ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ

ΙΔΡΥΜΑ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΠΟΛΥΜΕΣΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ 3D ΠΕΡΙΒΑΛΛΟΝΤΟΣ ΔΡΑΣΗΣ ΚΑΙ ΣΥΝΕΡΓΑΣΙΑΣ ΑΠΟΜΑΚΡΥΣΜΕΝΩΝ ΧΡΗΣΤΩΝ

ΛΑΣΗΘΙΩΤΑΚΗ ΜΑΡΘΑ

Α.Μ.: 549

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:

Μαλάμος Αθανάσιος

ΗΡΑΚΛΕΙΟ 2008

Περιεχόμενα

1. ΔΗΜΙΟΥΡΓΙΑ ΤΗΣ XML

Η ανάπτυξη της XML ξεκίνησε το 1996. Από το Φεβρουάριο του 1998 η XML αποτελεί Σύσταση του W3C. Ίσως, λοιπόν να θεωρήσετε ότι η XML δεν έχει ωριμάσει ακόμα τεχνολογικά. Στην πραγματικότητα, όμως, η τεχνολογία XML δεν είναι τόσο καινούρια. Πριν από την XML υπήρχε η SGML, η οποία αναπτύχθηκε στις αρχές της δεκαετίας του '80, τυποποιήθηκε από τον ISO το 1986, και χρησιμοποιήθηκε ευρέως σε προγράμματα με εκτεταμένη τεκμηρίωση. Η ανάπτυξη της HTML ξεκίνησε το 1990. Οι σχεδιαστές της XML επέλεξαν τα καλύτερα τμήματα της SGML, χρησιμοποίησαν την εμπειρία που είχαν αποκτήσει κατά την ανάπτυξη της HTML και παρήγαγαν μία γλώσσα η οποία δεν είναι λιγότερο ισχυρή από την SGML αλλά είναι πιο κανονικοποιημένη και πολύ πιο εύχρηστη. Βλέπετε, λοιπόν, είναι δύσκολο να διακρίνει κανείς την εξελικτική από την επαναστατική πρόοδο... Αξίζει να σημειωθεί, τέλος, ότι ενώ η SGML χρησιμοποιείται κυρίως για τεχνική τεκμηρίωση, και πολύ λιγότερο για δεδομένα άλλου είδους, για την XML ισχύει ακριβώς το αντίθετο.

Η XML σχεδιάστηκε με πρωταρχικό στόχο να επιτρέψει τη χρήση της SGML στο διαδίκτυο. Οι αρχικές προσπάθειες κινήθηκαν προς τη δημιουργία μιας νέας έκδοσης της SGML κατάλληλης για το Web, αλλά οδήγησαν τελικά στη δημιουργία μιας νέας γλώσσας. Η XML στην ουσία, αποτελεί μια συντετμημένη έκδοση της SGML έτσι ώστε να είναι ευκολότερος ο ορισμός νέων τύπων εγγράφων και η δημιουργία προγραμμάτων συντακτικής ανάλυσης των εγγράφων (parsers). Παραλείπονται τα πολυπλοκότερα και λιγότερο χρησιμοποιούμενα τμήματα της SGML εξακολουθεί όμως να είναι SGML, συνεπώς συμβατή με υπάρχουσες εφαρμογες και εργαλεία της SGML. Είναι χαρακτηριστικό ότι το reference manual της SGML απαριθμεί 600 σελίδες, σε αντίθεση με το αντίστοιχο της XML που απαριθμεί μόλις 26 σελίδες.

Ο βασικός στόχος της XML είναι να περιγράψει τη δομή ενός εγγράφου αποδίδοντας σημασιολογική πληροφορία στα περιεχόμενα του. Το κομμάτι της παρουσίασης του κειμένου το αναλαμβάνουν συμπληρωματικές τεχνολογίες, όπως η CSS (Cascading Style Sheets) και η XSL (XML Style Language).

1.1 eXtensible Markup Language - XML

Η XML είναι ένα πρότυπο του οργανισμού W3C για markup έγγραφα. Η XML περιγράφει μια κατηγορία αντικειμένων δεδομένων που καλούνται XML έγγραφα και εν μέρει περιγράφει τη συμπεριφορά προγραμμάτων υπολογιστών που τα επεξεργάζονται. Τα έγγραφα XML αποτελούνται από μονάδες αποθήκευσης που ονομάζονται οντότητες (entities), που περιέχουν είτε αναλυμένα λεκτικά (parsed) είτε μη αναλυμένα λεκτικά (unparsed) δεδομένα. Τα parsed δεδομένα αποτελούνται από χαρακτήρες που συνθέτουν δεδομένα χαρακτήρων ή markup. Το markup κωδικοποιεί την περιγραφή της διάταξης και λογικής δομής του περιεχομένου του εγγράφου. Επιπλέον η XML παρέχει ένα μηχανισμό επιβολής περιορισμών στη διάταξη και τη λογική δομή.

Η XML είναι μια meta-markup γλώσσα με την έννοια ότι δίνει τη δυνατότητα στο χρήστη να δημιουργεί τα δικά του tags (ετικέτες) σύμφωνα με τις ανάγκες της εφαρμογής. Αυτό σημαίνει ότι η XML δεν έχει σταθερό σύνολο tags και elements που να καλύπτουν τις ανάγκες κάθε εφαρμογής ή χρήστη. Το markup σε ένα έγγραφο XML περιγράφει τη δομή

του εγγράφου καθώς και τη σημασιολογία (semantics) του εγγράