, και το σύνολο των εντολών που ορίζουν την συμπεριφορά των αντικειμένων.
- Class IANobject που περιγράφει αντικείμενα. Κάθε αντικείμενο συνδέεται με την παρουσίαση στην οποία ανήκει, ενώ συνδέεται επίσης με τα γεγονότα αυτής της παρουσίασης στα οποία συμμετέχει, τις συνθήκες και τις πράξεις ή δράσεις. Παράλληλα κάθε αντικείμενο αυτής της τάξης διατηρεί στοιχεία σχετικά με τα βασικά του χαρακτηριστικά όπως τον τύπο του, τα σημεία που ορίζουν την γεωμετρία του κλπ.

Κατά αντίστοιχο τρόπο δηλώνεται και η κλάση που περιγράφει τις εντολές των παρουσιάσεων και που αποτελούνται σύμφωνα με τη γραμματική από γεγονότα (events), συνθήκες (conditions) και πράξεις (actions). Οι υπόλοιπες κλάσεις του μοντέλου συμπληρώνουν την περιγραφή και συντελούν στην πλήρη αποθήκευση του σεναρίου. Έτσι εντοπίζουμε τις κλάσεις γεγονός (class event), σύνθετο γεγονός (class complex event), συνθήκη (class condition), σύνθετη συνθήκη (class complex condition), πράξη (class action), απλή πράξη (class simple action) και τέλος κλάσεις που περιγράφουν την κάθε απλή πράξη ξεχωριστά (class start action, discrete translation, discrete rotation, discrete scaling, discrete color changing, continuous transformation , continuous Translation, continuous Rotation κλπ). Στο πέμπτο κεφάλαιο της εργασίας δίνεται μία πολύ καλή γραφική περιγραφή των κλάσεων που συνιστούν το μοντέλο δεδομένων και του τρόπου που αυτές συνδέονται μεταξύ τους έτσι ώστε ο αναγνώστης να αποκτήσει μία περισσότερο ολοκληρωμένη εικόνα του σχήματος.

Πάνω σ' αυτό το σχήμα βασιζόμαστε για να δώσουμε απάντηση στο πρώτο ερώτημα, δηλαδή τι είδους στατικές ερωτήσεις απαιτείται να σχεδιαστούν για ένα τέτοιο σύστημα. Ονομάζουμε την κατηγορία αυτή των ερωτήσεων, ως ερωτήσεις πάνω στα περιεχόμενα της βάσης (queries on database contents), και κατόπιν εκφράζουμε ποικίλα παραδείγματα αυτών χρησιμοποιώντας το συντακτικό της OQL (object query language). Ορισμένα παραδείγματα στατικών ερωτήσεων είναι τα ακόλουθα:

- Αναζήτησε τα αντικείμενα που συμμετέχουν σε μία παρουσίαση με το όνομα έστω "name"
- Αναζήτησε τις εντολές από τις οποίες αποτελείται μία παρουσίαση με το όνομα "name".
- Αναζήτησε αντικείμενο με συγκεκριμένο όνομα
- Αναζήτησε τον τύπο ενός αντικειμένου
- Αναζήτησε τα σημεία που προσδιορίζουν τον τύπο ενός αντικειμένου
- Αναζήτησε τα περιεχόμενα ενός αντικειμένου

- Αναζήτησε το χρώμα ενός αντικειμένου
- Αναζήτησε τα αντικείμενα που συνθέτουν ένα αντικείμενο σε περίπτωση που αυτό είναι σύνθετο.
- Αναζήτησε το όνομα, τον χρήστη και την ημερομηνία δημιουργίας μίας παρουσίασης στην οποία συμμετέχει ένα αντικείμενο
- Αναζήτησε τα γεγονότα με τα οποία συνδέεται ένα αντικείμενο
- Αναζήτησε τις πράξεις που ορίζουν τη συμπεριφορά ενός αντικειμένου.
- Αναζήτησε αντικείμενα που έχουν κόκκινο χρώμα
- Αναζήτησε αντικείμενα που έχουν ένα συγκεκριμένο τύπο.

Παρόμοιες ερωτήσεις μπορούν να χτιστούν πάνω σε όλες τις κλάσεις του σχήματος με τη χρήση OQL και με στόχο πάντοτε την άντληση design time πληροφοριών για τις παρουσιάσεις που δημιουργούν οι χρήστες.

Οι δυναμικές ερωτήσεις θα πρέπει να ικανοποιούν περισσότερο πολύπλοκες απαιτήσεις. Στη διερεύνηση αυτών των απαιτήσεων παρατηρούμε ότι η OQL πλέον δεν μας βοηθάει να εκφράσουμε τέτοιου είδους δυναμικές ερωτήσεις κι επομένως θα πρέπει να ορίσουμε νέες δομές για την επίτευξη αυτού του στόχου. Έτσι κινούμενοι μέσα σ' αυτό το πλαίσιο θα εξετάσουμε πρώτα τρία επιμέρους θέματα.

- Την ανάγκη για ύπαρξη νέων χωρικών και/ή χρονικών τελεστών
- Τον τρόπο με τον οποίον μπορούμε να ενσωματώσουμε στη γλώσσα επερωτήσεων τα γνωρίσματα των κινούμενων αντικειμένων, η τιμή των οποίων αλλάζει κατά την διάρκεια εκτέλεσης της παρουσιάσης (πρόκειται για τα λεγόμενα runtime attributes) και τέλος
- Την εγκυρότητα της παραδοσιακής δομής για επερωτήσεις δηλαδή της SELECT.. FROM..WHERE

Διερευνώντας κάθε ένα από τα παραπάνω ερωτήματα καταλήγουμε σε σημαντικά συμπεράσματα.
Σε σχέση με τους τελεστές διακρίνουμε την ανάγκη για ύπαρξη των:

- Meet που ελέγχει κατά πόσο δύο αντικείμενα έχουν κοινά σημεία ή όχι.
- Unmeet που ελέγχει κατά πόσο δύο αντικείμενα είναι ξένα μεταξύ τους
- Overlap που ελέγχει κατά πόσο δύο αντικείμενα έχουν κοινές περιοχές και τέλος
- Distance που υπολογίζει την απόσταση μεταξύ δύο αντικειμένων

Σε σχέση με τον τρόπο διαχείρισης των RunTime attributes εντοπίζουμε την ανάγκη ύπαρξης συναρτήσεων που θα έχουν την δυνατότητα να επιστρέφουν την ακολουθία τιμών των δυναμικά εξελισσόμενων γνωρισμάτων κατά τη διάρκεια της εκτέλεσης μιας παρουσίασης. Έτσι για να υπολογίσουμε το χρώμα ενός αντικειμένου σε μία συγκεκριμένη χρονική στιγμή αρκεί να χρησιμοποιήσουμε στην επερώτηση τη συνάρτηση colourOf(object). Αντίστοιχες συναρτήσεις ορίζονται για όλα τα γνωρίσματα που μεταβάλλονται δυναμικά.
Τέλος και σε ότι αναφορά την επάρκεια της παραδοσιακής δομής SELECT .. FROM .. WHERE παρατηρούμε ότι μία διαφοροποιημένη δομή θα μπορούσε να μας βοηθήσει να ξεχωρίσουμε τον χρόνο, το πιο σημαντικό ίσως κομμάτι των δυναμικών επερωτήσεων

από τις υπόλοιπες συνθήκες (conditions). Έτσι στην κλασική select προσθέτουμε ένα ακόμη μέρος, το when clause. Στο when clause της νέας δομής ορίζουμε όλες τις χρονικές εκφράσεις που σχετίζονται με τις συνθήκες του where.
Η χρήση του when αλλά και όλων των υπόλοιπων στοιχείων μας βοηθάει να να εκφράσουμε κάθε πιθανή δυναμική ερώτηση. Έτσι το τελευταίο μέρος της εργασίας είναι αφιερωμένο στην παρουσίαση δυναμικών επερωτήσεων οι πιο σημαντικές από τις οποίες είναι οι:

- Αναζήτησε αν δύο αντικείμενα συναντιούνται κατά την διάρκεια της εκτέλεσης ενός IAN.
- Αναζήτησε αν δύο αντκείμενα συναντώνται κατα την διάρκεια της εκτέλεσης ενός IAN στον χρόνο 4.
- Αναζήτησε αντκείμενα που έχουν κόκκινο χρώμα τη στιγμή που συναντιούνται
- Αναζήτησε αν ένα αντικείμενο έχει κόκκινο χρώμα ανάμεσα στο χρονικό διάστημα 10 και 15.
- Αναζήτησε τη χρονική στιγμή κατά την οποία δύο αντικείμενα συναντώνται.
- Αναζήτησε τη θέση ενός αντικειμένου σε μία συγκεκριμένη χρονική στιγμή.
- Αναζήτησε τον προσανατολισμό του αντκειμένου σε μία συγκεκριμένη χρονική στιγμή.
- Αναζήτησε την απόσταση δύο αντικειμένων την χρονική στιγμή 4 ή τέλος
- Αναζήτησε αντικείμενα που κατά το χρονικό διάστημα 10-15 έχουν γωνία ίση με 90 μοίρες και χρώμα κόκκινο.

Γενικά συμπεράσματα

Κλείνοντας τη σύντομη αυτή παρουσίαση στα ελληνικά παραθέτουμε σκέψεις και συμπεράσματα που προκύπτουν από τη συνολική μελέτη του θέματος. Τα βασικότερα οφέλη από την ενασχόλησή μας με το συγκεκριμένο θέμα είναι ότι πλέον έχουμε μία σαφή εικόνα της πληροφορίας που είναι χρήσιμο να αντλήσουμε από τέτοιου είδους παρουσιάσεις. Η πληροφορία αυτή έχει κατηγοριοποιηθεί (στατική, δυναμική κλπ) ενώ παράλληλα έχουν προταθεί ποικίλες ιδέες για τον τρόπο με τον οποίον μπορούμε να την αντλήσουμε. Επιπλέον προτείνεται για πρώτη φορά μία γλώσσα για τη δημιουργία τέτοιου είδους παρουσιάσεων. Επόμενο είναι ένα τέτοιο σύστημα να αποτελεί μία σημαντική προσπάθεια που παρά τις σημαντικές ελλείψεις του, μας οδηγεί σε ωφέλιμα συμπεράσματα για αυτό το χώρο.

Summary

This dissertation is placed at the intersection of two fields, multimedia and databases. Dissertation's basic aim is to propose an integraded system for storing, retrieving and manipulating presentations consisted of interactive animated objects (Interactive Animations). For that purpose we suggest a language for creation of Interactive Animation, and a data model where we can store information about them. The basic steps to achieve these goals are:

- To design a grammar specification for the language we propose
- To create a translator able to produce Interactive Animations written according to the new language
- To design a database schema and queries upon Interactive Animations

To implement these steps we use a model proposed by Dan Vodislav and M. Vazirgiannis [Vodislav-Vazirgiannis 1999]. We think this model is the most powerful because its major characteristics are:

- simplicity and
- orientation to databases

The model defines animation and all the basic notions related to it. The definitions are given according to the specific 2D animation model that it is adapted as base for the authoring and querying environment. The model define Interactive Animations and animated objects also. It distinguish between simple and complex objects. It defines the notion of coordinate systems, explains the difference between local and global coordinate system and discuss the difference between run-time and database status. Finally the model presents the basic methods related to simple or complex objects behavior.

System grammar is based on the scenario modelling results also. In dissertation, we propose a scenario structure, appropriate for Interactive Animations based on ECA rules (events conditions actions rules). ECA rules help introducing interaction to the language we intent to implement. According to the model and the ECA rules we build grammar specification. Grammar specification is even the most important part of our work as it will be the foundation for the translator and the queries design. In this specification all notions related to animation are included in a simple and declarative way. There is a structured presentation of objects and their behaviours and interaction is introduced using events and conditions. Translator uses as input this grammar in order to implement real scenarios.

Translator consists of three major objects: user interface, lexical analyzer and parser. Lexical analyzer is responsible for lexical analysis of the presentation written in our language while Parser is responsible for translating user description in a executable format (VRML).

The last step to our system implementation is data model and queries designed on this model. Trying to find out what kind of information we want to extract of Interactive Animations we classify queries in two groups: static and spatiotemporal queries. We investigate also the ability to query on the run time characteristics of an Interactive Animation. We explain what we need in addition to the standard OQL to express queries on dynamic data and we present the most important questions of the spatiotemporal category.

1. Introduction

Contents

1.1 Dissertation's scope and aim

The dissertation is placed at the intersection of two fields, multimedia and databases. Before describing dissertation's basic aim we consider important to explain in a few words these two different concepts and the relation between them.

Multimedia is the new way to present simple or complex information to the end–user, with a friendlier and more real manner. From the user' s perspective multimedia means that computer information can be represented through audio and/or video in addition to text, image graphics and animation. [Steinmetz 1995]. Multimedia spread became possible due to the great advances in information and communication technologies.

On the other side, the ***database*** is the basic and the most important element of an information system. For many decades databases support every kind of activity in the organisation and are related to many different systems: management information, operating and network systems as examples. As a natural result the new area of multimedia systems need a database support as well. We must have the opportunity to store manipulate and retrieve not only text or other discrete data but also audio, video, images and animation information. The requirements of the database support are even more complex when we have to put the different mono – media objects together on the same multimedia document, such as complex multimedia presentation, interactive WWW pages, animation and 3D Worlds [Vazirgiannis-Sellis 1998].

The scope of the dissertation is exactly that new scientific area of multimedia database management systems It is concentrated on multimedia presentations using animation so called ***Interactive Animations (IAN)***. We start by trying to implement a simple IAN authoring environment. We mention that the implementation of an authoring environment system is often a hard work. Hence for the creation of an IAN editor we need to do the following discrete steps:

- To define a simple declarative ***Interactive Animation model***.
- To design and implement a ***graphical user interface*** in order to have a user friendly authoring tool.
- To create a ***translator*** of an Interactive Animation to a rendering environment.

This dissertation is based on an existing interactive animation model (see paragraph 1.2) and tries to produce a grammar specification and a translator of Interactive Animations according to that specification. The specification must be configured in such a way so

that it will be easier afterwards to connect the produced Interactive Animations with a database. In other words dissertation will cover only the last step of the authoring environment creation procedure, previously described. Design and implementation of a graphical user interface require too much time and effort to be treated in this dissertation. We leave this part for future work.

The second part tries to investigate the database issues related to Interactive Animations. Based on the Interactive Animation scenario model and the produced grammar specification, we explore the following issues:

- Expressing a data model for Interactive Animations
- Defining queries on database contents in order to extract static information about Interactive Animations, the participating objects and their spatial and temporal relationships.
- Defining queries not only on the database contents but also on the run-time objects behaviour.

All these database issues represent a new and very challenging research field. Our goal is to explore in general level the database representation, storing and retrieval of IANs.

1.2 Related Work

In order to design and implement an ***Interactive Animations authoring environment*** we need a simple and generic model of animated behaviour as it has been mentioned above. By browsing in the bibliography we can find a large amount of papers proposing models for animation. (see References Papers – Technical reports about other animation models) However these models are specialised and define animation according to the application they are interested to. Hence these models can't be used as a basis for the creation of interactive animation multimedia presentations and database-oriented applications.

The dissertation will be based on the model proposed by Dan Vodislav and Michalis Vazirgiannis in the paper "Generic Object Modelling for Animation Contexts". This model seems to have all the characteristics needed to become the foundation of an authoring environment system and of the queries design. It is simple, generic, object and database oriented, in other words it has the most important attributes in order to help us model and implement an Interactive Animation description language and to design queries upon these Interactive Animations. We describe in detail this model in chapter two.

Another important influence to this dissertation is the work that has already been done in the area of Interactive Multimedia Documents. We can find in the bibliography, a model for the description of Interactive Multimedia Presentations (IMDs), an implementation effort of an authoring environment [Vazirgiannis 1997] and a complete database oriented approach [Vazirgiannis-Sellis 1998]. In these papers authors explore multimedia presentations consisting of mono media objects such as text, image, video or sound connected in a way so that the whole composition has a sense as presentation. The user

writes ***a scenario***, (a composition with definitions of the participated multimedia objects, their spatial and temporal relationships, and actions as results of user interaction with the multimedia presentation (like mouse, button click etc.)) and then translates this scenario to an executable format. The execution's result is an interactive multimedia presentation upon which we can query and extract information. In fact this work triggered the idea for the creation of Interactive Animations authoring environment.

From Interactive Multimedia Documents (IMD) related work, we have adopted the notion of scenario modelling and the notion of events. Events play a critical role as they trigger animation transformations and enable user interaction. We have then formulated these notions according to our animation model and theory. The influence from the work in the area of IMD is obvious in some analysis and design decisions, as we will see in the following chapters.

In the database section, we are influenced by [Guting and al. 1998]. This team presents a foundation for representing and querying moving objects. Some of the aspects of this work are really very important for the classification of queries on Interactive Animations that we are trying to do.

1.3 Synopsis

The introduction chapter describes the basic aims of this dissertation. We summarise these aims in the following table.

Dissertation aims
▪ Define a language and design a grammar specification in order to give user the opportunity to describe Interactive Animations
▪ Create a translator able to produce Interactive Animations according to the grammar specification and the user instructions.
▪ Design queries upon Interactive Animations.

Table 1: A brief description of dissertation basic aims

2. Basic definitions and animation model presentation

Contents

2.1 Animation

As mentioned in the previous paragraphs, the most important characteristic of our model is ***animation***. An Interactive Animation presentation consists of animated objects in contrast to Interactive Multimedia Documents where objects are static and without motion.

But what exactly animation is? Looking at a dictionary we will find the literal definition: ***To animate something means to bring it to life***. Animation covers all changes that have a visual effect. Visual effects can be of different nature. They might include time-varying positions (motion dynamics), shape, colour, transparency, structure and texture of an object (update dynamics), and changes in lighting, viewer position, orientation and focus. Having in mind this physical description we can now define the computer-based animation, we are interested to, as animation performed by a computer using graphical tools to provide visual effects [Steinmetz 1995]. Consequently we can give a general definition of ***Interactive Animation*** presentations (IAN) as presentations consisting of animated objects, providing to user various ways of interactivity.

We have now a clear idea of what animation and Interactive Animation presentation is. The next step in order to have a deep understanding of the model is to define animated objects that are participating in the presentation.

2.2 Animated objects

Animated objects are the main elements of the animation model, and they can be either simple or complex. The model gives a clear definition for simple objects as follow:

Simple animated object definition
A simple object is a tuple (***Form***, ***Sp***, ***Gr***, ***CT***, ***vis***, ***as***, ***it***), where ***Form*** denotes the object's form, ***Sp*** its spatial status, ***Gr*** the graphical status, ***CT*** its current continuous transformations, ***vis*** the current visibility status (***vis*** = true means visible), ***as*** is the object's activity status (active, suspended or idle) and ***it*** its internal time.

Table 2: Simple animated object definition

In other words a simple object has as major attributes:

- The ***form***: Form is the combination of ***geometry*** and its ***contents***. By using the term geometry we mean its type as line, rectangle, ellipse or polygon, and a group of points that define the object's form.
- The ***spatial status:*** It consists of the object's position in the IAN, its angle and it's scaling factors.
- The ***graphical status:*** Graphical status is described by the color, the line style, the line width, the filling style, the filling ratio etc. In this first implementation version we are interested only to the color but in the future we will treat other attributes as well.
- ***Continuous transformations:*** They are defined for translation, rotation and scaling of an object, and for the continuous changing of the color and can be fully described giving the transformation law as a function of time. The effect of a continuous transformation depends on the private transformation time and the transformation activity status. The activity status of a continuous transformation shows if a transformation is currently IDLE (no current continuous transformation exists), SUSPENDED (continuous transformation has been started and is currently paused) or ACTIVE (continuous transformation has been started and is not currently paused). Private time of continuous transformation starts from 0 each time a new continuous transformation is started and flows only when transformation's activity status is ACTIVE and when the object's activity status is active.
- ***Visibility:*** It is the attribute that defines if the object is shown or not in the Interactive Animation.
- ***Activity status:*** Activity status is ACTIVE when the object is instantiated in the Interactive Animation, became suspended when the object is paused and is ACTIVE again when the object is resumed. Notice that object's and continuous transformation's activity status are two completely different notions
- ***The internal time:*** Internal time finally is a time related directly to the object, starts when the object is instantiated and flows only when object activity status is ACTIVE.

Simple objects may be grouped together and form complex objects. As an example a car can be defined as a group of three different simple objects: two ellipsis (the wheels) and a polygon (the car's body). The interesting point in complex objects is that each part of the group can have its own transformation while the group has its own motion law but according to semantic and spatial constraints. A complex object definition is presented in the following table:

Complex object (group) definition
A complex object (group) is a tuple (***Form, Sp, Gr, CT, vis, as, it, Comp***) containing the same elements as a simple object, plus ***Comp***=[o_1,…..o_n] the group's list members, which may be simple or complex objects

Table 3: Complex object definition

After describing the object form and the object's major characteristics we have to position it to the IAN. A coordinate system must be defined for this purpose. As the model is 2D a coordinate system definition is the following:

Co-ordinate system definition
A coordinate system (CS) within a 2D space is defined by a reference point (the origin O), two orthogonal rays starting from O (the x and y-axis) and a metric system M = (u_x, u_y), where u_x, $u_y > 0$ define the unit length for each axis.

Table 4: Coordinate system definition

Object geometry is defined in the ***local coordinate system*** (LCS) while object position to the IAN is defined in the ***Global Coordinate System*** (GCS). The local coordinate system of an object is a CS created when the object is instantiated within the IAN. Initially, its origin is the object's reference point, the orientation is parallel to the GCS and the metric system is (1, 1). For each object within an IAN there is a unique CS called the ***parent CS of the object***. For simple objects, the parent CS is the GCS, but for group members this is the LCS of the group. The spatial status of an object is expressed relatively to the parent CS.

Another significant point to remark in the model is that some of the previous attributes or elements constitute the ***run-time status*** while some others constitute the ***database status***. This distinction is very important, as we are interested not only to the authoring environment creation but also to the design of a database schema for supporting this authoring environment. So the run-time status completely describe the object at any moment of its animation while the database status of an object includes all the elements that describe the object as an independent entity out of any IAN.

At last the model defines various methods related to the simple and complex objects behavior. The most important simple object's methods are:

- ***start***: Start method instantiates the object at a specific position in the IAN and defines its initial visibility.
- ***pause***, ***resume***, ***stop***: These methods modify the object activity status.
- ***translation, rotation, scaling, color:*** This group of functions apply the geometric and graphical transformations to the objects and finally
- ***hide, show:*** These methods hide and show the object

For a group object the model provides the same methods as in the simple objects, defining also the following additional methods

- ***parent***: Parent method returns the parent of the object or a special void value if there is no parent.
- ***add_member:*** This function adds a new member to a group.
- ***remove_member:*** This function removes a member from the group and finally
- ***destroy:*** Destroys an object or a group.

2.3 Interactive animation scenario

In order to put together all notions described in the previous paragraph we need to model a scenario. A ***scenario*** describes an animation. It is a list of "stages", each one consisting of one or several actions. The scenario also indicates the objects participating in each

"stage" and the spatiotemporal relations between them. One of the basic dissertation's aims is to configure a more detailed IAN scenario model adding the notions of events and conditions inside it and to produce the relevant grammar specification. Using this specification we will then build the translator.

Influenced by the scenario modeling already done in the multimedia documents authoring system, we find meaningful to adjust the notions of events, conditions and actions in the Interactive Animation too. Therefore we refine the previous scenario definition, and we define scenario as a list of statements ("stages" according to [Vodislav-Vazirgiannis 1999]) where each statement is an ECA rule (event, condition, action rule).

The ECA rule structure is really very powerful because it gives an easy way to define interaction and to model different behaviors. Defined in the scientific area of active databases an ECA rule is a statement consisted of three different parts: events (E), conditions (C) and actions (A), and their connection is implemented according to the following form:

On event **if** condition **then** action

According to [Vazirgiannis-Boll 1996] "*an event is defined as an instantaneous happening of interest*". Additionally an event is caused by some action that happens at a specific point in time and may be atomic or composite. In the multimedia literature events are not uniformly defined. Events are defined differently depending on the multimedia application where they are used. In Interactive Animations we define event as:

"An event is raised either by a user interaction action, by the change in the value of an object attribute, by the interaction between two or more object, or by the IAN timer. Event has attached a temporal instance and can be processed then by a procedure defined to handle this kind of events"

In this abstract definition a temporal instance is a time value calculated using the beginning of the interactive animation as a reference point. The previous definition probably isn't mature enough but at least is a first approach to event definition in the area of Interactive Animations.

In [Vazirgiannis-Boll 1996] we find also an interesting event classification approach for multimedia documents. We find important to do the same in Interactive Animations. So we define the following groups:

- ***User interaction events***: These kinds of events are caused by the interaction between the user and the IAN. Such events can be ***mouse events*** (mouse click, mouse drag, mouse release etc) or ***keyboard events*** which are generated each time a user presses or releases a key.
- ***Intra object events***: In Interactive Animations intra object events are defined as events that are recognised due to a change to some specific attributes of the objects participating in an IAN (visibility, colour, position change etc)

- ***Inter objects events***: Such events take place when two or more objects are involved in the occurrence of an action of interest and are raised if spatial and/or temporal relationships between two or more objects hold. The meeting of two different objects in an Interactive Animation is an example of Inter Objects Events.
- ***Complex events***: Complex events results from the combination of the previous described, so called simple events (user interaction, intra and inter objects events). We can combine simple events using various different ways in order to produce complex events. In our scenario modelling we adapt the following operators:
    1. e = ANY(k, e1,…,en): This event occurs when at least any of k of the events e1,…,en occur.
    2. e = SEQ(e1,….,en): This event occurs when all events e1…en occur in the order appearing in the list.
    3. e = TIMES(n, e1): This event occurs when there are n consecutive occurrences of event e1.

This classification will be our basic reference for the production of event part in the grammar specification.

Events are probably the most important elements of an ECA rule as an action can be triggered only by the occurrence of an event. But ***conditions*** also play a significant role. Even if an event is occurred, the ECA rule condition must be fulfilled so that the action that follows, take place. Conditions can be related to the internal status of the object (position, colour, activity status etc) but also to the occurrence of some topological relationships (as meet, disjoint etc). An abstract definition for condition can be the following:

"Conditions represent predicates that can be evaluated to true or false and thus affect accordingly the execution of related actions"

At last actions are the final part of each ECA rule and they are usually methods applied to the object in order to create different behaviors. Transformations, rotation, translation, scaling are example of actions that must be modeled in the scenario.

2.4 Synopsis

The second chapter defines animation and all the basic notions related to it. The definitions are given according to the specific 2D animation model [Vodislav-Vazirgiannis 1999] that it is adapted as base for the authoring and querying environment. In this chapter we define animation, Interactive Animations and animated objects. We distinguish between simple and complex objects. We define the notion of coordinate systems and we explain the difference between the local and the global coordinate system. We also discuss the difference between run-time and database status and we present the basic methods related to simple or complex objects behavior. The last part of the chapter is dedicated to the scenario modeling. In order to describe animation, a scenario must be written. We propose a scenario structure, appropriate for Interactive Animations based on ECA rules and we finish the definitions by further explaining

events, conditions and actions as ECA rule components. Having in mind all this theoretical background it is easier now to produce an appropriate grammar specification for Interactive Animation.

3. Grammar Specification

Contents

3.1 Grammar Specification presentation

Grammar specification is structured according to the basic animation's notions and defines Interactive Animations in a strict way. Hence IAN is defined as the sequence of objects and a scenario attached to these objects.

```
IAN ::= objects scenario;
```

Grammar specification presentation 1: IAN definition

Each object has a name as identity and is fully described by its point list, its type (accepted types are LINE, RECTANGLE, ELLIPSE POLYGON and GROUP), its contents (contents can be a text, a bmp file etc) and its colour (in r g b format). If the object is of type GROUP then it must also contains a composition of other objects (so called children). The objects part of grammar specification is next presented:

```
objects ::= object objects | empty;
object ::= OBJECT  object_name OPBRAC  form colour
           composition CLOSEBRAC;
object_name ::= STRING;
form ::= FORM OPBRAC geometry contents CLOSEBRAC;
geometry ::= type point_list;
type ::= TYPE type_value;
type_value ::= LINE
               | RECTANGLE
               | ELLIPSE
               | POLYGON
               | GROUP;
point_list ::= POINTS points;
points ::= point points | empty;
point ::= OPARENT coord_x COMMA coord_y CLOSEPARENT;
coord_x ::= NUMBER;
coord_y ::= NUMBER;
contents ::= CONTENTS contents_value | empty;
contents_value ::= FILE file_name | contents_name;
contents_name ::= STRING;
file_name ::=  STRING;
colour ::= COLOUR r g b | empty;
```

```
r ::= NUMBER;
g ::= NUMBER;
b ::= NUMBER;
composition ::= COMPONENT OPBRAC objects CLOSEBRAC | empty;
```

Grammar specification presentation 2: Objects definition

At this point we must mention that the point list has only one element in the case of rectangle and ellipsis. In case of line and polygon the point list can have as many points as the user defines and in case of group the points describe the position of every child in the group, so the number of points in the point list is the same as the number of children in the group.

A scenario is defined as a sequence of statements and each statement is an ECA rule, so is consisting of event, conditions and actions.

```
scenario ::= statements;
statements ::= statement statements | empty ;
statement ::= event conditions scenario_actions;
```

Grammar specification presentation 3: Scenario definition

Events can be simple or complex. According to the events classification in chapter two a simple event can be a user interaction, intra or inter object event. We also introduce the notion of time events to capture events that are raised when the animation time takes a specific value. User interaction is related either to a specific object or to the interactive animation in general, and can be expressed by a mouse click or by a keyboard press. Intra object events are expressed by a change in the value of the basic database and runtime attributes (colour, contents, visibility, activity status, internal time, position, angle and scaling). Inter object events are triggered when a special topological relation between two object takes place. In the case of our model we found that only meet (the instant when two objects meet one the other) and not meet (the instant when two objects don't have common points) have sense in Interactive Animations. Time events include the start event or a time expression. Complex events finally are defined according to the definitions given in the chapter two.

```
event ::= EVENT event_description;
event_description ::=  simple_event
                     | complex_event;
simple_event ::= user_interaction_event
               | intra_object_event
               | inter_object_event
               | time_event;
user_interaction_event ::= user_action
                         | object_name POINT user_action;
user_action ::= MOUSE_CLICK
              | KEYBOARD_PRESS;
```

```
intra_object_event ::= object_name POINT COLOUR IS r g b
                     | object_name POINT CONTENTS IS STRING
                     | object_name POINT VIS IS vis_value
                     | object_name POINT AS IS as_value
                     | object_name POINT IT IS it_value
                     | object_name POINT POS IS pos_value
                     | object_name POINT ANGLE IS
                       angle_value
                     | object_name POINT SCALE IS
                       scale_value;
vis_value ::= TRUE | FALSE;
as_value ::= ACTIVE
           | IDLE
           | SUSPENDED;
it_value ::= NUMBER;
pos_value ::= OPARENT pos_x COMMA pos_y CLOSEPARENT;
pos_x ::= NUMBER;
pos_y ::= NUMBER;
angle_value ::= NUMBER;
scale_value ::= OPARENT scale_x COMMA scale_y CLOSEPARENT;
scale_x ::= NUMBER;
scale_y ::= NUMBER;
inter_object_event   ::=   relation   OPARENT   parameters
CLOSEPARENT;
relation ::= MEET | NO_MEET;
parameters ::= object_name COMMA object_name;
time_event ::= START | TIME IS NUMBER;
complex_event ::= ANY OPARENT count COMMA simple_events
                  CLOSEPARENT
                | SEQ OPARENT simple_events CLOSEPARENT
                | TIMES OPARENT count COMMA simple_event
                  CLOSEPARENT;
simple_events ::= simple_event simple_events | empty;
count ::= NUMBER;
```

Grammar specification presentation 4: Events definition

The next part in an ECA rule is conditions. A condition can be simple or complex. Simple conditions are defined as a set of expression using equality or comparison operators. These expressions may be true or false. Complex conditions are constructed using simple conditions connected with OR and AND operators.

```
conditions ::= CONDITION  condition_description | empty;
condition_description ::= simple_condition
                        | complex_condition;
simple_condition ::= state_condition
                   | inter_object_event
```

```
                    | time_condition;
state_condition ::= object_name POINT COLOUR
                    equality_operator r g b
      | object_name POINT CONTENTS equality_operator STRING
      | object_name POINT VIS equality_operator vis_value
      | object_name POINT AS equality_operator as_value
      | object_name POINT IT comparison_operator NUMBER
      | object_name POINT POS comparison_operator pos_value
      | object_name POINT ANGLE comparison_operator
        angle_value
      | object_name POINT SCALE comparison_operator
        scale_value;

equality_operator ::= EQUAL | NOT_EQUAL;
comparison_operator ::= EQUAL
                      | NOT_EQUAL
                      | GREATER
                      | LESS
                      | GREATER_THAN
                      | LESS_THAN;

time_condition ::= TIME comparison_operator NUMBER;
complex_condition ::= or_condition AND or_condition
                    | simple_condition AND
                      simple_condition

                      | or_condition
                      | empty;
or_condition ::= OPARENT simple_condition OR
                 simple_condition CLOSEPARENT;
```

Grammar specification Presentation 5: Conditions definition

The last part of the grammar specification is actions. Similarly to events and conditions, actions can be simple or a sequence of simple actions. An action is always related to an object. Between the most important actions we distinguish discrete transformations (discrete rotation, translation or scaling) as well as continuous transformations (continuous translation, rotation and scaling).

```
scenario_actions ::=  ACTIONS actions_description |
                      NO_ACTION;
actions_description ::= simple_action | sequence_action;
simple_action ::= action_object_name POINT method;
action_object_name ::= STRING;
method ::= START OPARENT action_pos_value COMMA
           action_vis_value CLOSEPARENT
           | STOP OPARENT CLOSEPARENT
           | RESUME OPARENT CLOSEPARENT
```

```
            | PAUSE OPARENT CLOSEPARENT
            | HIDE OPARENT CLOSEPARENT
            | SHOW OPARENT CLOSEPARENT
            | ADD_MEMBER OPARENT object_name CLOSEPARENT
            | REMOVE_MEMBER OPARENT object_name
              CLOSEPARENT
            | DESTROY OPARENT CLOSEPARENT
            | transformation;
transformation ::= discrete_transformation
                 | continuous_transformation;
discrete_transformation ::= TRANSLATION OPARENT pos_value
                              CLOSEPARENT
                          | ROTATION OPARENT angle_value
                            CLOSEPARENT
                          | SCALING OPARENT scale_value
                            CLOSEPARENT
                          | COLOR_CHANGING OPARENT r g b
                            CLOSEPARENT;
continuous_transformation ::= TRANSLATION OPARENT
                    transformation_name COMMA law COMMA law
                    CLOSEPARENT
    | ROTATION OPARENT law CLOSEPARENT
    | SCALING OPARENT law COMMA law CLOSEPARENT
    | COLOR_CHANGING OPARENT law COMMA law COMMA law
      CLOSEPARENT;
    | STOP OPARENT transformation_name CLOSEPARENT
    | RESUME OPARENT transformation_name CLOSEPARENT
    | PAUSE OPARENT transformation_name CLOSEPARENT
transformation_name ::= STRING;
law ::= constant TIME PLUS constant
       | constant TIME MINUS constant
       | constant TIME;
constant ::= NUMBER;
action_pos_value ::= OPARENT action_pos_x COMMA
                     action_pos_y CLOSEPARENT;
action_pos_x ::= NUMBER;
action_pos_y ::= NUMBER;
action_vis_value ::= TRUE | FALSE;
action_scale_value ::= OPARENT action_scale_x COMMA
                       action_scale_y CLOSEPARENT;
action_scale_x ::= NUMBER;
action_scale_y ::= NUMBER;
sequence_action ::= SEQ OPARENT simple_actions CLOSEPARENT;
simple_actions ::=  simple_action dt simple_actions |
                    empty;
dt ::= NUMBER;
```

```
empty ::=  ;
```

Grammar specification presentation 6: Actions definition

3.2 Synopsis

Grammar specification is even the most important part of our work as it will be the foundation for the translator and the queries design. In this specification all the notions related to animation are included in a simple and declarative way. There is a structured presentation of objects and their behaviours and interactivity is introduced using events and conditions. Translator uses as input this grammar in order to implement real scenarios.

4. System architecture

Contents

4.1 General translator architecture

Translator consists of three different ***components***:

- User Interface.
- Lexical Analyser and
- Parser

Input to the system is the Interactive Animation description implemented according to the grammar defined in the previous chapter. Interactive animation description can be a simple ASCII file (simple_scenario.txt as an example).

Output of the system is a file that can be rendered to a browser supporting VRML (Virtual Reality Modelling Language). VRML is a 3D-file interchange format [Carey 1997]. It defines most of the commonly used semantics found in today's 3D applications such as hierarchical transformations, light sources viewpoint, geometry, animation, fog, material properties and texture mapping. Some other important VRML's characteristics are:

- VRML is also a 3D analogue to HTML and serves as a simple, multiplatform language for publishing 3D Web Pages and
- VRML provides the technology that integrates three dimensions, two dimensions, text and multimedia into a coherent model

As a 3D format VRML provides the majority of structures necessary for the implementation of our 2D-animation model. It will be also easier in the future to integrate our animation model with some basic 3D notions. The output is a file with the extension wrl and with the default name IAN (IAN.wrl) Before explaining each of the three translator components in detail, it is useful to give a simple graphical representation of the system in the figure below.

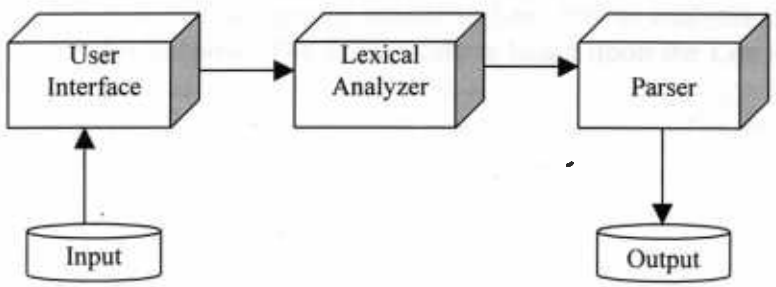


Figure 1: General Translator Architecture

4.2 User Interface

As we have already mentioned above the user interface doesn't provide at this moment any important facility (like drag and drop, ability to reuse objects etc). It is a simple Java window with a text box for receiving the input filename, a label for displaying any message raised during execution time and a click button that activates lexical analyser and parser components. The user interface form is presented in the figure below.

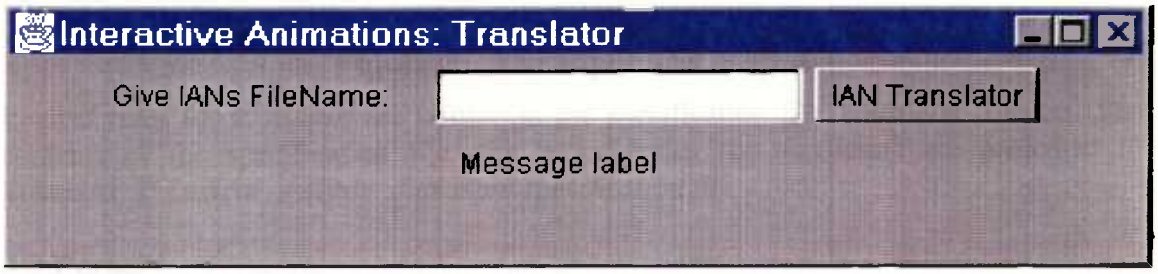


Figure 2: User Interface

User Interface is defined in the Translator.java module of the system. This module is responsible not only for receiving the description filename but also for implementing the connection with lexical analyzer and the parser. Translator module code is presented in *Appendix A Code listing, A1: Translator module.*

4.3 Lexical Analyser

IAN Translator is actually a compiler as it takes an abstract user description according to a grammar, then gives semantic to this description and finally generates code. According to the compiler's theory the two major parts are the lexical analyser and the parser. Parser is presented in the next chapter.

A lexical analyser breaks an input stream of characters into tokens. Writing lexical analysers by hand can be a tedious process, so software tools have been developed to ease this task [Berk 1997]. Perhaps the best known such utility is Lex. Lex is a lexical analyser generator for the UNIX operating system, targeted to the C programming language. Lex takes a specially formatted specification file containing the details of a lexical analyser. This tool then creates a C source file for the associated table-driven lexer.

As Translator is written in Java we need a similar to Lex, lexical analysis tool. JLex is the most famous tool for this purpose. The JLex utility is based upon the Lex lexical analyser generator model. JLex takes a specification file similar to that accepted by Lex, then creates a Java source file for the corresponding lexical analyser. The following figures present the whole procedure in a graphical way.

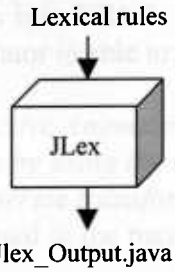


Figure 3: JLex input and output

Lexical rules for translator are written in the specification file Translator_Jlex.txt. The contents of the specification file (the lexical rules for translator) are presented in *Appendix A: Lexical analysis: Jlex input specification file.*

4.3 Parser

4.3.1 General Parser representation

Parser is the third and the most important component of the Translator. As with the lexical analyser, there are tools that automatically creates the parser code according to a specification file. Yacc is the most well-known Unix tool for parser creation and as Lex it generates code in C programming language. For creating the translator's parser we will use the Java Cup tool. CUP is a system for generating parsers from simple specifications. It serves the same role as the widely YACC and in fact offers most of the features of YACC. However, CUP is written in Java, uses specifications including embedded Java code, and produces parsers, which are implemented in Java. The following figure presents how Java Cup is used.

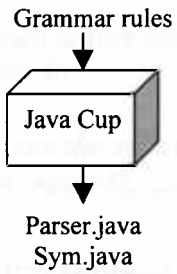


Figure 4: Java Cup Input and Output

Grammar rules are described in the Java Cup specification file (Translator_JCup.txt). The specification file is divided in three basic sections:

- The action code presented in *Appendix A, A3 Parser: Action code*
- The definition of terminal and non terminal symbols presented in *Appendix A: A4 Parser: Definition of terminal and non terminal symbols* and finally
- The grammar presented in *Appendix A: A5 Parser: Grammar part of the grammar*

The action code part is the most important section of the whole system. Actually in this part functions that do the translation into VRML are defined. Translator doesn't provide full functionality yet. Till now translator is able to:

1. *Create all object types*
2. *Starting the objects in the Interactive Animation*
3. *Provide the simplest interactivity by using the mouse click event and*
4. *Implementing continuous and discrete transformations*

Translator logic and design is presented in the paragraph that follows.

4.3.2 Translator abstract algorithm

Using the natural language with a more structured way we will try to present how translator works:

1. *For each object defined by the user in the IAN*
   1.1 Extracting from IAN user description all object's parameters.
      1.1.1 Extracting object name
      1.1.2 Extracting object type
      1.1.3 Extracting object point list
      1.1.4 Extracting object contents
      1.1.5 Extracting object color
   1.2 Assign objects parameters to an object of the class scenarioObject
   1.3 Put created scenarioObject in the Object List
2. *For each statement defined by the user in the IAN*
   2.1 Create object statement
   2.2 Assign event parameters for each event connected to this statement
   2.3 Assign condition parameters for each condition connected to this statement
   2.4 Assign action parameters for each action connected to this statement
   2.5 Assign statement in the statement list
3. *Opening VRML output file*
4. *For each object in the object list create the suitable proto node*
5. *For each object in the statement list create the suitable VRML node*
6. *Close VRML output file*

Figure 5: An abstract algorithm representing how translator works

From the previous algorithm we can understand that the main goal of the translator is to create the right nodes in the output files. This can be sometimes a very complicated task. To do that we use as basic data structures a list of objects, and a list of statement each one holding elements about objects and ECA rules. Next paragraph show various examples of simple scenarios and VRML nodes created by the translator as a result of the previous process.

4.3.3 Scenario examples and produced VRML nodes

4.3.3.1 Creating simple objects

There are five different types of simple objects that the translator must be able to create: rectangle, line, polygon, ellipsis and group. An example definition of a rectangle is presented below:

```
OBJECT <Name> {
                FORM {
                        TYPE RECTANGLE
                        POINTS (<x_coord>, <y_coord>)
                        CONTENTS  FILE <contents_filename>
                      }
                COLOUR <r> <g> <b>
              }
```

Figure 6: Rectangle definition

Parameters in this definition are expressed by using the < and > symbols. The user gives the real values and Translator uses them in order to create the real object in VRML. For creating a rectangle, parameters are the object name, the type with value *RECTANGLE*, the point list which in the case of a rectangle needs only one point (defined by *x_coord* and *y_coord* values), the contents defined by *contents_filename* parameter and the colour expressed in r g b format. For this object definition, Translator defines a prototype creating a PROTO node. This prototype can be then used during the scenario part of the Animation. The produced PROTO node is the following:

```
#VRML V2.0 utf8
PROTO <Name>[ ]
{
        Shape {
             geometry Box { size <x_coord> <y_coord>
                            0.0000001 }
             appearance Appearance {
                 material Material {
                          diffuseColor <r> <g> <b> }
                 texture ImageTexture { url
                          <contents_filename> }
               }
             }
} #end of RECTANGLE prototype
```

Figure 7: Code produced by Translator to create a rectangle.

Notice that in order to create a 2D object, as rectangle is, using the 3D environment of VRML, we select the geometry Box node giving at z parameter a very small value. The same logic of prototyping is used for the creation of ellipsis. A sample definition can be:

```
OBJECT <Name>    {
                     FORM
                     {
                        TYPE ELLIPSE
                        POINTS (<x_coord>, <y_coord>)
                     }
                     COLOUR <r> <g> <b>
                  }
```

Figure 8: Ellipsis definition

In this scenario we can see that in order to define the form we need only one point. Parameters are expressed, as in the rectangle definition. The PROTO node produced by the translator for the ellipsis definition is the following:

```
#VRML V2.0 utf8
PROTO <Name> [ ]
{
   Transform {
               rotation  0 1 1 3.14
               scale     <y_coord>/2 1 1
               children  Shape {
                 geometry Cylinder { radius <x_coord>/2
                                     height 0.00000000001 }
                 appearance Appearance {
                 material Material { diffuseColor <r> <g>
                                     <b> }
                 }
               }
            }
} #end of ELLIPSE prototype
```

Figure 9: Code produced by Translator to create an ellipsis.

Notice that in order to produce an ellipsis, Translator uses a Transform node. Changing the scale attribute (by dividing y_coord by 2) it defines the first radius. The second radius is defined by using the attribute radius in the geometry cylinder node and assigning to it the x_coord/2 value. Cylinder is a 3D object. So in order to produce an ellipsis we have to assign a near to zero value to the height attribute. Appearance and Material nodes are created in order to assign color to the ellipsis.

An example for line creation is presented in figure 10. The parameters here are: the name, the point list that defines the line and the color. The point list can have more than one, values. For example, if line is defined using two points, that means that this line is consisted of two line segments.

```
OBJECT <Name> {
          FORM {
                TYPE LINE
                POINTS (<x1_coord>, <y1_coord>)
                       (<x2_coord>, <y2_coord>)
                        ..
                       (<xn_coord>, <yn_coord>)
               }
        COLOUR <r> <g> <b>
}
```

Figure 10: Line definition

The code produced by the translator is presented in figure 11 below

```
#VRML V2.0 utf8
PROTO <Name> [ ]
{
     Shape {
            geometry IndexedLineSet {
            coord Coordinate { point [
                 <x1_coord> <y1_coord> 0
                 <x2_coord> <y2_coord> 0
                 ..
                 <xn_coord> <yn_coord> 0
            ] }
            coordIndex [ 0 1 .. n -1 ]
            color Color { color [<r> <g> <b>]}
            colorPerVertex FALSE
           }
     }
} #end of LINE prototype
```

Figure 11: Code produced by Translator to create a line

Translator in order to define line uses the geometry IndexedLineSet. In the coordinate node we define the points assigning to the z axe the zero value. Color is also assigned in the appropriate node. Next object is polygon. An example definition for polygon is presented below:

```
OBJECT <Name> {
```

```
FORM {
      TYPE POLYGON
      POINTS (<x1_coord>, <y1_coord>)
             (<x2_coord>, <y2_coord>)
             ..
            (<xn_coord>, <yn_coord>)
      }
 COLOUR <r> <g> <b>                    }
```

Figure 12: Polygon definition

The produced PROTO node uses the IndexedFaceSet geometry and creates a coordinate node inside using the points given to describe polygon form. Translator according to a specific algorithm connects these points and creates the polygon faces. The VRML file produced is presented below:

```
#VRML V2.0 utf8
PROTO <name> [ ]
{
   Shape {
           geometry IndexedFaceSet {
           coord Coordinate { point point [
                 <x1_coord> <y1_coord> 0
                 <x2_coord> <y2_coord> 0
                 ..
                 <xn_coord> <yn_coord> 0 ]
           }
           coordIndex [
            <create all the possible points combinations> ]
           color Color { color [<r> <g> <b>,..
                                <r> <g> <b> n times]}
           colorPerVertex FALSE
           }
        }
} #end of POLYGON prototype
```

Figure 13: Code produced by Translator to create a Polygon

Notice that in coordIndex field we must assign all the possible combinations of points given putting –1 as terminal point each time. For example if we have 4 points we must create the following combinations: 0 1 2 –1, 0 2 3 –1, 0 1 3 –1 and 1 2 3 –1.

The last example for object's creation concerns group of objects. We show an example of a group definition using parameters. The group is consisted of n children so it has n points definition in the field points.

```
OBJECT <Name> {
        FORM {
                TYPE GROUP
                POINTS (<x1_coord>, <y1_coord>)
                       (<x2_coord>, <y2_coord>)
                       ..
                      (<xn_coord>, <yn_coord>)
               }
               COLOUR <r> <g> <b>
               COMPONENT { OBJECT <Child_Name1> {
                             FORM {
                                    TYPE <Child Type>
                                    POINTS < point list >
                                   }
                             COLOUR <r> <g> <b>
                           }

                           OBJECT <Child_Name2> {
                               FORM {
                                      TYPE <Child Type>
                                      POINTS < point list >
                                     }
                               COLOUR <r> <g> <b>
                           ..
                           OBJECT <Child_Name n> {
                               FORM {
                                      TYPE <Child Type>
                                      POINTS < point list >
                                     }
                               COLOUR <r> <g> <b>
                          }

                       }
                  }
```

Figure 14: Group definition

The produced VRML PROTO node is the following:

```
#VRML V2.0 utf8
PROTO <Child_Name1> [ ]
{
      <creating the PROTO according to the object type>
} #end of child1 prototype
```

```
PROTO <Child_Name2> [ ]
{
     <creating the PROTO according to the object type>

} #end of child2 prototype
..
PROTO <Child_Name n> [ ]
{
     <creating the PROTO according to the object type>

} #end of child n prototype

PROTO <Name> [ ]
{
      Group {
                children [
                          Transform {
                               translation
                                  <x1_coord>, <y1_coord> 0
                               children <Child_Name1> {}
                               }
                          Transform {
                                  <x2_coord>, <y2_coord> 0
                               children <Child_Name2>{}
                               }
                           ..

                          Transform {
                                  <xn_coord>, <yn_coord> 0
                               children <Child_Name n>{}
                           }

                         ]}
}
```

Figure 15: Code produced by Translator to create a group of objects

The translator logic is to create different PROTO for each child and then use these PROTO to define the group and position of the children inside the group.

4.3 3.2 Producing code for events

Defining objects is the first step to create an Interactive Animation. Next step is to give them a behavior by translating the user given ECA rules. To trigger this behavior we have

to create events. Start event is usually used to position objects in the IAN space. Mouse click event is also implemented and for instance is the only event that can trigger a sequence of actions. Start event can be declared as:

```
EVENT START
```

in the scenario definition and is used only to mark the beginning of the Interactive Animation.

Mouse click event is defined as:

EVENT *<Object_Name>*.MOUSE_CLICK

To implement mouse click translator creates a touchSensor node giving a definition to it so that it will be easier to create various routes afterwards. The produce node has the following format.

```
DEF TOUCH<Object_Name> TouchSensor { enabled TRUE }
```

4.3 3.3 Producing code for actions

Going to the most interesting action part now, we start by describing the start action. Start action is even the most important as it positions the already defined objects in the Interactive Animation space. We must notice that for the moment we use the VRML space as the Interactive animation space. Start action for a specific object can be described in the Interactive animation as

```
<Object_Name>.START((<x_position>, <y_position>), TRUE)
```

For the instance we suppose that visibility in start action is always TRUE. For Start action Translator uses the PROTO definition and creates a Transform node translating object to the position defined. The produced node using parameters is presented below:

```
DEF <Object_Name> Transform {
  translation <x_position> <y_position> 0
    children [<Object_Name> { } ]
 }
```

Figure 16: Start action

Notice that the object name refers to the PROTO that has already been in the object definition part.

Discrete Translation can be defined in the Interactive Animation as:

```
<Object_Name>.TRANSLATION((<x_coord>, <y_coord>))
```

The VRML code created for discrete translation uses a structure similar to the continuous translation. Translator creates a Position Interpolator node, and a Time Sensor. Supposing that the discrete translation is triggered by a click event, Translator creates the routes appropriate in order to trigger translation. Produced code is presented below:

```
DEF POS<Object_Name> PositionInterpolator {
  key [ 0 ]
  keyValue [<x_coord> <y_coord> 0 ] }

DEF TIME<Object_Name> TimeSensor { }

ROUTE TOUCH<Object_Name>.touchTime TO
      TIME<Object_Name>.startTime
ROUTE TIME<Object_Name>.fraction_changed TO
      POS<Object_Name>.set_fraction
ROUTE POS<Object_Name>.value_changed
      TO <Object_Name>.set_translation
```

Figure 17: Discrete Translation Action

In the same way we can build discrete rotation and discrete scaling. Continuous transformations use the law given by the user in order to compute the various object positions. Continuous Translation can be defined as:

<Object_Name>.C_TRANSLATION(<*x_time_expression*>, <*y_time_expression*>)

The produced code for continuous translation is presented below:

```
DEF POS<Object_Name> PositionInterpolator {

    key [ 0 1 2 3 4 5 6 7 8 9 ]

    <we decide to use by default ten values for the time>

    keyValue [ calculating x and y using
        x_time_expression and y_time_expression ] }

DEF TIMERect TimeSensor { cycleInterval 3.0 }
< we decide to use 3.0 as default cycleInterval >


ROUTE TOUCH<Object_Name>.touchTime TO
      TIME<Object_Name>.startTime
ROUTE TIME<Object_Name>.fraction_changed
      TO POS<Object_Name>.set_fraction
ROUTE POS<Object_Name>.value_changed
```

```
      TO <Object_Name>.set_translation
```

Figure 18: Continuous Translation Action

Continuous Rotation uses the orientation interpolator in order to compute the various positions. We define continuous rotation as:

```
<Object_Name>.C_ROTATION(<angle_time_expression>)
```

The code produced by Translator is next presented:

```
DEF OI<Object_Name> OrientationInterpolator {
 key [ 0.0, 0.1, 0.3, 0.6, 0.8, 1.0 ]
 < we decide to use these key as default values >
 keyValue [ 0 0 1 <value_produced_by_time_expression>,
            0 0 1 <value_produced_by_time_expression>,
            0 0 1 <value_produced_by_time_expression>,
            0 0 1 <value_produced_by_time_expression>,
            0 0 1 <value_produced_by_time_expression>,
            0 0 1 <value_produced_by_time_expression> ] }

 DEF TIMERect TimeSensor {
 cycleInterval 3.0 }

ROUTE TOUCH<Object_Name>.touchTime TO
      TIME<Object_Name>.startTime
ROUTE TIME<Object_Name>.fraction_changed
      TO OI<Object_Name>.set_fraction
ROUTE OI<Object_Name>.value_changed
      TO <Object_Name>.rotation
```

Figure 19: Continuous Rotation action

Continuous Scaling uses the Position Interpolator to produces values but we must give the right parameters so that the produced values can calculate an accurate scaling.

4.3 3.4 General Scenario examples

To define a scenario, Translator must be able to combine separate node definitions according to the user instructions. Let's put them all together and present some simple or more complicated scenario examples.

1. Define a rectangle of size (1, 4) and of red colour. Define its position at 3, 2 point. When user mouse-clicks on this object, rectangle must be translated at a new position (2, 1).

Scenario must be defined as:

```
OBJECT Rect {
                FORM {
                        TYPE RECTANGLE
                        POINTS (1, 4)
                      }
                COLOUR 1 0 0
               }
EVENT START
ACTIONS SEQ(Rect.START((3, 2), TRUE) 0)
EVENT Rect.MOUSE_CLICK
ACTIONS SEQ(Rect.TRANSLATION(( 2 , 1)) 0 )
```

The produced code combines the definitions given above:

```
#VRML V2.0 utf8
PROTO Rect[ ]
{
                Shape {
                 geometry Box { size 1 4 0.0000001 }
                            appearance Appearance {
                            material Material {
                      diffuseColor   1 0 0 }
                }
         }
} #end of RECTANGLE prototype

DEF Rect Transform {
  translation 3 2 0
    children [ Rect{ } ]
 }

 DEF TOUCHRect TouchSensor { enabled TRUE }
 DEF POSRect PositionInterpolator {
 key [ 0 ]
 keyValue [  2 1 0 ] }
 DEF TIMERect TimeSensor { }

ROUTE TOUCHRect.touchTime TO TIMERect.startTime
ROUTE TIMERect.fraction_changed TO POSRect.set_fraction
ROUTE POSRect.value_changed TO Rect.set_translation
```

The visual result makes things more clear. In the first figure below we can see the starting position and the form of the object at starting time, while in the second we can see the position of the rectangle after mouse click and discrete translation action. Rectangle is translated from point 3,2 to 2,1 as user defines.

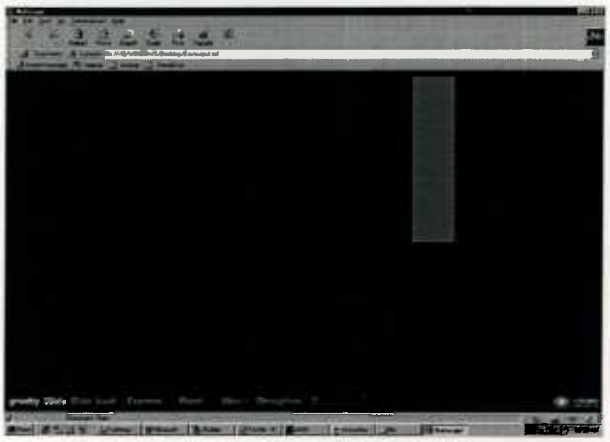


Figure 20: Starting Position of the rectangle

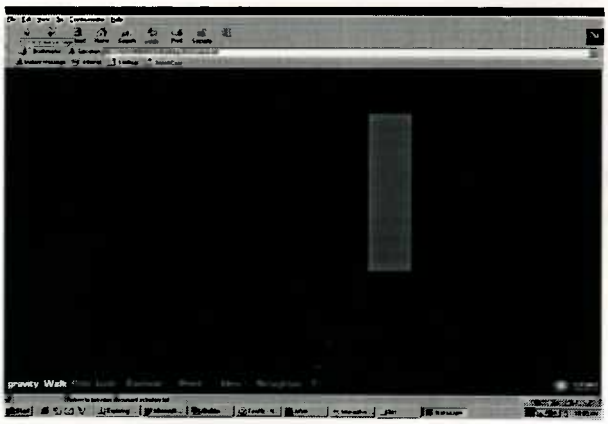


Figure 21: Rectangle Position after user interaction

Another scenario example can be:

2. Define two rectangles of size 1, 4 and 1,3 and of red and green colour each one, position them at point 3, 2 and 1,1 and then when the user mouse click translate them continuously according the laws defined.

Scenario Definition is the following:

```
OBJECT Rect {
                FORM {
```

```
                    TYPE RECTANGLE
                    POINTS (1, 4)
                  }
            COLOUR 1 0 0
                    }
OBJECT Rect2 {
             FORM {
                    TYPE RECTANGLE
                    POINTS (1, 3)
                  }
              COLOUR 0 1 0
            }
EVENT START
ACTIONS SEQ(Rect.START((3, 2), TRUE) 0
          Rect2.START((1, 1), TRUE) 0)
EVENT Rect.MOUSE_CLICK
ACTIONS SEQ( Rect.C_TRANSLATION( 2 TIME + 5, 1 TIME + 3 ) 0
           Rect2.C_TRANSLATION(2 TIME + 5, 1 TIME + 3) 0)
```

Translator produces the following code:

```
#VRML V2.0 utf8
PROTO Rect[ ]
{
              Shape {
               geometry Box { size 1 4 0.0000001 }
                          appearance Appearance {
                          material Material {
                     diffuseColor   1 0 0 }
              }
        }
} #end of RECTANGLE prototype

PROTO Rect2[ ]
{
              Shape {
               geometry Box { size 1 3 0.0000001 }
                          appearance Appearance {
                          material Material {
                     diffuseColor   0 1 0 }
              }
        }
} #end of RECTANGLE prototype

DEF Rect Transform {
  translation 3 2 0
    children [ Rect{ } ]
```

```
 }
DEF Rect2 Transform {
  translation 1 1 0
    children [ Rect2{ } ]
 }

 DEF TOUCHRect TouchSensor { enabled TRUE }


 DEF POSRect PositionInterpolator {
 key [   0 1 2 3 4 5 6 7 8 9 ]

 keyValue [  5 3 0,7 4 0,9 5 0,11 6 0,13 7 0,15 8 0,17 9
             0,19 10 0,21 11 0,23 12 0, ] }

 DEF TIMERect TimeSensor {
 cycleInterval 3.0 }

ROUTE TOUCHRect.touchTime TO TIMERect.startTime
ROUTE TIMERect.fraction_changed TO POSRect.set_fraction
ROUTE POSRect.value_changed TO Rect.set_translation

DEF POSRect2 PositionInterpolator {
key [   0 1 2 3 4 5 6 7 8 9 ]

keyValue [  1 0 0,2 2 0,3 4 0,4 6 0,5 8 0,6 10 0,7 12 0,8
            14 0,9 16 0,10 18 0, ] }

DEF TIMERect2 TimeSensor {
cycleInterval 3.0 }

ROUTE TOUCHRect.touchTime TO TIMERect.startTime
ROUTE TIMERect.fraction_changed TO POSRect2.set_fraction
ROUTE POSRect2.value_changed TO Rect2.set_translation
```

To be more clear let's see some instances of the scenario execution.

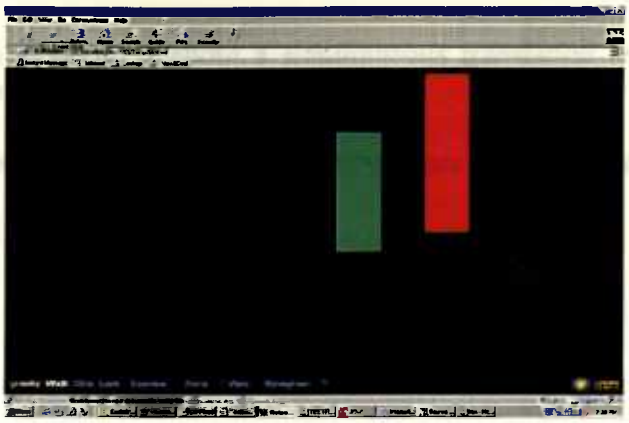


Figure 22: Example 2, objects at starting time

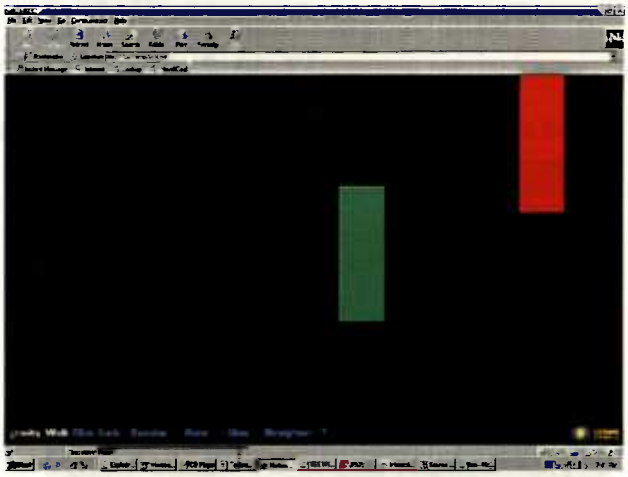


Figure 23: Example 2 Objects moving after mouse click

Figure 24: Example 2 Another instance of object's motion

The last scenario example is about a simultaneous translation and rotation. The visual result is really very impressive.

3. Create a rectangle and after mouse click translate and rotate it at the same time.

```
OBJECT Rect {
                FORM {
                        TYPE RECTANGLE
                        POINTS (1, 4)
                      }
                 COLOUR 1 0 0
               }
EVENT START
ACTIONS SEQ(Rect.START((3, 2), TRUE) 0)
EVENT Rect.MOUSE_CLICK
ACTIONS SEQ(   Rect.C_TRANSLATION( 2 TIME + 5, 1 TIME + 3 ) 0
Rect.C_ROTATION( 2 TIME + 2) 0)
```

Various instance of the visual result are given in the figures below:

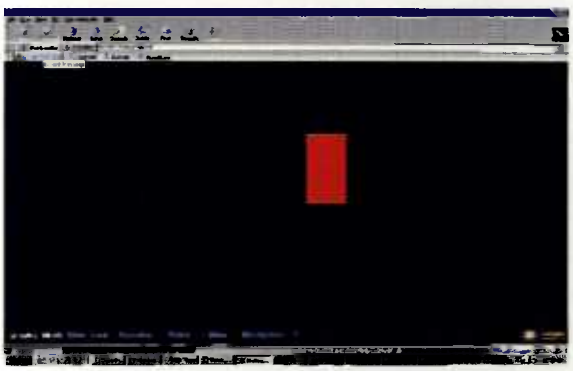


Figure 25: Example 3, rectangle starting position

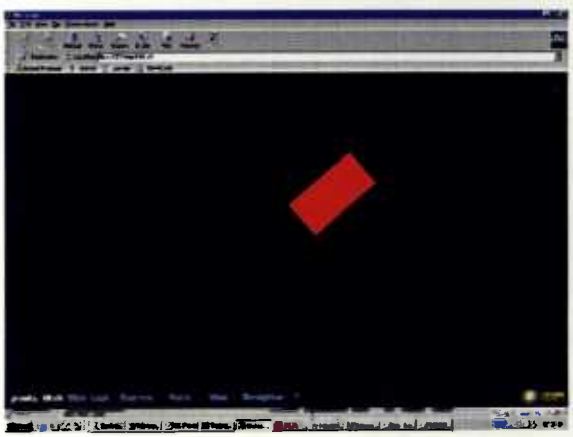


Figure 26: Example 3, Instances of the continuous translation and rotation

4.4 Synopsis

In this chapter we have presented the translator architecture. We have given a general description of the basic translator's modules: user interface, lexical analyzer and parser. We have presented parser in more detail, as it is the most important part of the system. We have explained how translation works and what is the produced VRML code in case of object creation, mouse_click events, and transformations continuous and discrete. We have also given some examples of scenario execution that put things all together.

5. Interactive Animations and databases

Contents

5.1 Introduction

The authoring environment presented in chapter four creates Interactive Animations. The user gives a description according to the grammar specification and can render the result on a browser. And of course he has the ability to create a large number of Interactive Animations related or no to a specific subject. The challenging question is how we can manage these Interactive Animations and how we can extract further information, combining them. It is very important to explore what kind of information users want to extract from Interactive Animations, and to decide what kind of data can produce this information. The only answer to the previous questions is the connection of Interactive Animations with databases. In this chapter we try to do this connection by creating at first a data model that better fits Interactive Animations. Then upon this data model we explore what kind of queries are valuable to users and we try to classify and express them. In the conceptual level we take into account some critical parameters as ***our grammar specification for authoring environment*** and the ***theory of spatiotemporal databases.*** The whole procedure is implemented using the basic principles of object oriented data model and OQL (object query language) to express queries on the created database model. Next paragraph presents in more detail the methodology that we are going to use in order to create the database schema.

5.2 Methodology

The process of designing a database always begins with an analysis of what information the database must hold and the relationships among components of that information. The result of this phase is the ***database schema.*** We can use various ways to express this design. The traditional approach is the well-known Entity-Relational Model. But as object oriented analysis and design become more and more powerful and facilitate the real world modeling, an object-oriented approach is proposed to serve this goal. This approach is called ODL (Object definition language) and is a standard language for specifying the structure of databases in object-oriented terms.

After analysis and design phase database schema must take a physical existence and we must have the ability to do queries upon this schema. In order to do this we need to use a real query language. To translate the Entity Relational model we use the famous SQL (structured query language) standard. For object oriented database design we have to use

another standard called OQL (object query language), a powerful and easy to use SQL like query language with special features for dealing with complex objects, values and methods.

In this dissertation we have decided to use the object oriented approach. After a first consideration of our problem we see that we have to deal with more complicated information as continuous and time dependent data, spatial and temporal relationships or user interaction. The relational database model seems to be less powerful in this occasion in comparison to the object oriented design, which is finally adopted in this dissertation. In the next paragraphs we are going to follow a specific procedure according to the object oriented model, as figure below shows.

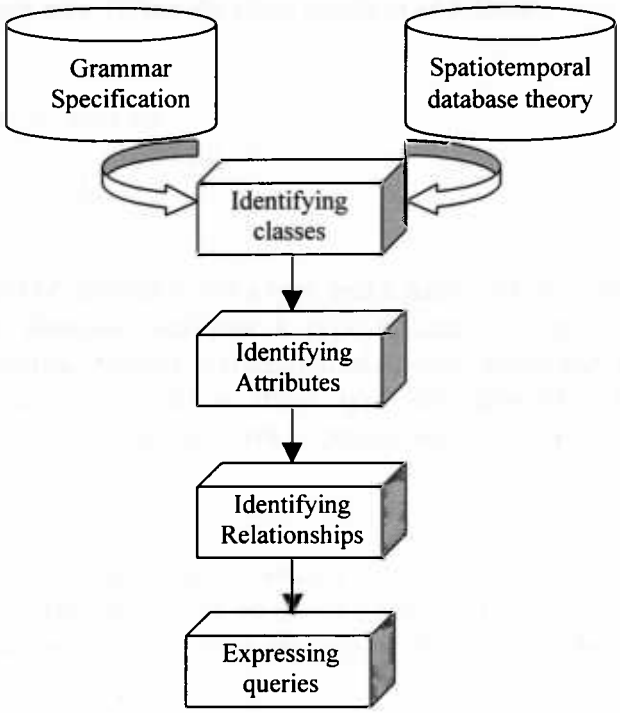


Figure 27: Object oriented database and queries design

The input in this process is the grammar specification described in chapter three and some ideas from the spatiotemporal database theory [Guting and al. 1998]. The first step is to identify the classes exactly as we do in the object oriented design. Each class has some attributes that characterise the objects belonging to it. Classes are connected according to some relationships. After classes definition, the last and probably the most important step in the whole process is to identify the queries that can be expressed using these classes. Next paragraph introduces classes at first.

5.3 Classes attributes and relationships

To express classes and their attributes we will use the ODL-like notation. A class can be written as:

```
Class <name> {
   < list of properties >
}
```

The list of properties contains tree different kinds of elements: attributes, relationships and methods. So we can now refine the class notation as follow:

```
Class <name> {
   < list of attributes >
   < list of relationships >
   < list of methods >
}
```

Each element in the list of attributes has a type and a name. About types we must mention that ODL offers the database designer a type system similar to that found in the conventional programming languages consisted of ***atomic***, ***interface*** types and some type ***constructors.*** (For more information about type see appendix B ODL and OQL reference). We will use the following form to present attributes of each object:

```
attribute type < attribute name >
```

Relationships show the way an object connects to other objects in the same or another class. They have also a type which can be either a reference to an object of some class or a collection (a set as an example) of such references. We can use the following format to present them:

```
Relationship Type < Class_Name > Objects_Name
```

Classes structure is next presented.

Class IAN

Description: This class is the description of each IAN created according to the IAN grammar specification. It has a name that briefly describes its subject. It has been written by a user, so-called author and at a specific creation date. An IAN according to the grammar specification is consisted of objects and statements of the type event condition action. So we can distinguish the IAN relations with a set of objects and a set of statements.

Notation:

```
Class IAN {
   attribute string IAName;
   attribute string author;
   attribute Struct Date
   {integer day, integer month, integer year} creationDate;
   relationship Set<IANobject> IANobjects;
   relationship List<statement> IANstatements;
}
```

Class IANobject

Description: Class IANobject is defined to describe objects that participate in an Interactive Animation. Every object of the class must have as attributes the name, its type, a point list that define its form, the filename of its contents and its colour. In addition every object in the list must be declared in one and only one IAN. At last if an IANobject is of type GROUP that means that it has also relations with other children-objects. To help the querying part we put also a boolean attribute called isParent in order to find easily if an object is a parent or not.

Notation:

```
Class IANobject {
   attribute string objectName;
   attribute string type;
   attribute List < Struct Point { float coord1, float
coord2 } >
             pointList;
   attribute string contents;
   attribute Struct Colour {integer r, integer g,
                integer b} objectColour;
   attribute Boolean isParent;
   relationship IAN declaredIn;
   relationship Set<IANobject> children;
   relationship Set<event> eventsParticipatingIn
   relationship Set<condition> conditionsParticipatingIn
   relationship Set<action> actionsParticipatingIn
}
```

Class IANstatement

Description: Class IANstatement has no attributes. But the main role of this class is to connect IAN with events, conditions and actions. Each statement belongs to one and only

one IAN. On the other hand a statement may be built by an event, a condition and an action (a sequence of actions). These relations are expressed using the following notation.

Notation:

```
Class IANstatement {
   relationship IAN declaredIn;
   relationship Event event
   relationship Condition condition;
   relationship Action action;
}
```

Class Event

Description: Event class describe an event that triggers an action if the specified condition is at the same time fulfilled. According to the specification, an event participates in one and only one statement and can be related with zero, one or two at maximum objects (zero when the event is not connected to a specific object but to the IAN in general, one in all events of type objectName.event and two only in the case of the topological relations that we have decided to put in the model (MEET and NO_MEET)). The main attributes of this class are the event type and its value when the event is of type event == expression. Accepted types for type attribute are according to the grammar specification:

- COLOUR_EXPRESSION
- CONTENTS_EXPRESSION
- VISIBILITY_EXPRESSION
- ACTIVITY_STATUS_EXPRESSION
- INTERNAL_TIME_EXPRESSION
- POSITION_EXPRESSION
- ANGLE_EXPRESSION
- SCALE_EXPRESSION
- MOUSE_CLICK
- KEYBOARD_PRESS
- MEET
- NO_MEET
- START
- GLOBAL_TIME

Event class contains also the inverse relationship with statement and a relationship with the objects with which can be related.

Notation:

```
Class Event {
```

```
    attribute string simpleEventType;
    attribute string equationValue;
    relationship Set<IANobject> relatedTo;
    relationship statement participatesIn;
}
```

Class ComplexEvent

Description: Complex event is actually a composition of events. Each object in complex event class has also a type. Complex events can have one of the following types:

- ANY
- SEQ
- TIMES

Another important attribute is the count, which has a different meaning for each type of complex event. For example with TIMES, count has the meaning of how many times an event occurs while for ANY means the occurrence of a specific number (count) of events. For SEQ complex events count has no meaning and its value is undefined. At last a complex event is related with a set of events and of course to one and only one statement. The notation can be expressed as follow:

Notation:

```
Class complexEvent {
    attribute string complexEventType;
    attribute integer count;
    relationship Set<Event> relatedTo;
    relationship Statement participatingIn;
}
```

Class Condition

Description: Class condition has a structure similar to that of event. Condition has a type attribute and an attribute expression value for each expression. Note that expression in condition can be not only an equation but also an expression using comparison operators (<, >,<=,>= etc.). As a result of this difference we need the operator attribute also in order to specify the precise kind of expression. In addition condition class is related to zero (when it is a time condition), one (in case of state condition) and two objects (in case of inter object condition). It is also related to one and only one statement where it is declared in. The accepted values for condition type attribute are:

- COLOUR_EXPRESSION
- CONTENTS_EXPRESSION
- VISIBILITY_EXPRESSION
- ACTIVITY_STATUS_EXPRESSION
- INTERNAL_TIME_EXPRESSION

- POSITION_EXPRESSION
- ANGLE_EXPRESSION
- SCALE_EXPRESSION
- TIME_EXPRESSION
- MEET_EXPRESSION
- NO_MEET_EXPRESSION

Notation:

```
Class Condition {
   attribute string conditionType;
   attribute string expressionValue;
   attribute string operator;
   relationship Set<IANobject> relatedTo;
   relationship statement participatesIn;
}
```

Class ComplexCondition

Description: Class complex condition describes a composition of simple conditions using logical operators (OR, AND). Therefore it contains two attributes: A list with the logical operators that are used in the complex condition and a list with the conditions that build the complex condition.

Notation:

```
Class Condition {
   attribute List<string> logicalOperators;
   attribute List<condition> listOfConditions;
}
```

Class Action

Description: Action class has as main attribute, a list of structures of type (time_interval, simple_action) according to the grammar specification. Time interval is an atomic value that describes the time between two simple actions in the sequence of actions. Simple action is an object of the class SimpleAction that is described afterwards. This class has also and some relationships. It is related to one and only one statement and to one and only one event that triggers this action.

Notation:

```
Class Action {
   List <Struct ActionElement{integer timeInterval,
```

```
                simpleAction Lsimplestring}>
    relationship statement participatesIn;
    relationship event relatedTo;
}
```

Class simpleAction

Description: Every sequence in action part of an ECA rule is consisted of time intervals and simple actions. This class describes exactly these simple actions. The main attribute in this class is the actionType that help us distinguish between the different types of actions. Each simple action has different number and semantic of parameters. Consequently there is a need to create ***subclasses***. To further explain, often a class contains certain objects that have special properties not associated with all members of the class. So it is useful to organize the class into subclasses, each subclass having its own special attributes and/or relationships in addition to those of the class as a whole. In ODL we can declare a subclass according to the following format.

```
Class < Class_Name> : <class from which it inherits> {
    < additional properties>
}
```

Action subclasses are: startAction, discreteTranslation, discreteRotation, discreteScaling, discreteColorChanging and ContinuousTransformations. Subclass ContinuousTransformations contains also another level of subclasses as each kind of continuous transformation has some extra actions related to them. ContinuousTransformationsActions subclasses are ContinuousTranslation, ContinuousRotation, ContinuousScaling, ContinuousColorChanging. We are obliged to have a different class for each action as each action has completely different parameters and of course it has completely different conceptual meaning. SimpleAction type may have one of the following values.

- START
- STOP
- RESUME
- PAUSE
- HIDE
- SHOW
- ADD_MEMBER
- REMOVE_MEMBER
- DESTROY
- DISCRETE_TRANSLATION
- DISCRETE_ROTATION
- DISCRETE_SCALING
- CONTINUOUS_TRANSLATION
- CONTINUOUS_ROTATION
- CONTINUOUS_SCALING

- CONTINUOUS_COLOR CHANGING

At last we must mention that every simpleAction is connected to an object and this is a very important relationship of the class.

Notation:

```
Class simpleAction {
   string actionType
   relationship Action participatesIn;
   relationship IANobject relatedTo;
}
```

Class startAction

Description: The first subclass of simpleAction is StartAction. StartAction is the class that fully describes all information about object initiation and starting position. Attributes of this class are position value for x-axis, position value for y axis and visibility value. The relation with a specific object is inherited by the class Simple Action.

Notation:

```
Class startAction : simpleAction{
   float positionX;
   float positionY;
   boolean visibilityValue;
}
```

Class discreteTranslation

Description: Class discreteTranslation describe a discrete translation of an object. To fully describe the discrete Translation we need only the two values for position in axe x and position in axe y.

Notation:

```
Class discreteTranslation : simpleAction{
   float discretePositionX;
   float discretePositionY;
}
```

Class discreteRotation

Description: Class discreteRotation describe a discreteRotation of an object. DiscreteRotation is completely described by the attribute angleValue.

Notation:

```
Class discreteRotation : simpleAction{
   float discreteAngleValue;
}
```

Class discreteScaling

Description: Class discreteScaling has as attributes the scalingValue for x axe and the scalingValue for y axe. The notation is as follow:

Notation:

```
Class discreteScaling: simpleAction{
   float discreteScalingXValue;
   float discreteScalingYValue; }
```

Class discreteColorChanging

Description: Class discreteColorChanging is described by the three attributes that indicates the new value for colour parameters r, g, b.

Notation:

```
Class discreteColorChanging: simpleAction{
   float discreteColorRValue;
   float discreteColorGValue;
   float discreteColorBValue;
}
```

Class continuousTransformationActions

Description: This class has as subclasses the continuousTranslation, continuousRotation, continuousScaling and continuousColorChanging. The two main attributes that are inherited in all subclasses, are the continuous transformation name and the TransformationActionsType. Transformation Actions Type may have one of the following values.

- RESUME
- PAUSE
- STOP
- CONTINUOUS_TRANSLATION
- CONTINUOUS_ROTATION
- CONTINUOUS_SCALING
- CONTINUOUS_COLOR_CHANGING

Notation:

```
Class continuousTransformationActions : simpleAction{
   Attribute String translationName;
   Attribute String TransformationActionsType;
}
```

Class continuousTranslation

Description: Continuous transformations have the same logic as discrete transformations but they use the law to compute position. For continuous translation we have two different laws for each axe.

Notation:

```
Class continuousTranslation :
continuousTransformationActions {
   String PositionXLaw;
   String PositionYLaw;
}
```

Class continuousRotation

Description: For continuousRotation we need as an attribute only the law for the changing in angle value. The notation is as follow:

Notation:

```
Class continuousRotation: continuousTransformationActions {
   String angleChangingLaw;
}
```

Class continuousScaling

Description: As with the previous classes continuousScaling has two attributes that describe the law changing for each axe.

Notation:

```
Class continuousScaling : continuousTransformationActions {
   String scalingXChangingLaw;
   String scalingYChangingLaw;
}
```

Class continuousColorChanging

Description: Class continuousColorChanging contains as attributes the laws for the changing of each colour.

Notation:

```
Class continuousColorChanging :
continuousTransformationActions {
   String colourRChangingLaw;
   String colourGChangingLaw;
   String colourBChangingLaw;}
```

Next paragraph summarizes the data model using a graphical representation of classes and subclasses.

5.4 Data Model graphical representation

Data model will be the reference during the queries building process. A graphical representation of the schema may be very useful for this purpose. The first figure express the main classes hierarchy while the other shows the simple action subclasses.

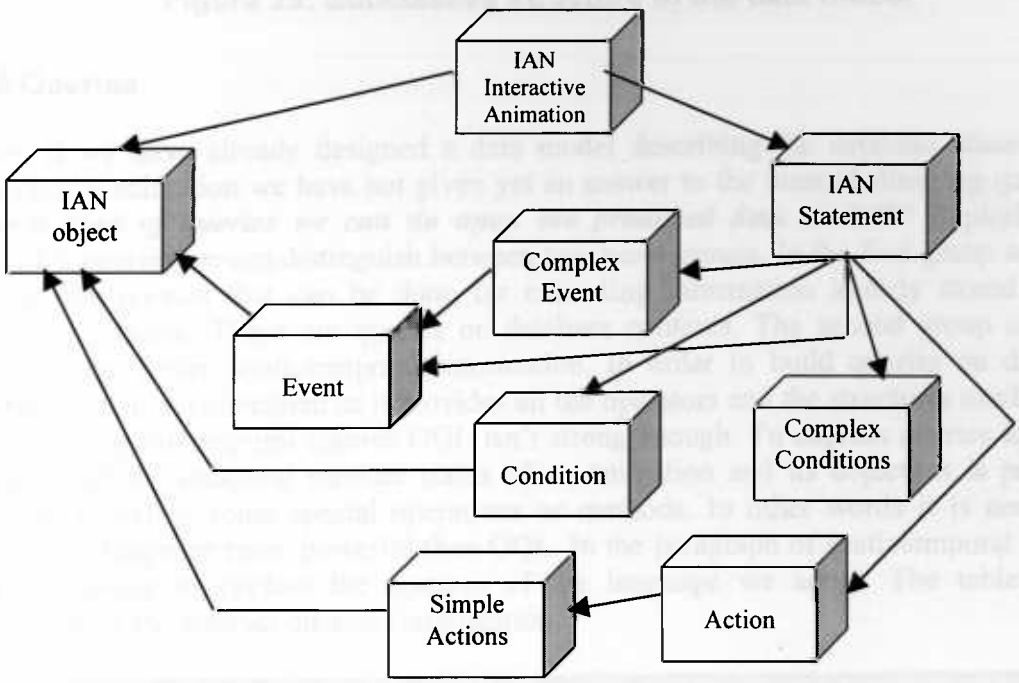


Figure 28: Main classes in data model

A graphical representation of subclasses follows.

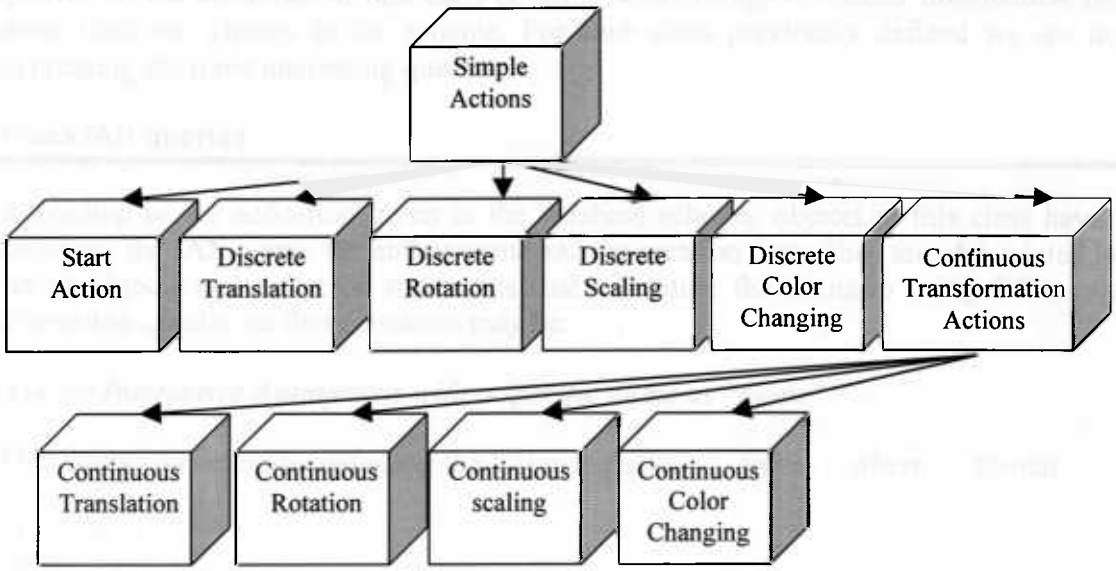


Figure 29: Subclasses structure in the data model

5.5 Queries

Even if we have already designed a data model describing the database status of an interactive animation we have not given yet an answer to the most challenging question: ***"what kind of queries we can do upon the produced data model?"*** Exploring the possible queries we can distinguish between two basic groups. In the first group we must define the queries that can be done for extracting information already stored in the database schema. These are queries on database contents. The second group contains queries that return spatiotemporal information. In order to build queries on database contents OQL is convenient as it provides all the operators and the structures needed. But in case of spatiotemporal queries OQL isn't strong enough. To express queries upon the spatial and the temporal runtime status of an animation and its objects it is probably needed to define some special operations or methods. In other words it is needed to provide a language more powerful than OQL. In the paragraph of spatiotemporal queries we are going to explain the features of the language we adopt. The table below summarizes the abstract queries classification.

Queries classification
▪ Queries on database contents
▪ Spatiotemporal Queries

Table 5: Queries abstract classification

5.5.1 Queries on database contents

We can extract various informations from the database contents either building simple queries on the attributes of one class or using relationships to extract information from more than one classes in the schema. For each class previously defined we are now expressing the most interesting queries.

Class IAN queries

According to the definition given in the database schema, objects in this class have as attributes the IAN name, the author name and the creation date. They are also related to a set of objects and a list of statements that constitute the scenario using ECA rules. Interesting queries on these contents may be:

Ask for Interactive Animations with a specific name as "name"

This query can be expressed using the following select … from … where … format

```
SELECT i
FROM IAN i
WHERE i.IAName = 'name'
```

Ask for Interactive Animations written by an author named "author_name"

This simple query can be expressed as

```
SELECT i
FROM IAN i
WHERE i.author = 'author_name'
```

Ask for Interactive Animations written before 1/3/1999 as an example.

This query can be expressed as

```
SELECT i
FROM IAN i
WHERE i.creationDate <= '1/3/1999'
```

The result of each of the previous queries is a group of objects that satisfy the defined condition in the where part. We can also create queries that return only attribute values (only name as an example). A last important remark is that we can express more sophisticated conditions using various comparison and logical operators. According now to the specified relationships we can build queries that combine attributes of more than one classes. Examples of this kind of queries are:

Ask for objects participating to an Interactive Animation with name "name".

To give answer to this query we have to extract the objects related with the IAN. A complex query that can do that will be the following.

```
SELECT o
```

```
FROM IAN i, i.IANobjects  o
WHERE i.IAName = 'name'
```

Ask for statements that build an Interactive Animation called "name".

By this query the user can extract the scenario part of an Interactive animation.

```
SELECT st
FROM IAN i, i.statements  st
WHERE i.IAName = 'name'
```

Based on these relationships we can create more sophisticated queries creating conditions and combining attributes not only from the IAN class but from the objects class also (class statements doesn't contains any attribute). We can extract as an example all the objects that are related to a specific Interactive Animation and their type is rectangle.

Class IANobject queries

From this class we can extract useful information about the type, the size, the colour or the contents of an object. These queries can be build exactly as in the IAN case. A simple example can be to select all objects having RECTANGLE as a type. This query can be expressed as:

```
SELECT o
FROM IANobject o
WHERE o.type = 'RECTANGLE'
```

Figure 30: A select example for IANobject class

All the possible queries for object attributes are:

Queries on object attributes
1. Ask for an object with a specific object name
2. Ask the type of an object with a specific name
3. Ask the point list of an object with a specific name
4. Ask the contents of an object with a specific name
5. Ask the colour of an object with a specific name
6. Ask if a specific object is parent or not

Table 6: Queries on object attributes

Using relationships in class IANobject we can extract more information related to objects from other classes. Useful queries can be:

Queries based on the IANobject relationships
1. Ask for the attributes of the IAN in which an object is declared.
2. Ask for attributes of the children of an object
3. Ask for the events an object with a specific name declared in a specific IAN is related to

4. Ask for the actions that define the behaviour of an object in the specific IAN.

Table 7: Queries based on the IANobject relationships

We must remark that relationship with the condition is defined in the IAN object class in order for the data schema to be consistent. But there isn't any interesting question related to objects and conditions to pose. On the other hand is very interesting to explore the kind of events, an object participates in and of course the kind of actions.

We can also build queries that return as result a group of objects with common characteristics.

Queries on objects with common characteristics
1. Ask for objects in an Interactive Animation that have the red colour
2. Ask for objects in an Interactive Animation that have a specific type (ex. rectangle)
3. Ask for Interactive Animations with objects of type GROUP

Table 8: Queries on objects with common characteristics

Queries on objects and IAN extract the most important information. Other classes with attributes on which we can build interesting queries are:

Class Event queries

An important query on the objects of the event class is:
Ask for the objects that are related to a specific event. This is the only query that has interest on this class.

Class Action queries

The interesting queries in Action class are presented below.

Queries on Action class objects
1. Ask for the event that triggers this action
2. Ask for the objects participating in this sequence of actions.

Table 9: Queries on Action class objects

Class simpleAction queries and all subclasses

In this group of classes there is a lot of attributes but these attributes are more useful in order to build Spatiotemporal Queries. There isn't sense for example to extract the law motion of a transformation but it is very important to use this law in order to compute the position of an object at a specific time. Interesting queries based on these attributes will be presented in the next paragraph.

5.5.2 Spatiotemporal Queries

5.5.2.1 Language definition

Queries presented in the previous paragraph are static that means that they are interested only to the contents of the database. This is of course very important as we can extract useful information about Interactive Animations. But it would be more useful to be able to query on the runtime behavior of the objects as we have already mentioned above. To do that we must in a way simulate the behavior of the objects participating in interactive animations. This would be an easy task if the time was continuous and if there weren't interaction. Interaction creates a lot of problems, as it is impossible to know the user behavior, which is different in every Interactive Animation execution. But even under these conditions it is very interesting to explore the possible queries that we can express in the runtime behavior of objects in an Interactive Animation.

From a general perspective of the problem, we can see that standard OQL doesn't provide appropriate structures and operators to extract dynamic information. In order to express such queries we need to describe a new language based on OQL but more powerful and spatiotemporal oriented. Three issues must be under consideration for this purpose:

- The need for new spatial and/or temporal operators
- The way to embed run-time attributes of animated objects in the language.
- The accuracy of the traditional query structure SELECT.. FROM ..WHERE ..

Let's explore one by one the issues above.
To express spatiotemporal queries we need spatiotemporal operators also. Standard OQL can't check if two object are meeting, are disjoint etc. We need to construct special operators for this purpose. In the theory of spatiotemporal databases we can find a lot of new operators that can produce spatiotemporal expressions. In case of Interactive Animations we select the operators that have a sense and can be used to return useful information. These are more spatial and can be very powerful when we combine them with the run-time attributes operators defined in the next paragraph. The following table summarizes and define them:

Operator	Description
Meet (object1, object2)	Meet is a spatial operator that explores if two objects are meeting, that means if two objects have common borders.
Unmeet (object1, object2)	Unmeet is a spatial operator that explores if two objects are not meeting that means if two objects don't have common points
Overlap(object1, object2)	Overlap is also a spatial operator and investigates not only if two objects have common borders but common areas in general.
Distance (object1, object2)	Distance computes the distance between two objects.

	According to [Guting and al. 1998] the distance function determines the minimum distance between the closest pair of points where the first element is from the first argument and the second element from the second argument.

Table 10: Operators for building spatiotemporal queries.

The second issue, about how to treat runtime attributes is also very important. The query language must be able to treat sufficiently all the attributes that can be changed during animation. These attributes are grouped in the table below:

Attributes changing during Interactive Animation
Object activity status
Color
Position
Angle
Scaling
Object internal time
Animation time

Table 11: Attributes changing during Interactive Animation

Some of these attributes may have a value stored in the database while other are defined only during interactive animation execution. It is obvious that an expression as o.colour has a different meaning when the attribute refers to the run time. In fact the expression o.colour returns the value of the attribute color of an object, at starting time (to be more precise it returns the database value) and not the color of an object during the interactive animation execution. So we have to use another notation for these attributes. To express this difference a notation like:

```
runtime_attribute_of(object)
```

is the most appropriate. According to this concept we have the following operators that compute the value of an attribute during execution and returns its value. We have:

`activityStatusOf(object)` that returns the object activity status during execution
`colourOf(object)` that returns the color of an object during execution
`positionOf(object)` that returns the position of an object during execution
`angleOf(object)` that returns the angle of an object during execution
`scalingOf(object)` that returns the scaling of an object during execution
`internalTimeOf(object)` that returns the internal time of an object during an animation
`time(interactive_animation)`: This attribute refers to the animation time and receives as parameter the specific animation.

We can now use all these attributes exactly as the static one (colourOf(o1) = red as an example) but the meaning of these expressions are much different as we have already explained. Notice that we don't define runtime operators for attributes as the internal time or the activity status of a transformation, as we consider that there isn't any interesting query to do using these runtime attributes. As an example, it is important to be able to extract information about the activity status of an object but it isn't as important to know the activity status of one of the transformations of an object.

We are now ready to use the operators previously defined, in a query statement. We could use the "traditional" `SELECT .. FROM .. WHERE..` query structure for this purpose, but we believe that another structure which help us to distinguish between time expressions and conditions, is more sufficient and appropriate. To implement that, we add to the main `SELECT .. FROM .. WHERE..` query structure a new part, the WHEN clause. In the WHEN clause we must always define the time related to the WHERE clause conditions. If there isn't a time definition, query statement must consider conditions during all animation time. The new structure can be defined as:

```
SELECT <objects or attributes>
FROM <database>
WHEN <time expressions>
WHERE <conditions>
```

Time expressions are built either using the time attributes, internal time of an object and animation time, or using the spatial operators (meet, unmeet etc). Using spatial operators in a time expression WHEN clause must compute the time (the instance) that the specific spatial event (meet or overlap) occurs. In WHEN clause we can also define time intervals using the well-known inequality operators (<,>,>= etc). Let's see some examples of this new query structure and put all the discussed issues all together.

An interesting question may be

"Ask for objects that have the red color when they are meeting"

These is a typical spatiotemporal question. Time in this case is indirectly defined by the meeting event. In this case a SELECT .. FROM .. WHERE .. structure couldn't give an accurate answer. The query must be able to check if at the specific time of the first object meeting, these objects has the red color. By using the structure:

```
SELECT o1, o2
FROM IAN i, i.IANobjects o1, i.IANobjects o2
WHERE colorOf(o1) = 'red' AND colourOf(o2) = 'red'
      AND meet(o1,o2)
```

we will have as result a set of objects that while they are meeting during animation, they have also the red color. Using the new structure we will have the results expected. Structure:

```
SELECT o1, o2
FROM IAN i, i.IANobjects o1, i.IANobjects o2
WHEN meet(o1,o2)
WHERE colourOf(o1) = 'red' AND colourOf(o2) = 'red'
```

returns only this pair of objects that have the red color at the specific instant of their meeting.

5.5.2.2 Queries examples

Using operators and structures defined above we can now express a variety of interesting queries. In this paragraph we try to classify in a way these queries trying to present the most important examples.

A first group of queries is to extract information about objects that may have a specific spatial status at any moment during Interactive Animation. In this case we don't need a WHEN clause and the query refers to all the interactive animation time. An example can be:

Ask if two objects, object1 and object2, are meeting during Interactive Animation named 'name'

The following statement gives an answer to this question

```
SELECT o1, o2
FROM IAN i, i.IANobjects o1, i.IANobjects o2
WHERE i.IAName = 'name' AND meet(o1, o2)
```

We can build of course similar statements using the defined spatial operators as unmeet, overlap and distance. Distance can be used in order to construct expression as distance (o1,o2) > 10 etc.

The second group of queries uses the WHEN clause and defines a specific time instance of the animation time.

Ask if two objects object1 and object2 are meeting in an Interactive Animation named 'name' at time 4

```
SELECT o1, o2
FROM IAN i, i.IANobjects o1, i.IANobjects o2
WHEN time(i) = 4
WHERE i.IAName = 'name' AND meet(o1, o2)
```

Notice that in this query it is clear the difference between time and conditions expressions. Meet operator here is used to describe a spatial condition that must be evaluated at the time defined in WHEN clause. We can write a variety of such queries combining time attributes (time and internal time) with spatial operators and database objects characteristics.

It is also very interesting to use spatial events in order to define time. We have already given an example of the use of spatial operators to define time in the paragraph of language definition. We repeat at this point the presented example.

Ask for objects that have the red color when they are meeting"

```
SELECT o1, o2
FROM IAN i, i.IANobjects o1, i.IANobjects o2
WHEN meet(o1,o2)
WHERE colourOf(o1) = 'red' AND colourOf(o2) = 'red'
```

In the WHEN clause we can use all the spatial operators meet, unmeet, overlap and distance. Distance must be used inside an equality or inequality expression as it returns a numeric value as result.

We can also define time intervals. Time intervals are a very complicated issue. At this abstract definitions we give the simplest form of time interval use. An example makes things clear.

Ask if an object has the red colour between10 and 15 animation time

```
SELECT o
FROM IAN i, i.IANobjects o
WHEN time(i)>=10 AND time(i)<=15
WHERE colourOf(o) = 'red'
```

This query will return the objects that have the red colour at least one instance inside the time interval. It would be also very interesting to check for objects that have the red colour during the specified time interval but to implement this we will probably need some extra operators. We leave this issue for the moment.

It is very interesting to build queries that ask not only database but also runtime attributes. There is some interesting examples

Ask the time that two objects o1 named 'o1' and o2 named 'o2' are meeting.

This query must be able to return the specific time instance when the two objects are first meeting.

```
SELECT timeOf(i)
```

```
FROM IAN i, i.IANobjects o1, i.IANobjects o2
WHEN meet(o1, o2)
WHERE o1.objectName ='o1' AND o2.objectName = 'o2'
```

In this case we can see a different use of the WHEN clause in order to specify the time as the query result. Position, angle, colour, activity status may be also of interest

Ask for the position of an object 'o1' at a specific time

```
SELECT positionOf(o1)
FROM IAN i, i.IANobjects o1
WHEN time(i) = 4
WHERE o1.objectName = 'o1';
```

Ask for the angle of an object o1 at a specific time

```
SELECT angleOf(o1)
FROM IAN i, i.IANobjects o1
WHEN time(i) = 4
WHERE o1.objectName = 'o1';
```

We can also ask for the distance of two objects

Ask for the distance between two objects object1 and object2 at time = 4

```
SELECT distance(o1, o2)
FROM IAN i, i.IANobjects o1, i.IANobjects o2
WHEN time(i) = 4
WHERE o1.objectName = 'object1' AND o1.objectName =
'object2';
```

A last group finally is to combine run time attribute to construct more sophisticated conditions. An example may be:

Ask for objects that between time 10 and 15 have angle equal to 90 and red color

```
SELECT o
FROM IAN i, i.IANobjects o
WHEN time(i)>=10 AND time(i)<=15
WHERE angleOf(o) = 90 AND colourOf(o) = 'red'
```

As we have already mentioned above this query will return objects that at least one time present the defined from the where part characteristics (angle = 90 and color = red). We can combine of course all the run time attributes as position, activity status etc.

5.6 Synopsis

Chapter 5 is dedicated to the database part of our system. In this section a database model is presented. Based on this model we express the most interesting queries on database contents. Then we go a bit further and we investigate the ability to query on the run time characteristics of an Interactive Animation. We explain what we need in addition to the standard OQL to express queries on dynamic data and we present the most important questions of this category.

6. Conclusions and further work

In this dissertation our basic aim was to give a first approach of Interactive Animations modeling and querying. To be more specific we have achieved to:

- Produce a grammar specification for Interactive Animations modeling
- Create a simple authoring environment for Interactive Animations
- Design a database model related to Interactive Animations and
- Express the most important queries on static and on dynamic data related to IAN

Working on this topic we have now a clear idea about what is the useful information we can extract and what are the main problems to implement such a system. We have of course to do a lot of improvements. Translator doesn't provide all the features described in the design phase. Grammar specification must be able to treat more complicated conditions and laws. In the database part we have to think about how we can implement the dynamic queries, and of course how we can simulate Interactive animations. These topics are very important and challenging. It will be very interesting to continue this effort. Undoubtedly we have gained a lot of experience as we have achieved to do the first step form the theoretical to a more technical perspective of the Interactive Animations problem. This was even the most important result.

APPENDIX A: Code Listing

A1. Translator module presentation

Translator module defines Translator's user Interface and implements the connections between lexical analyser and parser.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java_cup.runtime.*;

/* ********************************************
 *     Main Code - Creation of main frame      *
 *                                             *
 * ******************************************** */

public class Translator extends Frame {

/* Global variables that can be used from
   all the methods in this class */

   public static TextField t_TxtField = new TextField();
   public static Label t_LblMessages = new Label();

   /* The following global variable is used by
      translator button during the connection
      with lexical analyser and parser */

   static boolean do_debug_parse = false;

   public static void main(String args[]){

      /* Creating and initialise the frame */
      Translator translator = new Translator();
      Panel t_Panel = new Panel();
      BorderLayout t_borderLayout = new BorderLayout();
      Label t_giveFileLbl = new Label();

      /* TextField Static variable see below */
      /* Message label Static variable see below */

      Button translatorButton = new Button();

      translator.setSize(new Dimension(463, 269));
      translator.setVisible(true);
      translator.setTitle("Interactive Animations: Translator");

      /* Add components on the frame */
```

```
    /* Add layout ( layout doesn't work very well ) */
    translator.setLayout(t_borderLayout);

    /* Add and initialise panel */
    translator.add(t_Panel, BorderLayout.CENTER);
    t_Panel.setVisible(true);
    t_Panel.setBackground( Color.lightGray );
    t_Panel.setSize(new Dimension (463, 269));

    /* Add and initialise filename label */
    t_Panel.add(t_giveFileLbl, null);
    t_giveFileLbl.setVisible(true);
    t_giveFileLbl.setSize(new Dimension (200, 75));
    t_giveFileLbl.setForeground( Color.black );
    t_giveFileLbl.setText("Give IANs FileName:");

    /* Add and initialise textField */
    t_Panel.add(t_TxtField, null);
    t_TxtField.setVisible(true);
    t_TxtField.setSize(new Dimension (400, 100));
    t_TxtField.setColumns(18);

    /* Add and initialise translator button  */
    t_Panel.add(translatorButton, null);
    translatorButton.setVisible(true);
    translatorButton.setLabel("IAN Translator");

    /* Add and initialise message label */
    t_Panel.add(t_LblMessages, null);
    t_LblMessages.setVisible(true);
    t_LblMessages.setSize(new Dimension (200, 400));
    t_LblMessages.setForeground( Color.black );
    t_LblMessages.setText("Message label");

}

/* ********************************************
 *  This is the main code in order to connect  *
 *  the Jlex and JCup specification            *
 * ********************************************

/* Methods declaration
   Click event on the translation button */
public boolean action( Event e, Object translatorButton ) {

    /* Local variables declaration
       At first all variables about
       reading files                 */

    String IAN_File;
    FileReader IAN_FileReader;
```

```
BufferedReader IANfileBuffer = null;

/* At second all variables about
   writing output files           */
String VRML_File;
FileWriter VRML_FileWriter;
BufferedWriter  VRMLfileBuffer = null;
String VRML_line = null;

/* Initial values */
t_LblMessages.setText("");

/* Check if the text field is empty.
   if the text field is empty then
   raise an exception and exit event. */
try
{
   if ( t_TxtField.getText().length() == 0)
      throw new Exception("IANs file isn't existent");
}
catch ( Exception noFileName )
{ t_LblMessages.setText(" You must give a IAN's Filename "
    + "before click on translator button" );
  return true; }

/* if the user has given a fileName try to open it */
try
{
   IAN_File = t_TxtField.getText();
   IAN_FileReader = new FileReader( IAN_File );
   IANfileBuffer = new BufferedReader( IAN_FileReader );
}
catch ( Exception FileNotFoundException )
{ t_LblMessages.setText("File not found" );
  return true; }

/* if a consistent file exists then create
   and prepare for writing the output file
   using the default name output.wrl and putting
   it in the current directory. At first give
   a default name to the output file          */

/* create a parsing object */
parser parser_obj = new parser(new Yylex(IAN_FileReader));

Symbol parse_tree = null;
try
{
   if (do_debug_parse)
      parse_tree = parser_obj.debug_parse();
   else
```

```
            parse_tree = parser_obj.parse();
        }
        catch (Exception ex)
        {
            /* do cleanup here - - possibly rethrow e */ }
        finally
        {
            /* do close out here */
        }
        return true;

    }
}
```

A2. Lexical analysis: Jlex input specification file

The following specification is the input in Jlex java tool in order to generate a lexical analyser for the translator.

```
import java.lang.System;
import java.io.*;
import java_cup.runtime.*;

%%
%cup

%implements java_cup.runtime.Scanner
%function next_token
%type java_cup.runtime.Symbol

%eofval{
   return new Symbol(sym.EOF);
%eofval}

%line
%char

ALPHA=[A-Z a-z]
DIGIT=[0-9]
SPECIAL_CHARACTERS= [\t\ \b\r\n]
STRING_TEXT=(\\\"|[^\n\"]|\\{SPECIAL_CHARACTERS}+\\)*

%%
"OBJECT" { return new Symbol(sym.OBJECT); }
"FORM" { return new Symbol(sym.FORM); }
"TYPE" { return new Symbol(sym.TYPE); }
"LINE" { return new Symbol(sym.LINE); }
```

```
"RECTANGLE" { return new Symbol(sym.RECTANGLE); }
"ELLIPSE" { return new Symbol(sym.ELLIPSE); }
"POLYGON" { return new Symbol(sym.POLYGON); }
"GROUP" { return new Symbol(sym.GROUP); }
"POINTS" { return new Symbol(sym.POINTS); }
"CONTENTS" { return new Symbol(sym.CONTENTS); }
"FILE" { return new Symbol(sym.FILE); }
"COLOUR" { return new Symbol(sym.COLOUR); }
"COMPONENT" { return new Symbol(sym.COMPONENT); }

"EVENT" { return new Symbol(sym.EVENT); }
"START" { return new Symbol(sym.START); }
"MOUSE_CLICK" { return new Symbol(sym.MOUSE_CLICK); }
"KEYBOARD_PRESS" { return new Symbol(sym.KEYBOARD_PRESS); }
"VIS" { return new Symbol(sym.VIS); }
"AS" { return new Symbol(sym.AS); }
"IT" { return new Symbol(sym.IT); }
"POS" { return new Symbol(sym.POS); }
"ANGLE" { return new Symbol(sym.ANGLE); }
"SCALE" { return new Symbol(sym.SCALE); }
"TRUE" { return new Symbol(sym.TRUE); }
"FALSE" { return new Symbol(sym.FALSE); }
"ACTIVE" { return new Symbol(sym.ACTIVE); }
"IDLE" { return new Symbol(sym.IDLE); }
"SUSPENDED" { return new Symbol(sym.SUSPENDED); }
"MEET" { return new Symbol(sym.MEET); }
"NO_MEET" { return new Symbol(sym.NO_MEET); }
"ANY" { return new Symbol(sym.ANY); }
"SEQ" { return new Symbol(sym.SEQ); }
"TIMES" { return new Symbol(sym.TIMES); }
"CONDITION" { return new Symbol(sym.CONDITION); }
"OR" { return new Symbol(sym.OR); }
"AND" { return new Symbol(sym.AND); }
"ACTIONS" { return new Symbol(sym.ACTIONS); }
"START" { return new Symbol(sym.START); }
"STOP" { return new Symbol(sym.STOP); }
"RESUME" { return new Symbol(sym.RESUME); }
"PAUSE" { return new Symbol(sym.PAUSE); }
"HIDE" { return new Symbol(sym.HIDE); }
"SHOW" { return new Symbol(sym.SHOW); }
"PARENT" { return new Symbol(sym.PARENT); }
"ADD_MEMBER" { return new Symbol(sym.ADD_MEMBER); }
"REMOVE_MEMBER" { return new Symbol(sym.REMOVE_MEMBER); }
"DESTROY" { return new Symbol(sym.DESTROY); }
"TRANSFORMATION" { return new Symbol(sym.TRANSFORMATION); }
"TRANSLATION" { return new Symbol(sym.TRANSLATION); }
```

```
"ROTATION" { return new Symbol(sym.ROTATION); }
"SCALING" { return new Symbol(sym.SCALING); }
"COLOR_CHANGING" { return new Symbol(sym.COLOR_CHANGING); }
"TIME" { return new Symbol(sym.TIME); }

"{" { return new Symbol(sym.OPBRAC); }
"}" { return new Symbol(sym.CLOSEBRAC); }
"(" { return new Symbol(sym.OPARENT); }
")" { return new Symbol(sym.CLOSEPARENT); }
"," { return new Symbol(sym.COMMA); }
"." { return new Symbol(sym.POINT); }
"=" { return new Symbol(sym.IS); }
">" { return new Symbol(sym.GREATER); }
"<" { return new Symbol(sym.LESS); }
">=" { return new Symbol(sym.GREATER_THAN); }
"<=" { return new Symbol(sym.LESS_THAN); }
"==" { return new Symbol(sym.EQUAL); }
"<>" { return new Symbol(sym.NOT_EQUAL); }
"+" { return new Symbol(sym.PLUS); }
"*" { return new Symbol(sym.MUL); }

{SPECIAL_CHARACTERS}+ { }

{ALPHA}({ALPHA}|{DIGIT}|_)*
        { return new Symbol(sym.STRING, yytext()); }

{DIGIT}({DIGIT})*
        {return new Symbol(sym.NUMBER, yytext()); }

\"{STRING_TEXT}\" { String str =
         yytext().substring(1,yytext().length() - 1);
         return new Symbol(sym.STRING, yytext());
                                    }
.{java.lang.System.out.println("Unmatched input: " +
         yytext()); }
```

A3. Parser: Action code

Action code is the most important part of the parser. The code produced until now is presented below:

```
import java_cup.runtime.*;
import java.io.*;

action code {:
/*************** PART 1 Classes Definition ********************/
```

```
/* ----------------scenarioObject class----------------*/

/* Class describing Coordinates */

class Coord {
              int x;
              int y;
            }

/* Class describing Color */

class Colour {
               int r;
               int g;
               int b;
             }
/* Class describing a Scenario Object */

class scenarioObject {
                        String objectName;
                        String  typeValue;
                        Coord coordValues[] = new Coord[ 15 ];
                        int coordIndex = 0;
                        String contents= "";
                        boolean isFilename;
                        Colour colour =  new Colour();

                        /* variables changing continuously -
                           not treated yet */
                        boolean visibility = false;
                        String activityStatus = "IDLE";
                        int internalTime = 0;
                        float positionX;
                        float positionY;
                        float angle;
                        float scaleX;
                        float scaleY;
                      }

/* This class will be used to hold elements about the law */

class law {
             int firstConstant ;
             String sign;
             int secondConstant;
           }

/*************** LIST FOR SCENARIO OBJECTS  ********************/

/* Class describing a Node of the Objects List */
class ListNode {
```

```
                scenarioObject data;
                ListNode next;

   ListNode( scenarioObject o )
   {
      data = o;
      next = null;
   }

   ListNode( scenarioObject o, ListNode nextNode )
   {
      data = o;
      next = nextNode;
   }

   Object getObject() { return data;  }

   ListNode getnext() { return next; }
}

/* Class defining the list */
class List {
             private ListNode firstNode;
             private ListNode lastNode;
             private String ListName;
             ListNode currentNode = null;
   public List( String s )
   {
      ListName = s;
      firstNode = lastNode = null;
   }

   public void insertAtFront ( scenarioObject insertItem )
   { if  ( isEmpty() )
        firstNode = lastNode = new ListNode ( insertItem );
     else
        firstNode = new ListNode ( insertItem, firstNode );
   }

   public void insertAtBack (scenarioObject insertItem )
   {
      if ( isEmpty() )
         firstNode = lastNode = new ListNode( insertItem );
      else
      lastNode = lastNode.next = new ListNode ( insertItem );
     }

     public Object removeFromFront() throws EmptyListException
     {
        Object removeItem = null;
        if ( isEmpty() )
```

```
      throw new EmptyListException( ListName );
   removeItem = firstNode.data;
   /* reset the first node and last node references */
   if ( firstNode.equals( lastNode ) )
      firstNode = lastNode  = null;
   else
      firstNode = firstNode.next;
   return removeItem;
 }

public Object removeFromBack() throws EmptyListException
{
   Object removeItem = null;

   if ( isEmpty() )
      throw new EmptyListException (ListName);

   removeItem = lastNode.data;

   /* reset the firstNode and lastNode references */
   if ( firstNode.equals ( lastNode ))
      firstNode = lastNode = null;
   else
      {
         ListNode current = firstNode;

         while ( current.next != lastNode )
         current = current. next;
         lastNode = current;
         current.next = null;
       }

    return removeItem;
 }

 public boolean isEmpty() { return firstNode == null; }


 public void print()
 { if ( isEmpty() ) {
      System.out.println( "Empty " + ListName );
      return;
   }

  System.out.print( "The" + ListName + " is: ");
  ListNode current = firstNode;

  while ( current != null ) {
     System.out.print ( current.data.objectName + " ");
     System.out.print ( current.data.typeValue + " ");
```

```
            for ( int i = 0; i < current.data.coordIndex ; i++ ) {
               System.out.print
                   ( current.data.coordValues[i].x + " ");
               System.out.print
                   ( current.data.coordValues[i].y + " ");
            }

          current = current. next;
        }
        System.out.println();
        System.out.println();
    }

    public void first()
    {
       currentNode = firstNode;
       System.out.print ( currentNode.data.objectName + " ");
    }

    public void advance()
    {
       if ( currentNode != null )
          currentNode = currentNode.next;
    }
}


class EmptyListException extends RuntimeException {
   public EmptyListException( String ListName )
   {
      super( "The " + ListName + " is empty" );
    }
}

/*** PART 2 Classes and variables related to statements  *****/
class Event {
               String objectName;
               String eventType;
               String equationValue;
               int count;
               /* Count has a specific value in the case of
                  complex event only */
            }

class Condition {
                 /* Condition class has a structure similar to
                    the Event class */
                 String objectName;
                 String conditionType;
                 String conditionExpressionValue;
                 String operator;
```

```
                }

class Action {
                String objectName;
                String methodName;
                Coord positionValue = new Coord();
                boolean visibilityValue;
                Coord scaleValue = new Coord();
                int angleValue;
                int time;
                law laws[] = new law[3];
                int lawIndex = 0;
              }

class Statement {
                  Event Events[] = new Event[5];
                  int eventIndex = 0;
                  /* Notice that typically for every statement
                     there is only one event. To have more than
                     one events we must write a complex event
                     type. We suppose that a complex event can't
                     have more than five simple events
                     in the composition */
                  Condition Conditions[] = new Condition[5];
                  int conditionIndex = 0;
                  Action Actions[] = new Action[5];
                  int actionIndex = 0;
                }

/*************** LIST FOR STATEMENT OBJECTS  ***************/

/* Class describing a Node of the Objects List */
class ListNodeStat {
                     Statement data;
                     ListNodeStat next;

   ListNodeStat( Statement o )
   {
      data = o;
      next = null;
   }

   ListNodeStat( Statement o, ListNodeStat nextNode )
   {
      data = o;
      next = nextNode;
    }

    Object getObject() { return data;  }

    ListNodeStat getnext() { return next; }
```

```
}

/* Class defining the list */
class ListStat {
                private ListNodeStat firstNode;
                private ListNodeStat lastNode;
                private String ListName;
                ListNodeStat currentNode = null;

   public ListStat( String s )
   {
      ListName = s;
      firstNode = lastNode = null;
                        }
   public void insertAtFront ( Statement insertItem )
   { if  ( isEmpty() )
        firstNode = lastNode = new ListNodeStat ( insertItem );
     else
        firstNode = new ListNodeStat ( insertItem, firstNode );
   }

    public void insertAtBack (Statement insertItem )
    {
       if ( isEmpty() )
          firstNode = lastNode = new ListNodeStat( insertItem );
       else
          lastNode = lastNode.next = new ListNodeStat
                                     ( insertItem );
    }

    public Object removeFromFront() throws EmptyListException
    {
       Object removeItem = null;
       if ( isEmpty() )
          throw new EmptyListException( ListName );
          removeItem = firstNode.data;

       /* reset the first node and last node references */
       if ( firstNode.equals( lastNode ) )
          firstNode = lastNode  = null;
       else
          firstNode = firstNode.next;
       return removeItem;
    }


    public Object removeFromBack() throws EmptyListException
    {
       Object removeItem = null;

       if ( isEmpty() )
```

```
      throw new EmptyListException (ListName);

   removeItem = lastNode.data;

   /* reset the firstNode and lastNode references */

   if ( firstNode.equals ( lastNode ))
      firstNode = lastNode = null;
   else
   {
      ListNodeStat current = firstNode;

      while ( current.next != lastNode )
      current = current. next;
      lastNode = current;
      current.next = null;
   }

   return removeItem;
                        }
public boolean isEmpty() { return firstNode == null; }


public void print()
{ if ( isEmpty() ) {
      System.out.println( "Empty " + ListName );
      return;
   }

   System.out.print( "The" + ListName + " is: ");
   ListNodeStat current = firstNode;

   while ( current != null ) {
      current = current. next;
   }

   System.out.println();
   System.out.println();
 }

 public void first()
 {
    currentNode = firstNode;
 }

 public void advance()
 {
    if ( currentNode != null )
    currentNode = currentNode.next;
  }
```

```
 }

/********** PART 3 Global Variables Definition ***************/

String VRML_File;
FileWriter VRML_FileWriter;
BufferedWriter  VRMLfileBuffer = null;
String VRML_line = null;

scenarioObject currentObject = new scenarioObject();
List scenario_object_list = new List("Object List");
Event currentEvent = new Event();
Condition currentCondition = new Condition();
Action currentAction = new Action();
law currentLaw = new law();
Statement currentStatement = new Statement();
ListStat statementList = new ListStat("Statement List");

/******* PART 4 Program Procedures Definition *************/

/********************OBJECT CREATION***********************/

/* ------------------Create_Output_File-------------------------*/

/*  Create the file and assign the first common file lines  */


public void Create_Output_File()
{

   try
   {
      VRML_File = "d:\\jdk1.2.1\\bin\\output.wrl";
      VRML_FileWriter = new FileWriter( VRML_File );
      VRMLfileBuffer = new BufferedWriter( VRML_FileWriter );

      /* Write the first basic line */
      VRML_line = "#VRML V2.0 utf8";
      VRMLfileBuffer.write( VRML_line );
      VRMLfileBuffer.newLine( );
    }

    catch ( Exception IOException )
    {
       System.out.println("1" );
       System.out.println
          ("I can't create or write to this file" );
    }
 }  /* End of Create_Output_File() */
```

```
/* ------------Close_Output_File---------------------------*/

public void Close_Output_File() {

   try
   {
      VRMLfileBuffer.close( );
    }

    catch ( Exception IOException )
    {
       System.out.println("2" );
       System.out.println("I can't close this file" );
     }
}  /* End of Close_Output_File() */

/* ----------------Assign object to List Node-----------*/

public void Assign_Object_To_ListNode() {

   scenario_object_list.insertAtBack(currentObject);
}

/* ------------Write a line to the VRML file--------------*/
public void writeVRMLline() {

   try
   {
      VRMLfileBuffer.write( VRML_line );
      VRMLfileBuffer.newLine( );
   }
   catch ( Exception IOException )
   {
      System.out.println("3" );
      System.out.println("I can't close this file" );
    }
}

/* ---------------Clear_Scenario_Object-------------------*/

public void Clear_Scenario_Object() {
   currentObject =  new scenarioObject();
}

/* ----------Assign_Object_Name_Value-------------------*/

public void Assign_Object_Name_Value( String lobjectName) {
   currentObject.objectName = lobjectName;
}

/* -------------Assign_Type_Value-----------------------*/
```

```
public void Assign_Type_Value( String ltypeValue) {
   currentObject.typeValue = ltypeValue;
}

/* -------------------Assign_Coordinate_x-----------------*/
public void Assign_coordinate_x( int lcoord ) {

   currentObject.coordValues[currentObject.coordIndex] =
         new Coord();
   currentObject.coordValues[currentObject.coordIndex].x =
         lcoord;
}

/* -----------------Assign_Coordinate_y--------------------*/

public void Assign_coordinate_y( int lcoord ) {

   currentObject.coordValues[currentObject.coordIndex].y =
         lcoord;
   currentObject.coordIndex = currentObject.coordIndex + 1;
}

/* -----------------assignContents-----------------------*/

public void assignContents( String lContents ) {
   currentObject.contents = lContents;
}

/* -----------------assignIsfile-------------------------*/

public void assignIsfile( String lisFile ) {
   if (lisFile == "FILE")
   {
      currentObject.isFilename = true;
   }
   else
      currentObject.isFilename = false;
}

/* -----------------Assign_colour_r----------------------*/

public void Assign_colour_r( int lr) {
   currentObject.colour.r = lr;
}

/* ---------------Assign_colour_g-----------------------*/

public void Assign_colour_g( int lg) {
   currentObject.colour.g = lg;
}
```

```
/* ---------------Assign_colour_b-------------------------*/

public void Assign_colour_b( int lb) {
   currentObject.colour.b = lb;
}

/* ---------------Create_Objects-----------------------*/

public void Create_Objects() {
   try
   {
      scenario_object_list.first();
      while ( scenario_object_list.currentNode != null ) {
         if ( scenario_object_list.currentNode.data.typeValue
            == "RECTANGLE" )
           createRectangle();
         else if (scenario_object_list.currentNode.data.typeValue
              == "LINE" )
           createLine();
         else if (scenario_object_list.currentNode.data.typeValue
              == "ELLIPSE" )
           createEllipse();
         else if (scenario_object_list.currentNode.data.typeValue
              == "POLYGON" )
            createPolygon();
         else if (scenario_object_list.currentNode.data.typeValue
              == "GROUP" )
            createGroup();

         scenario_object_list.currentNode =
            scenario_object_list.currentNode. next;
      } /* end while */
     }

     catch ( Exception IOException )
     {
        System.out.println("4" );
        System.out.println
           ("I can't create or write to this file" );   }
     } /* end of method */


/* ----------------CreateRectangle-----------------*/

public void createRectangle()
{
    try
    {
       /* Create PROTO for Rectangle */

      VRML_line = "PROTO " +
```

```
    scenario_object_list.currentNode.data.objectName + "[ ]";
  writeVRMLline();
  VRML_line = "{";
  writeVRMLline();

  /* Create shape node */
  VRML_line = "                Shape { ";
  writeVRMLline();
  VRML_line = "                    geometry Box { size "  +
     Integer.toString
     (scenario_object_list.currentNode.data.coordValues[0].x)
     + " " +
     Integer.toString
     (scenario_object_list.currentNode.data.coordValues[0].y)
     + " 0.0000001 }";
  writeVRMLline();

  /* Create appearance node */
  VRML_line = "                                appearance
            Appearance { ";
  writeVRMLline();

  /* Create Material Node */
  VRML_line = "
                   material Material {  ";
  writeVRMLline();

/* check if it is needed to create the
   diffuseColor Attribute */
if ( (scenario_object_list.currentNode.data.colour.r == 0)
    &&
     (scenario_object_list.currentNode.data.colour.g == 0)
    &&
     (scenario_object_list.currentNode.data.colour.b == 0) )
{
   /* Close Material Node */
   VRML_line = "                              }  ";
   writeVRMLline();
}
else
/* create diffuseColor attribute */
{
   VRML_line = "                          diffuseColor   " +
            Integer.toString
            (scenario_object_list.currentNode.data.colour.r)
            + " " +
            Integer.toString
            (scenario_object_list.currentNode.data.colour.g)
            + " " +
            Integer.toString
            (scenario_object_list.currentNode.data.colour.b)
```

```
                + " }";
            writeVRMLline();
        }

    /* if there is contents then we must create
       the ImageTexture Node */
    if (scenario_object_list.currentNode.data.contents != "")
    {
        VRML_line = "texture ImageTexture { url "
            + scenario_object_list.currentNode.data.contents
            + " }" ;
        writeVRMLline();
    }


/* Close Appearance Node */
    VRML_line = "                }  ";
    writeVRMLline();
/* Close Shape Node */
    VRML_line ="        }";
    writeVRMLline();
/* End of PROTO Rectangle */
    VRML_line = "} #end of RECTANGLE prototype";
    writeVRMLline();

/* write an empty line to separate PROTO */
    VRML_line = "";
    writeVRMLline();

}
catch ( Exception IOException )
{
    System.out.println("5" );
    System.out.println("I can't create or write to this file" );
}
                    }
/* ---------------------Create Line-------------------------*/

public void createLine()
{
    try
    {
        /* Create PROTO for Line */

        VRML_line = "PROTO " +
                scenario_object_list.currentNode.data.objectName
                + " [ ]";
        writeVRMLline();
        VRML_line = "{";
        writeVRMLline();
```

```
/* Create shape node */
VRML_line = "                Shape { ";
writeVRMLline();
VRML_line = "                  geometry IndexedLineSet {  ";
writeVRMLline();

/* Create coord Coordinate node */
VRML_line = "                                coord Coordinate {
          point [  ";
for ( int i = 0; i <
scenario_object_list.currentNode.data.coordIndex ; i++ ) {
   VRML_line = VRML_line + Integer.toString
    (scenario_object_list.currentNode.data.coordValues[i].x)
    + " ";
    VRML_line = VRML_line + Integer.toString
    (scenario_object_list.currentNode.data.coordValues[i].y)
    + " ";
    VRML_line = VRML_line + "0 ";
    VRML_line = VRML_line + "    ";
}
VRML_line = VRML_line + " ] } ";
writeVRMLline();

/* Create Coord Index attributes */
VRML_line = "coordIndex [    ";
for ( int i = 0; i <
scenario_object_list.currentNode.data.coordIndex ; i++ ) {
   VRML_line = VRML_line + Integer.toString(i) + " ";
}
VRML_line = VRML_line + " -1 ]";
writeVRMLline();

/* create color attribute */
VRML_line = "color Color { color [ " +
Integer.toString
(scenario_object_list.currentNode.data.colour.r) + " " +
Integer.toString
(scenario_object_list.currentNode.data.colour.g) + " " +
Integer.toString
(scenario_object_list.currentNode.data.colour.b) + " ]}";
writeVRMLline();

/* create colorPerVertex attribute */
VRML_line = "colorPerVertex FALSE ";
writeVRMLline();

/* Close Indexed Line Set Node */
VRML_line ="       }";
writeVRMLline();

/* Close Shape Node */
```

```
      VRML_line =" }";
      writeVRMLline();
      /* End of PROTO Line */
      VRML_line = "} #end of LINE prototype";
      writeVRMLline();
   }
   catch ( Exception IOException )
   {
      System.out.println("6" );
      System.out.println
         ("I can't create or write to this file" );   }
   }

/* -------------Create Ellipse---------------------------*/

public void createEllipse()
{
   /* local variables declaration */
   float scaleFactor;

   try
   {
      /* Create PROTO for Ellipse */

      VRML_line = "PROTO " +
         scenario_object_list.currentNode.data.objectName
         + " [ ]";
      writeVRMLline();
      VRML_line = "{";
      writeVRMLline();

      /* Create Transform node */
      VRML_line = "               Transform { ";
      writeVRMLline();
      VRML_line = "                rotation  0 1 1 3.14 ";
      writeVRMLline();
      VRML_line = "                scale   ";

      /* compute x radious */
      scaleFactor =
         (scenario_object_list.currentNode.data.coordValues[0].x
         / 2)
         / scenario_object_list.currentNode.data.coordValues[0].y
         / 2);
      VRML_line = VRML_line + Float.toString(scaleFactor)
         + " 1 1 ";
      writeVRMLline();

      /* Create children Shape Node */
      VRML_line = "       children  Shape {  ";
      writeVRMLline();
```

```
      /* create geometry node */
      VRML_line = "geometry Cylinder { radius " +
         scenario_object_list.currentNode.data.coordValues[0].y
         /2 +
         " height 0.00000000001 }";
      writeVRMLline();

     /* create appearance node */
     VRML_line = "appearance Appearance { ";
     writeVRMLline();

     /* create material node */
     VRML_line = "material Material { diffuseColor " +
        Integer.toString
        (scenario_object_list.currentNode.data.colour.r)
        + " " +
        Integer.toString
        (scenario_object_list.currentNode.data.colour.g)
        + " " +
        Integer.toString
        (scenario_object_list.currentNode.data.colour.b)
        + " }";
     writeVRMLline();

     /* Close appearance node */
     VRML_line ="       }";
     writeVRMLline();

     /* Close Shape Node */
     VRML_line =" }";
     writeVRMLline();

     /* Close Transform Node */
     VRML_line =" }";
     writeVRMLline();

     /* End of PROTO Ellipse */
     VRML_line = "} #end of ELLIPSE prototype";
     writeVRMLline();
   }
   catch ( Exception IOException )
   {
      System.out.println("7" );
      System.out.println
       ("I can't create or write to this file" );    }
    }

/* -------Create Polygon-------------------------*/
public void createPolygon()
{
```

```
try
{
   /* Create PROTO for Polygon */

   VRML_line = "PROTO " +
      scenario_object_list.currentNode.data.objectName
      + " [ ]";
   writeVRMLline();
   VRML_line = "{";
   writeVRMLline();

   /* Create shape node */
   VRML_line = "                Shape { ";
   writeVRMLline();
   VRML_line = "                 geometry IndexedFaceSet {  ";
   writeVRMLline();

  /* Create coord Coordinate node */
  VRML_line = "                              coord Coordinate {
            point [  ";
  for ( int i = 0; i <
  scenario_object_list.currentNode.data.coordIndex ; i++ ) {
     VRML_line = VRML_line + Integer.toString
     (scenario_object_list.currentNode.data.coordValues[i].x)
     + " ";
     VRML_line = VRML_line + Integer.toString
     (scenario_object_list.currentNode.data.coordValues[i].y)
     + " ";
     VRML_line = VRML_line + "0 ";
     VRML_line = VRML_line + "    ";
   }
   VRML_line = VRML_line + " ] } ";
   writeVRMLline();

    /* Create Coord Index attributes using a special
       algorithm in case of polygon */
    VRML_line = "coordIndex [    ";
    for ( int k = 0; k <
    scenario_object_list.currentNode.data.coordIndex; k++) {
       for ( int i = 0; i <
       scenario_object_list.currentNode.data.coordIndex;i++ )
       {                                        if  (k != i)
         VRML_line = VRML_line + Integer.toString(i) + " ";
       } /* end of for i */
       VRML_line = VRML_line + "-1 ";
    } /* end of for k */
     VRML_line = VRML_line + " ]  ";
     writeVRMLline();

     /* create color attribute */
     VRML_line = "color Color { color [ ";
```

```
        for ( int i = 0; i <
        scenario_object_list.currentNode.data.coordIndex;i++ ) {
           VRML_line = VRML_line +  Integer.toString
           (scenario_object_list.currentNode.data.colour.r)
           + " ";
           VRML_line = VRML_line + Integer.toString
           (scenario_object_list.currentNode.data.colour.g)
           + " ";
           VRML_line = VRML_line + Integer.toString
           (scenario_object_list.currentNode.data.colour.b)
           + ", ";
        }
        VRML_line = VRML_line + "]}";
        writeVRMLline();

        /* create colorPerVertex attribute */
        VRML_line = "colorPerVertex FALSE ";
        writeVRMLline();

        /* Close Indexed Face Set Node */
        VRML_line ="       }";
        writeVRMLline();

        /* Close Shape Node */
        VRML_line =" }";
        writeVRMLline();
        /* End of PROTO Polygon */
        VRML_line = "} #end of LINE prototype";
        writeVRMLline();
      }
      catch ( Exception IOException )
      {
        System.out.println("8" );
        System.out.println
          ("I can't create or write to this file" );   }
       }

/* -----------------Create Group-------------------------*/
public void createGroup()
{
/* In this function we must be able to create and groups inside a
group. For this purpose we must use a stack structure. At this
moment we will create the simplest group algorithm which is about
one single group */

scenarioObject groupTable[] = new scenarioObject[10];
/* We suppose that the group has in the maximum 10 objects */

int objectCount;
/* This variable shows how many objects the group contains */
```

```
groupTable[0] = scenario_object_list.currentNode.data;
/* The first element of the table is always the group node.
   It is this place that I always must use as reference */

for ( int i = 0; i < groupTable[0].coordIndex ; i++ ) {
   scenario_object_list.advance();
   groupTable[i + 1] =  scenario_object_list.currentNode.data;

   /* Writing in the VRML file the PROTO for each group */
   if ( scenario_object_list.currentNode.data.typeValue
       == "RECTANGLE" )
      createRectangle();
   else if ( scenario_object_list.currentNode.data.typeValue
       == "LINE" )
      createLine();
   else if ( scenario_object_list.currentNode.data.typeValue
       == "ELLIPSE" )
      createEllipse();
   else if ( scenario_object_list.currentNode.data.typeValue
       == "POLYGON" )
      createPolygon();
}

/* Create PROTO for group using information from groupTable */

try
{
   VRML_line = "PROTO " + groupTable[0].objectName + " [ ]";
   writeVRMLline();
   VRML_line = "{";
   writeVRMLline();

   /* Create group node */
   VRML_line = "               Group { ";
   writeVRMLline();
   VRML_line = "                children [  ";
   writeVRMLline();

   for ( int i = 1; i <= groupTable[0].coordIndex; i++ ) {

      /* Create Transform node for each child */
      VRML_line = "                              Transform { ";
      writeVRMLline();

      /* Translate object */
      VRML_line = "                                    translation  "
      + groupTable[0].coordValues[i-1].x + " " +
      groupTable[0].coordValues[i-1].y + " 0";
      writeVRMLline();

      /* Define children name */
```

```
      VRML_line = "                                    children " +
      groupTable[i].objectName + " {}";
      writeVRMLline();
      VRML_line = "                                    }" ;
      writeVRMLline();
   }    /* end for command for each child */

   /* Close children and group node */
   VRML_line = "                                ]}" ;
   writeVRMLline();
   /* Close PROTO node */
   VRML_line = "                          }" ;
   writeVRMLline();
} /* end try */
catch ( Exception IOException )
{
   System.out.println("9" );
   System.out.println("I can't create or write to this file" );
}

}

/******** PART 5 Program Procedures Definition ************/

/************************EVENT CREATION************************/

/* -------Assign Event Object Name-----------------------------*/

public void assignEventObjectName( String leventObjectName) {
   currentEvent.objectName = leventObjectName;
}

/* -------Assign Event Type------------------------*/

public void assignEventType( String leventType) {
   currentEvent.eventType = leventType;
}

/* --------Assign Equation Value---------------------*/
public void assignEquationValue( String lequationValue) {
/* This function must read the string equation value and
translate it to the right format. It is not defined for the
moment */
}

/* ------Assign Count Value-------------------------*/
public void assignCount( int lcount) {
   currentEvent.count = lcount;
}

/* -------Assign Event to Event Array ------------*/
```

```
public void assignEventToEventArray() {

   currentStatement.Events[currentStatement.eventIndex]
   = currentEvent;
   currentStatement.eventIndex = currentStatement.eventIndex + 1;
}

/* ---------Clear  current event object ---------------*/
public void clearCurrentEvent() {
   currentEvent =  new Event();
}

/* ---------Create event ----------------------------*/
public void createEvent(Event levent) {
   if ( levent.eventType == "START" )
      startEvent(levent);
   else if ( levent.eventType == "MOUSE_CLICK" )
      mouseClick(levent);
}

/* - ----Events procedures ----------------------------*/

/* ----------START EVENT ----------------------------*/
public void startEvent(Event levent) {
/* Start event doesn't produce any VRML code but is written for
   compatibility reason */
}

/* ---------MOUSE CLICK EVENT  ---------------------*/
public void mouseClick(Event levent) {
/* Mouse click event assign to Touch Sensor 's enabled attribute
   the value TRUE  */
try
{
   /* write an empty line */
   VRML_line = " ";
   writeVRMLline();

   VRML_line =  "  DEF TOUCH" + levent.objectName
   + " TouchSensor   { enabled TRUE }" ;
   writeVRMLline();

   /* write another empty line  */
   VRML_line = " ";
   writeVRMLline();
}
catch ( Exception IOException )
{
   System.out.println("10" );
   System.out.println("I can't create or write to this file" );
}
```

```
}

/* ------------KEYBOARD PRESS EVENT  -------------------*/
public void keyboardPress(Event levent) {

}

/* ------------ COLOUR EXPRESSION EVENT -----------------*/
public void colourExpression(Event levent) {

}

/* ----------- CONTENTS EXPRESSION EVENT  ---------------------*/
public void contentsExpression(Event levent) {

}

/* ----------- VISIBILITY EXPRESSION EVENT  -------------------*/
public void visibilityExpression(Event levent) {

}

/* ----------- ACTIVITY  STATUS EXPRESSION EVENT  ------------*/
public void activityStatusExpression(Event levent) {

}

/* ---------- INTERNAL TIME EXPRESSION EVENT  ----------------*/
public void internalTimeExpression(Event levent) {

}

/* -------------- POSITION EXPRESSION EVENT  -----------------*/
public void positionExpression(Event levent) {

}

/* ---------------- ANGLE EXPRESSION EVENT  -----------------*/
public void angleExpression(Event levent) {

}

/* ------------- SCALE EXPRESSION EVENT  --------------------*/
public void scaleExpression(Event levent) {

}

/* -------------- MEET EXPRESSION EVENT  --------------------*/
public void meetExpression(Event levent) {

}
```

```
/* ------------- NO MEET EXPRESSION EVENT  --------------------*/
public void noMeetExpression(Event levent) {

}

/* --------------- TIME EVENT  -------------------------------*/
public void timeEvent(Event levent) {

}

/* ----------------- ANY EVENT  -------------------------------*/
public void any(Event levent) {

}

/* --------------- SEQ EVENT  --------------------------------*/
public void seq(Event levent) {

}

/* --------------- TIMES EVENT  ------------------------------*/
public void times(Event levent) {

}

/********** PART 6 Program Procedures Definition **********/
/*********************CONDITION CREATION**********************/
/* ---------Assign Condition Object Name--------------------*/
public void assignConditionObjectName( String
            lconditionObjectName) {
   currentCondition.objectName = lconditionObjectName;
}

/* ----------Assign Condition Type--------------------------*/
public void assignConditionType( String lconditionType) {
   currentCondition.conditionType = lconditionType;
}

/* --------Assign condition expression Value----------------*/
public void assignconditionExpressionValue
       ( String lconditionExpressionValue) {
/* This function must read the string equation value and
translate it to the right format. It is not defined for the
moment */
}

/* ---------------Assign operator Value---------------------*/
public void assignOperator( String loperator) {
   currentCondition.operator = loperator;
}
```

```
/* -----------Assign conditions to Condition Array ---------*/
public void assignConditionToConditionArray() {
   currentStatement.Conditions[currentStatement.conditionIndex]
    = currentCondition;
   currentStatement.conditionIndex =
    currentStatement.conditionIndex + 1;
}

/* ---------Clear  current condition object ----------------*/
public void clearCurrentCondition() {
   currentCondition =  new Condition();
 }

/* --------------- Create condition -------------------------*/
public void createCondition(Condition lcondition) {
/* no if statement defined yet */
}

/* -----------Conditions procedures -------------------------*/

/* --------------COLOUR CONDITION  -------------------------*/
public void colourCondition(Condition lcondition) {

}

/* -----------------CONTENTS CONDITION  ---------------------*/
public void contentsCondition(Condition lcondition) {

}

/* -------------VISIBILITY CONDITION  ---------------------*/
public void visibilityCondition(Condition lcondition) {

}

/* -------------ACTIVITY  STATUS CONDITION  -----------------*/
 public void activityStatusCondition(Condition lcondition) {

}

/* ------------INTERNAL TIME CONDITION  --------------------*/
public void internalTimeCondition(Condition lcondition) {

}

/* -------------POSITION CONDITION  -------------------------*/
public void positionCondition(Condition lcondition) {

}
```

```
/* -----------ANGLE CONDITION  ------------------------------*/
public void angleCondition(Condition lcondition) {

}

/* -------------SCALE CONDITION  ----------------------------*/
public void scaleCondition(Condition lcondition) {

}

/* ------------MEET CONDITION  -----------------------------*/
public void meetCondition(Condition lcondition) {

}

/* ----------- NO MEET CONDITION  --------------------------*/
public void noMeetCondition(Condition lcondition) {

}

/* --------------------TIME CONDITION  ----------------------*/
public void timeCondition(Condition lcondition) {

}

/********** PART 7 Program Procedures Definition *********/

/*********************ACTIONS CREATION*********************/

/* ---------------Assign Action Object Name -------------*/
public void assignActionObjectName( String lobjectName) {
   currentAction.objectName = lobjectName;
}

/* ---------Assign Method Name --------------------------------*/
public void assignMethodName( String lmethodName) {
   currentAction.methodName = lmethodName;
}

/* ---------------Assign Action Position X -------------------*/
public void assignActionPosX( int lactionPosX) {
   currentAction.positionValue.x = lactionPosX;
}

/* ----------Assign Action Position Y ----------------------*/
public void assignActionPosY( int lactionPosY) {
   currentAction.positionValue.y = lactionPosY;
}

/* ---------Assign Action Visibility ------------------------*/
public void assignActionVisibility(String lactionVisibility) {
```

```
if (lactionVisibility == "TRUE")
   currentAction.visibilityValue = true;
else if (lactionVisibility == "FALSE")
   currentAction.visibilityValue = false;
else
   System.out.println
  ("This is not an acceptable visibility value");
}

/* ---------------Assign Action Scale X ---------------------*/
public void assignActionScaleX( int lactionScaleX) {
   currentAction.scaleValue.x = lactionScaleX;
}

/* ----------Assign Action Scale Y ---------------------------*/
public void assignActionScaleY( int lactionScaleY) {
   currentAction.scaleValue.y = lactionScaleY;
}

/* ---------------Assign Action angle law --------------------*/
public void assignActionAngleLaw() {
   currentAction.laws[0] = currentLaw;
   currentLaw = new law();
}

/* --------Assign Action Translation law ---------------------*/
public void assignActionTranslationLaw() {
   if  ( currentAction.lawIndex == 0 ) {
      currentAction.laws[0] = currentLaw;
      currentAction.lawIndex = currentAction.lawIndex + 1;
      currentLaw = new law();
    }
    else
    {
      currentAction.laws[1] = currentLaw;
      currentLaw = new law();
     }
}

/* ---------------Assign Action Scaling law ------------------*/
public void assignActionScalingLaw() {
   if  ( currentAction.lawIndex == 0 ) {
      currentAction.laws[0] = currentLaw;
      currentAction.lawIndex = currentAction.lawIndex + 1;
      currentLaw = new law();
    }
    else
    {
       currentAction.laws[1] = currentLaw;
       currentLaw = new law();
    }
```

```
}

/* -------Assign Action to action Array ----------------------*/
public void assignActionToActionArray() {

   currentStatement.Actions[currentStatement.actionIndex] =
   currentAction;
   currentStatement.actionIndex = currentStatement.actionIndex
   + 1;
}

/* --------------Clear  current Action object -----------------*/
public void clearCurrentAction() {
   currentAction =  new Action();
}


/* ------------- Create Nodes for Actions --------------------*/
public void createActions(Action laction) {
try
{
   if (laction.methodName == "START" )
      start(laction);
   else if (laction.methodName == "STOP")
      stop(laction);
   else if (laction.methodName == "RESUME")
      resume(laction);
   else if (laction.methodName == "PAUSE")
      pause(laction);
   else if (laction.methodName == "HIDE")
      hide(laction);
   else if (laction.methodName == "SHOW")
      show(laction);
   else if (laction.methodName == "ADD_MEMBER")
      addMember(laction);
   else if (laction.methodName == "REMOVE_MEMBER")
      removeMember(laction);
   else if (laction.methodName == "DESTROY")
      destroyChild(laction);
   else if (laction.methodName == "DISCRETE_TRANSLATION")
      discreteTranslation(laction);
   else if (laction.methodName == "DISCRETE_ROTATION")
      discreteRotation(laction)
   else if (laction.methodName == "DISCRETE_SCALING")
      discreteScaling(laction);
    else if (laction.methodName == "DISCRETE_COLOR_CHANGING")
      discreteColorChanging(laction);
    else if (laction.methodName == "CONTINUOUS_TRANSLATION")
      continuousTranslation(laction);
    else if (laction.methodName == "CONTINUOUS_ROTATION")
```

```
      continuousRotation(laction);
    else if (laction.methodName == "CONTINUOUS_SCALING")
      continuousScaling(laction);
    else if (laction.methodName == "CONTINUOUS_COLOR_CHANGING")
      continuousColorChanging(laction);
}
catch ( Exception IOException )
{
   System.out.println("11" );
   System.out.println("I can't create or write to this file" );
}
} /* end of method */

/* -----------------Start Action ----------------------------*/
public void start( Action laction ) {
/* This action is joined with Mouse_click action. This must
   probably change in the future */
try
{
   /* Create Translation for the refered Object */
   VRML_line = "DEF " + laction.objectName + " Transform { ";
   writeVRMLline();
   VRML_line =  "  translation " +
   Integer.toString(laction.positionValue.x) + " "
   + Integer.toString(laction.positionValue.y) + " 0" ;
   writeVRMLline();

   /* Create children node */
   VRML_line = "    children [ " + laction.objectName + "{ } ]";
   writeVRMLline();

   /* Close transform node */
   VRML_line = " }";
   writeVRMLline();
}
catch ( Exception IOException )
{
  System.out.println("12" );
  System.out.println("I can't create or write to this file" );
}
}

/* ---------------------Stop Action --------------------------*/
public void stop(Action laction) {

}

/* ---------------------Resume Action ------------------------*/
public void resume(Action laction) {

}
```

```
/* -------------Pause Action ----------------------------------*/
public void pause(Action laction) {

}

/* ---------------Hide Action ------------------------------*/
public void hide(Action laction) {

}

/* ----------------------Show Action -------------------------*/
public void show(Action laction) {

}

/* --------------Add Member Action ----------------------------*/
public void addMember(Action laction) {

}

/* ----------------Remove Member Action ----------------------*/
public void removeMember(Action laction) {

}

/* ----------------Destroy Action
public void destroyChild(Action laction) {

}

/* ---------Discrete Translation Action ---------------------*/
public void discreteTranslation(Action laction) {
int x;
int y;
try
{
   /* Create position Interpolator  */
   VRML_line = " DEF POS" + laction.objectName + "
   PositionInterpolator { ";
   writeVRMLline();

   /* in Discrete transformation key  has only the zero value */
   VRML_line =  " key [ 0 ] " ;
   writeVRMLline();

   /* Key values */
   VRML_line =  " keyValue [  " ;
   VRML_line =  VRML_line +
   Integer.toString(laction.positionValue.x) + " " +
   Integer.toString(laction.positionValue.y) + " 0";
```

```
    VRML_line =  VRML_line + " ] }";
    writeVRMLline();

    /* Empty line */
    VRML_line =  "  " ;
    writeVRMLline();

     /* Create time sensor to trigger the position Interpolator */
     VRML_line = " DEF TIME" + laction.objectName + " TimeSensor
     { }";
     writeVRMLline();

     /* Empty line */
     VRML_line =  "  " ;
     writeVRMLline();

     /* Creating routes to translate object */
     VRML_line = "ROUTE TOUCH" + laction.objectName +
     ".touchTime TO TIME" + laction.objectName + ".startTime";
     writeVRMLline();

     VRML_line = "ROUTE TIME" + laction.objectName +
     ".fraction_changed TO POS" + laction.objectName +
     ".set_fraction";
     writeVRMLline();

     VRML_line = "ROUTE POS" + laction.objectName +
     ".value_changed TO " + laction.objectName +
     ".set_translation";
     writeVRMLline();
}
catch ( Exception IOException )
{
   System.out.println("13" );
   System.out.println("I can't create or write to this file" );
}
}
/* -----------Discrete Rotation Action ----------------------*/
public void discreteRotation(Action laction) {

}
/* ------------ Discrete Scaling Action ---------------------*/
public void discreteScaling(Action laction) {

}
/* --------Discrete Color Changing Action -------------------*/
public void discreteColorChanging(Action laction) {
}
/* --------Continuous Translation Action --------------------*/
public void continuousTranslation(Action laction) {
int x;
```

```
int y;
try
{
   /* write an empty line */
   VRML_line = " ";
   writeVRMLline();

   /* Create position Interpolator  */
   VRML_line = " DEF POS" + laction.objectName + "
   PositionInterpolator { ";
   writeVRMLline();

   /* we decide to approximate using 10 keys each time */
   VRML_line =  " key [  " ;
   for ( int i = 0;  i<= 9 ; i++ ) {
      VRML_line =  VRML_line + " " + Integer.toString(i);
    }
    VRML_line = VRML_line + " ]";
    writeVRMLline();

    /* write an empty line */
    VRML_line = " ";
    writeVRMLline();

    /* Approximate key values */
    VRML_line =  " keyValue [  " ;
    for ( int i = 0; i <= 9 ; i++ ) {
       x =  laction.laws[0].firstConstant*i +
            laction.laws[0].secondConstant;
       y =  laction.laws[1].firstConstant*i +
            laction.laws[1].secondConstant;
       VRML_line =  VRML_line + Integer.toString(x) + " "
       + Integer.toString(y) + " 0,";
    }
    VRML_line =  VRML_line + " ] }";
    writeVRMLline();

    /* write an empty line */
    VRML_line = " ";
    writeVRMLline();

    /* Create time sensor using 3.0 as interval */
    VRML_line = " DEF TIME" + laction.objectName
    + " TimeSensor { ";
    writeVRMLline();
    VRML_line = " cycleInterval 3.0 }";
    writeVRMLline();

    /* write an empty line */
    VRML_line = " ";
    writeVRMLline();
```

```
    /* Creating routes to animate object */
    VRML_line = "ROUTE TOUCH" + laction.objectName +
    ".touchTime TO TIME" + laction.objectName
    + ".startTime";
    writeVRMLline();

    VRML_line = "ROUTE TIME" + laction.objectName +
    ".fraction_changed TO POS" + laction.objectName +
    ".set_fraction";
    writeVRMLline();

    VRML_line = "ROUTE POS" + laction.objectName +
    ".value_changed TO " + laction.objectName +
    ".set_translation";
    writeVRMLline();
}
catch ( Exception IOException )
{
   System.out.println("14" );
   System.out.println("I can't create or write to this file" );
}
}
/* -----------Continuous Rotation Action ----------------------*/
public void continuousRotation(Action laction) {
double angle;
double keyArray[] = { 0.0, 0.1, 0.3, 0.6, 0.8, 1.0 } ;
try
{

   /* write an empty line */
   VRML_line = " ";
   writeVRMLline();

   /* Create orientationInterpolator  */
   VRML_line = " DEF OI" + laction.objectName + "
   OrientationInterpolator { ";
   writeVRMLline();

   /* Create Keys. We decide to approximate using 6 keys each
      time */
   VRML_line =  " key [ 0.0, 0.1, 0.3, 0.6, 0.8, 1.0 ]  " ;
   writeVRMLline();

   /* Create key values using the law */
   VRML_line =  " keyValue [  " ;
   for ( int i = 0; i <= 5 ; i++ ) {
      angle = laction.laws[0].firstConstant*keyArray[i] +
      laction.laws[0].secondConstant;
      VRML_line =  VRML_line + " 0 0 1 " +Double.toString(angle);
      if (i != 5) { VRML_line =  VRML_line + ", "; }
```

```
    }

    VRML_line =  VRML_line + " ] }";
    writeVRMLline();

    /* write an empty line */
    VRML_line = " ";
    writeVRMLline();

    /* Create time sensor using 3.0 as interval */
    VRML_line = " DEF TIME" + laction.objectName +
    " TimeSensor { ";
    writeVRMLline();
    VRML_line = " cycleInterval 3.0 }";
    writeVRMLline();

    /* write an empty line */
    VRML_line = " ";
    writeVRMLline();

    /* Creating routes to rotate object */
     VRML_line = "ROUTE TOUCH" + laction.objectName +
     ".touchTime TO TIME" + laction.objectName + ".startTime";
    writeVRMLline();

    VRML_line = "ROUTE TIME" + laction.objectName +
    ".fraction_changed TO OI" + laction.objectName +
    ".set_fraction";
    writeVRMLline();

      VRML_line = "ROUTE OI" + laction.objectName +
      ".value_changed TO " + laction.objectName + ".rotation";
      writeVRMLline();

      /* write an empty line */
      VRML_line = " ";
      writeVRMLline();
}
catch ( Exception IOException )
{
   System.out.println("15" );
   System.out.println("I can't create or write to this file" );
}
}
/* ----------Continuous Scaling Action ----------------------*/
public void continuousScaling(Action laction) {
double x_scale;
double y_scale;
try
{
   /* write an empty line */
```

```
VRML_line = " ";
writeVRMLline();

/* Create position Interpolator  */
VRML_line = " DEF POS" + laction.objectName + "
PositionInterpolator { ";
writeVRMLline();

/* Approximate using 10 keys each time as in continuous
   translation case */
VRML_line =  " key [  " ;
for ( int i = 0;  i<= 9 ; i++ ) {
  VRML_line =  VRML_line + " " + Integer.toString(i);
}
VRML_line = VRML_line + " ]";
writeVRMLline();

/* write an empty line */
VRML_line = " ";
writeVRMLline();

/* Approximate key values */
VRML_line =  " keyValue [  " ;
for ( int i = 0; i <= 9 ; i++ ) {
  x_scale =  laction.laws[0].firstConstant*i +
  laction.laws[0].secondConstant / 25;
  y_scale =  laction.laws[1].firstConstant*i +
  laction.laws[1].secondConstant / 25;
  VRML_line =  VRML_line + Integer.toString(x_scale) + " " +
  Integer.toString(y_scale) + " 0,";
}
VRML_line =  VRML_line + " ] }";
writeVRMLline();

/* write an empty line */
VRML_line = " ";
writeVRMLline();

/* Create time sensor using 3.0 as interval */
VRML_line = " DEF TIME" + laction.objectName
            + " TimeSensor { ";
writeVRMLline();
VRML_line = " cycleInterval 3.0 }";
writeVRMLline();

/* write an empty line */
VRML_line = " ";
writeVRMLline();

/* Creating routes to animate object */
VRML_line = "ROUTE TOUCH" + laction.objectName
```

```
    + ".touchTime TO TIME" + laction.objectName
    + ".startTime";
    writeVRMLline();

    VRML_line = "ROUTE TIME" + laction.objectName +
    ".fraction_changed TO POS" + laction.objectName +
    ".set_fraction";
    writeVRMLline();

     VRML_line = "ROUTE POS" + laction.objectName +
     ".value_changed TO " + laction.objectName + ".set_scale";
     writeVRMLline();
}
catch ( Exception IOException )
{
    System.out.println("14" );
    System.out.println("I can't create or write to this file" );
}
}
/* ----------- Continuous Color Changing Action ---------------*/
public void continuousColorChanging(Action laction) {

}

/******************** Statement List ***************************/
/* -------------Assign statement to statement list ------------*/
public void assignStatementToList( ) {

statementList.insertAtBack(currentStatement);

}

/* -------------Clear Current Statement ------------------------*/
public void clearCurrentStatement() {
    currentStatement =  new Statement();
}

/* -------------Create Statements-------------------------------*/
public void createStatements() {
try
{
    statementList.first();
    while ( statementList.currentNode != null ) {

    /* Create events from the event array */
    for ( int i = 0;  i<statementList.currentNode.data.eventIndex;
    i++ ) {
        createEvent (statementList.currentNode.data.Events[i]);
    } /* end for event Array */

    /* Create conditions from the conditions array */
```

```
   for ( int i = 0;
   i<statementList.currentNode.data.conditionIndex; i++ ) {
      createCondition
         (statementList.currentNode.data.Conditions[i]);
   } /* end for condition Array */

   /* Create actions from the action array */
   for ( int i = 0; i<statementList.currentNode.data.actionIndex;
   i++ ) {
      createActions(statementList.currentNode.data.Actions[i]);
   } /* end for action Array */


   statementList.currentNode =  statementList.currentNode. next;
   } /* end while */
}

catch ( Exception IOException )
{
   System.out.println("16" );
   System.out.println("I can't create or write to this file" );
}
} /* end of method */

/************* PART 7 General procedures ***********************/
/************* Defining laws ***********************************/
/* ------------Assign first constant to the law ---------------*/

public void assignFirstConstantLaw( int lfirstConstant) {
   currentLaw.firstConstant = lfirstConstant;
}

/* -------------Assign Sign to the law-------------------------*/
public void assignSign( String lsign) {
   currentLaw.sign = lsign;
}

/* -------------Assign second constant to the law -------------*/
public void assignSecondConstantLaw( int lsecondConstant) {
   currentLaw.secondConstant = lsecondConstant;
}
:};
```

A4. Parser: Definition of terminal and non terminal symbols

The following statements define inside the parser the terminal and non terminal objects

```
terminal    OBJECT, OPBRAC, CLOSEBRAC;
terminal    STRING, FORM, TYPE, LINE;
terminal    RECTANGLE, ELLIPSE, POLYGON, GROUP;
terminal    POINTS, OPARENT, COMMA, CLOSEPARENT;
terminal    NUMBER, CONTENTS, FILE, COLOUR, COMPONENT;

terminal EVENT, START;
terminal MOUSE_CLICK, KEYBOARD_PRESS, TIME;
terminal POINT, IS, VIS, AS, IT, POS, ANGLE;
terminal SCALE, TRUE, FALSE;
terminal ACTIVE, IDLE, SUSPENDED, MEET, NO_MEET;
terminal ANY, SEQ, TIMES, CONDITION;
terminal GREATER, LESS, GREATER_THAN, LESS_THAN, EQUAL,
         NOT_EQUAL;
terminal OR, AND;
terminal ACTIONS, STOP, RESUME, PAUSE, HIDE, SHOW;
terminal PARENT, ADD_MEMBER, REMOVE_MEMBER, DESTROY,
         TRANSFORMATION;
terminal TRANSLATION, ROTATION, SCALING, COLOR_CHANGING;
terminal C_TRANSLATION, C_ROTATION, C_SCALING, C_COLOR_CHANGING;
terminal PLUS, MUL, T, NO_ACTION;

non terminal objects, object, object_name;
non terminal form, geometry, type, type_value;
non terminal point_list, points, point, coord_x, coord_y ;
non terminal contents, contents_value, file_name, contents_name;
non terminal colour, r, g, b, composition;
non terminal empty;

non terminal IAN, scenario, statements;
non terminal statement, event, conditions, scenario_actions;
non terminal event_description, simple_event, complex_event;
non terminal user_interaction_event, intra_object_event,
             inter_object_event, time_event;
non terminal user_action,vis_value;
non terminal as_value, it_value, pos_value, angle_value,
             scale_value, scale_x, scale_y;
non terminal pos_x, pos_y, relation, parameters, count,
             simple_events;
non terminal condition_description, simple_condition,
             complex_condition;
non terminal state_condition, time_condition, equality_operator,
             comparison_operator;
non terminal and_condition, method, law;
non terminal actions_description, simple_action, simple_actions,
```

```
                sequence_action, action_object_name;
non terminal    transformation, continuous_transformation,
                discrete_transformation;
non terminal    action_pos_value, action_pos_x, action_pos_y,
                action_vis_value, dt;
non terminal    event_object_name;
non terminal    angle_law, angle_constant1, angle_constant2;
non terminal    translation_law, translation_constant1,
                translation_constant2;
non terminal    scaling_law, scaling_constant1, scaling_constant2;
```

A5. Parser: Grammar part of the parser

In this appendix the last part of the parser, the grammar definition is presented.

```
IAN ::= {: Create_Output_File(); :} objects
        {: Create_Objects(); :} scenario
        {: Close_Output_File(); :};

objects ::= object {: Assign_Object_To_ListNode();
                      Clear_Scenario_Object(); :}objects
          | empty;

object ::= OBJECT object_name
           OPBRAC  form colour composition CLOSEBRAC;

object_name ::= STRING:parser_object_name
{: Assign_Object_Name_Value(parser_object_name.toString()); :};

form ::= FORM OPBRAC geometry contents CLOSEBRAC;
geometry ::= type point_list;
type ::= TYPE type_value;

type_value ::= LINE {: Assign_Type_Value("LINE"); :}
             | RECTANGLE {: Assign_Type_Value("RECTANGLE"); :}
             | ELLIPSE {: Assign_Type_Value( "ELLIPSE"); :}
             | POLYGON {: Assign_Type_Value("POLYGON"); :}
             | GROUP{: Assign_Type_Value( "GROUP"); :};

point_list ::= POINTS points;
points ::= point points | empty;
point ::= OPARENT coord_x COMMA coord_y CLOSEPARENT;
coord_x ::= NUMBER: parser_coord_x
           {:Assign_coordinate_x
           (Integer.parseInt(parser_coord_x.toString())); :};
coord_y ::= NUMBER: parser_coord_y
           {: Assign_coordinate_y
           (Integer.parseInt(parser_coord_y.toString())); :};

contents ::= CONTENTS contents_value | empty;
contents_value ::= FILE file_name | contents_name;
```

```
contents_name ::= STRING:parser_contents
                  {: assignContents(parser_contents.toString());
                  assignIsfile("CONTENTS"); :};
file_name ::=  STRING:parser_filename{:
assignContents(parser_filename.toString());
                assignIsfile("FILE"); :};
colour ::= COLOUR r:r g:g b:b | empty;
r ::= NUMBER: parser_r
   {: Assign_colour_r(Integer.parseInt(parser_r.toString())); :};
g ::= NUMBER: parser_g
   {: Assign_colour_g(Integer.parseInt(parser_g.toString())); :};
b ::= NUMBER: parser_b
   {: Assign_colour_b(Integer.parseInt(parser_b.toString())); :};
composition ::= COMPONENT {: Assign_Object_To_ListNode();
                Clear_Scenario_Object(); :}
                OPBRAC objects CLOSEBRAC
                | empty;

scenario ::= statements {: createStatements(); :};
statements ::= statement statements | empty ;
statement ::= event {: assignEventToEventArray();
                       clearCurrentEvent(); :}
              conditions {: assignConditionToConditionArray();
                       clearCurrentCondition(); :}
              scenario_actions {: assignStatementToList();
                       clearCurrentStatement();:};

event ::= EVENT  event_description;

event_description ::=  simple_event
                       | complex_event;

simple_event ::= user_interaction_event
                 | intra_object_event
                 | inter_object_event
                 | time_event;


user_interaction_event ::= user_action
                           {:assignEventObjectName("IAN"); :}
                           | event_object_name POINT user_action;

user_action ::= MOUSE_CLICK {: assignEventType("MOUSE_CLICK"); :}
                | KEYBOARD_PRESS
                  {: assignEventType("KEYBOARD_PRESS"); :};

intra_object_event ::=  event_object_name POINT COLOUR
             {: assignEventType("COLOUR_EXPRESSION"); :} IS r g b
           | event_object_name POINT CONTENTS
             {: assignEventType("CONTENTS_EXPRESSION"); :}
             IS STRING
```

```
          | event_object_name POINT VIS
            {: assignEventType("VISIBILITY_EXPRESSION"); :}
               IS vis_value
          | event_object_name POINT AS
            {: assignEventType("ACTIVITY_STATUS_EXPRESSION"); :}
               IS as_value
          | event_object_name POINT IT
            {: assignEventType("INTERNAL_TIME_EXPRESSION"); :}
               IS it_value
          | event_object_name POINT POS
            {: assignEventType("POSITION_EXPRESSION"); :}
               IS pos_value
          | event_object_name POINT ANGLE
            {: assignEventType("ANGLE_EXPRESSION"); :}
               IS angle_value
          | event_object_name POINT SCALE
            {: assignEventType("SCALE_EXPRESSION"); :}
               IS scale_value;

vis_value ::= TRUE | FALSE ;
as_value ::= ACTIVE
           | IDLE
           | SUSPENDED;
it_value ::= NUMBER;
pos_value ::= OPARENT pos_x COMMA pos_y CLOSEPARENT;
pos_x ::= NUMBER;
pos_y ::= NUMBER;
angle_value ::= NUMBER;
scale_value ::= OPARENT scale_x COMMA scale_y CLOSEPARENT;
scale_x ::= NUMBER;
scale_y ::= NUMBER;

inter_object_event ::= relation OPARENT parameters CLOSEPARENT;

relation ::= MEET {: assignEventType("MEET_EXPRESSION"); :}
         | NO_MEET {: assignEventType("NO_MEET_EXPRESSION"); :};
parameters ::= event_object_name COMMA event_object_name;

time_event ::= START {:assignEventObjectName("IAN");
                       assignEventType("START"); :}
             | TIME {:assignEventObjectName("IAN");
                       assignEventType("TIME"); :} IS NUMBER ;

complex_event ::= ANY {:assignEventType("ANY"); :}
                  OPARENT count  COMMA simple_events CLOSEPARENT
                | SEQ {:assignEventType("SEQ"); :}
                  OPARENT simple_events CLOSEPARENT
                | TIMES {:assignEventType("TIMES"); :}
                  OPARENT count COMMA simple_event CLOSEPARENT;

simple_events ::= simple_event simple_events | empty;
```

```
count ::= NUMBER:parserCount
   {: assignCount(Integer.parseInt(parserCount.toString())); :};

event_object_name ::= STRING:parser_event_object_name
 {: assignEventObjectName
    (parser_event_object_name.toString()); :};


conditions ::= CONDITION condition_description | empty;
condition_description ::= simple_condition | complex_condition;

simple_condition ::= state_condition
                   | inter_object_event
                   | time_condition;

state_condition ::= object_name POINT COLOUR
         {: assignConditionType("COLOUR_CONDITION"); :}
          equality_operator r g b
       | object_name POINT CONTENTS
         {: assignConditionType("CONTENTS_CONDITION"); :}
          equality_operator STRING
       | object_name POINT VIS
         {: assignConditionType("VISIBILITY_CONDITION"); :}
          equality_operator vis_value
       | object_name POINT AS
         {: assignConditionType("ACTIVITY_STATUS_CONDITION"); :}
          equality_operator as_value
       | object_name POINT IT
         {: assignConditionType("INTERNAL_TIME_CONDITION"); :}
          comparison_operator NUMBER
        | object_name POINT POS
          {: assignConditionType("POSITION_CONDITION"); :}
           comparison_operator pos_value
        | object_name POINT ANGLE
          {: assignConditionType("ANGLE_CONDITION"); :}
           comparison_operator  angle_value
        | object_name POINT SCALE
          {: assignConditionType("SCALE_CONDITION"); :}
           comparison_operator scale_value;

equality_operator ::= EQUAL | NOT_EQUAL;
comparison_operator ::= EQUAL
                      | NOT_EQUAL
                      | GREATER
                      | LESS
                      | GREATER_THAN
                      | LESS_THAN;

time_condition ::= TIME
                   {: assignConditionType("TIME_CONDITION"); :}
                   comparison_operator NUMBER;
```

```
complex_condition ::= and_condition OR and_condition
                    | simple_condition OR simple_condition
                    | and_condition
                    | empty;
and_condition ::= OPARENT simple_condition
                  AND simple_condition CLOSEPARENT;

scenario_actions ::=  ACTIONS actions_description | NO_ACTION;
actions_description ::= simple_action | sequence_action;

simple_action ::= action_object_name POINT method
                  {: assignActionToActionArray();
                     clearCurrentAction(); :};
action_object_name ::=  STRING:parser_action_object_name
                  {: assignActionObjectName
                     (parser_action_object_name.toString());  :};

method ::= START {: assignMethodName("START"); :}
             OPARENT action_pos_value
             COMMA action_vis_value CLOSEPARENT
           | STOP {: assignMethodName("STOP");  :}
             OPARENT CLOSEPARENT
           | RESUME {: assignMethodName("RESUME");  :}
             OPARENT CLOSEPARENT
           | PAUSE {: assignMethodName("PAUSE");  :}
             OPARENT CLOSEPARENT
           | HIDE {: assignMethodName("HIDE");  :}
             OPARENT CLOSEPARENT
           | SHOW {: assignMethodName("SHOW");  :}
             OPARENT CLOSEPARENT
           | ADD_MEMBER {: assignMethodName("ADD_MEMBER");  :}
             OPARENT CLOSEPARENT
           | REMOVE_MEMBER {:assignMethodName("REMOVE_MEMBER"); :}
             OPARENT CLOSEPARENT
           | DESTROY {: assignMethodName("DESTROY");  :}
             OPARENT CLOSEPARENT
            | transformation;

transformation ::= discrete_transformation
                 | continuous_transformation;
discrete_transformation ::=
TRANSLATION {: assignMethodName("DISCRETE_TRANSLATION"); :}
            OPARENT action_pos_value CLOSEPARENT
| ROTATION {: assignMethodName("DISCRETE_ROTATION"); :}
            OPARENT angle_value CLOSEPARENT
| SCALING {: assignMethodName("DISCRETE_SCALING");  :}
            OPARENT scale_value CLOSEPARENT
| COLOR_CHANGING {:assignMethodName("DISCRETE_COLOR_CHANGING");:}
            OPARENT r g b CLOSEPARENT;
```

```
continuous_transformation ::=
C_TRANSLATION {: assignMethodName("CONTINUOUS_TRANSLATION"); :}
      OPARENT translation_law COMMA translation_law CLOSEPARENT
| C_ROTATION {: assignMethodName("CONTINUOUS_ROTATION");  :}
      OPARENT angle_law CLOSEPARENT
| C_SCALING {: assignMethodName("CONTINUOUS_SCALING");  :}
      OPARENT scaling_law COMMA scaling_law CLOSEPARENT
| C_COLOR_CHANGING
      {:assignMethodName("CONTINOUS_COLOR_CHANGING");:}
      OPARENT law COMMA law COMMA law CLOSEPARENT;

translation_law ::= translation_constant1 TIME PLUS
                    {: assignSign("+"); :} translation_constant2;

translation_constant1 ::= NUMBER: parser_translation_constant1
    {: assignFirstConstantLaw
    (Integer.parseInt
    (parser_translation_constant1.toString())); :};

translation_constant2 ::= NUMBER: parser_translation_constant2
    {: assignSecondConstantLaw
    (Integer.parseInt
    (parser_translation_constant2.toString()));
    assignActionTranslationLaw(); :};

angle_law ::= angle_constant1 TIME PLUS
              {: assignSign("+"); :} angle_constant2;
angle_constant1 ::= NUMBER: parser_angle_constant1
    {: assignFirstConstantLaw
    (Integer.parseInt
    (parser_angle_constant1.toString())); :};
angle_constant2 ::= NUMBER: parser_angle_constant2
    {: assignSecondConstantLaw
    (Integer.parseInt(parser_angle_constant2.toString()));
    assignActionAngleLaw(); :};

scaling_law ::= scaling_constant1 TIME PLUS
    {: assignSign("+"); :} scaling_constant2;
scaling_constant1 ::= NUMBER: parser_scaling_constant1
    {: assignFirstConstantLaw
    (Integer.parseInt(parser_scaling_constant1.toString())); :};
scaling_constant2 ::= NUMBER: parser_scaling_constant2
    {: assignSecondConstantLaw
    (Integer.parseInt(parser_scaling_constant2.toString()));

     assignActionScalingLaw(); :};

action_pos_value ::= OPARENT action_pos_x
                     COMMA action_pos_y CLOSEPARENT;
action_pos_x ::= NUMBER: parser_action_pos_x
   {: assignActionPosX
```

```
    (Integer.parseInt(parser_action_pos_x.toString())); :};
action_pos_y ::= NUMBER: parser_action_pos_y
    {: assignActionPosY
    (Integer.parseInt(parser_action_pos_y.toString())); :};
action_vis_value ::= TRUE {: assignActionVisibility("TRUE"); :}
                   | FALSE {: assignActionVisibility("FALSE"); :};

sequence_action ::= SEQ OPARENT simple_actions  CLOSEPARENT;
simple_actions ::=  simple_action dt simple_actions | empty;
dt ::= NUMBER;
empty ::=  ;
```

Appendix B: ODL and OQL reference. Types in ODL

ODL offers the database designer a type system similar to that found in C or other conventional programming languages. A type system is built from a basis of types that are defined by themselves and certain recursive rules whereby complex types are built from simpler types. In ODL, the basis consists of:

1. *Atomic types*: integer, float, character, character sting, Boolean, and enumerations. The latter are list of names declared to be synonyms for integers. An example of enumeration can be: enum Colours { green, bleu, beige } eyesColor.

2. *Interface types*: which represent types that are actually structures, with components for each attributes and relationships of that interface. These names represent complex types defined using the rules below but we can think of them as basic types.

The basic types are combined into structured types using the following ***type constructors***:

1. *Set*: If T is any type, then Set<T> denotes the type whose values are all finite sets of elements of type T.
2. *Bag*: If T is any type, then Bag<T> denotes the type whose values are bags or multisets of type T. A bag allows an element to appear more than once. For example { 1, 2, 1 } is a bag but not a set because 1 appears more than once.
3. *List*: If T is any type then List<T> denotes the type whose values are finite lists of zero or more elements of type T. As a special case, the type string is a shorthand for the type List<char>.
4. *Array:* If T is a type, and i is an integer, then array<T,i> denotes the type whose elements are arrays of i elements of type T. For example Array<char, 10> denotes character strings of length 10.
5. *Structures:* if T1, T2, T3,….,Tn are types, and F1, F2,…Fn are names of fields, then
   Struct N { T1 F1, T2 F2 ,..., Fn Tn }
   denotes the type named N whose elements are structures with n fields. The *ith* field is named Fi and has type Ti.

The first four types set, bag, list and array are called *collection types*. There are different rules about which types may be associated with attributes and which with relationships.

- The type of an attribute is built starting with either an atomic type or a structure whose fields are atomic types. Then we may optionally apply a collection type to the initial atomic type or structure.
- The type of a relationship is either an interface type or a collection type applied to an interface type.

It is important to remember that interface types may not appear in the type of an attribute and atomic types do not appear in the type of the relationship. It is this distinction that separates attributes and relationships We must finally notice that there is a difference in the way complex types are built for attributes and relationships so each allows an optional collection type as the final operator, but only attributes allow a structure type.

The following example is very helpful in order to understand the possible types of attributes and relationships.

Some of the possible types of attributes are:

```
1. integer
2. Struct N { string field1, integer field2 }
3. List<real>
4. Array<Struct N { string field1, integer field2 }.
```

Example 1 is an atomic type, 2 is a structure of atomic types 3 is a collection of an atomic type and 4 a collection of structures built from atomic types. These are the only four possibilities for attribute types.

Now suppose the interface names I1 and I2 are available basic types. Then we may construct relationship types such as I1 or Bag<I2>. However the following are illegal types:

1. Struct N {I1 field1, I2 field2}. Relationships types cannot involve structures.
2. Set<integer>. Relationships types cannot involve atomic types.
3. Set <Array<I1> Relationships types cannot involve two applications of collection types (neither can attribute types).

References

Papers – Technical reports (http://www.dbnet.ece.ntua.gr/~michalis/publications.html)

1. [Vazirgiannis 1999], "Interactive Multimedia Documents: Modeling, Authoring and Implementation Experiences", Springer-Verlag, LNCS Series, October 1999.
2. [Vazirgiannis-Sellis 1998] M. Vazirgiannis, T. Sellis ***Multimedia Data Base Management Systems: Modelling and Querying Interactive Multimedia Documents*** Paper 1998 to appear in "Advanced Databases" (Editors, M. Piattini, O. Diaz), Artech House
3. [Vazirgiannis-Sellis 1997] M.Vazirgiannis, T. Sellis ***"Event and Action representation and composition for multimedia Application scenario modelling "*** Department of Electrical Engineering 1997, ERCIM workshop on Interactive Distributed Multimedia Systems and Services, BERLIN 3/1996.
4. [Vodislav-Vazirgiannis 1999] Dan Vodislav, M. Vazirgiannis ***Generic Object Modelling for Animation Contexts*** Paper 1999.
5. [Vazirgiannis 1997] ***Interactive Multimedia Documents Modelling and Rendering: Implementation experiences*** M Vazirgiannis and implementation team 1997.
6. [Vazirgiannis-Boll 1996] ***Events in Interactive Multimedia applications: Modelling and Implementation design***_Michalis Vazirgiannis, Susanne Boll December 1996 in the proceedings of IEEE International Conference on Multimedia Computing and Systems (ICMCS'97), June 1997, Ottawa, Canada
7. [Guting and al. 1998] Ralf Harmut Guting, Michael H. Bohlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, Michalis Vazirgiannis, ***A Foundation for Representing and Querying Moving Objects,*** FernUniversitat September 1998 to appear in ACM-TODS journal.
8. [Vakaloudis 1998] Alex Vakaloudis, Babis Theodoulidis ***"The Storage and Querying of 3D Objects for the dynamic Composition of VRML Words"*** TimeLab Dept. Of Computation UMIST 1998

Papers –Technical reports about other animation models

9. [BV95] A. Del Bimbo, E. Vicario, ***" Specification by example of virtual agents behavior",*** IEEE Transactions on Visualization and Computer Graphics, vol 1, no 2, December 1995.
10. [CrK94] J.F. Cremer, J.K. Kearney, ***"Scenario authoring for virtual environment",*** Proc. Of IMAGE VII Conf., Tucson, AZ, June 12-17 1994.
11. [DiM99] A. Diaz and R. Melster. ***"Patterns for modeling behavior in virtual environment applications"***, Second Workshop on Hypermedia Development: design patterns in hypermedia (in conjunction with Hypertext 99). Darmstadt, Germany, February 1999.

12. [DoH97] J. Dollner and K. Hinrics, ***"Object-oriented 3D modeling, animation and interaction"***, in The Journal of Vizualization and Computer Animation, vol. 8:33-64, 1997.
13. [EGSV99] M. Erwig, R. H. Gueting, M. Schneider, M. Vazirgiannis, ***"Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases",*** GeoInformatica Journal, Kluwer Publishers, 1999.
14. [MDG98] R. Melster, A. Diaz and B. Groth. ***"SCORE - The virtual museum, development of a distributed, object-oriented system for 3D real-time visualization".*** Technical report 1998-15, TU Berlin, Germany.October 1998.
15. [NB95] Marc Najork, Mark Brown , ***"Obliq-3d: A high Level, Fast Turnaround 3D Animation System"***, IEEE Transactions on Visualization and Computer Graphics, vol 1, no 2, June 1995,

Books

16. [Steinmetz 1995] Ralf Steinmetz and Klara Nahrstedt ***Multimedia: Computing, Communications & Applications*** , Prentice Hall 1995
17. [Sommerville 1996] Ian Somerville ***Software Engineering*** Addison-Wesley 1996
18. [Roberts 1998] Jason Roberts Director ***The official guide to Macromedia Director, Lingo, and Shockwave,*** Macromedia Press 1998.
19. [Carey 1997] Rick Carey, Gavin Bell ***The Annotated VRML 2.0 Reference Manual***, Addison–Wesley 1997.
20. [Abiteboul-Hull-Vianu 1995] ***Foundations of Databases*** Abiteboul, Hull, Vianu Addison Wesley Publishing Company 1995
21. [Ullman and al. 1997] ***A first Course In Database Systems*** Jeffrey D. Ullman, Jennifer Widom Prentice Hall 1997

22. [Berk 1997] Berk Elliot ***Jlex: A lexical analyser generator for Java*** Department of Computer Science, Princeton University Version 1.2, May 5, 1997 Internet address http://www.cs.princeton.edu/~appel/modern/java/JLex/
23. [Hudson 1999] Hudson E. Scott ***Cup Parser Generator for Java*** Graphics Visualisation and Usability Center, Georgia Institute of Technology July 1999 Internet address http://www.cs.princeton.edu/~appel/modern/java/CUP/
24. [Deitel 1997] P.J. Deitel and H. M. Deitel "Java. How to program" Prentice Hall International 1997.
25. [Weiss 1998] Mark Allen Weiss ***"Data structures and Problem solving using Java"*** Florida International University Addison Wesley 1998
26. [O2 tech 199-] O2 Technology ***"The O2 system manuals***" release 4.6 April 1996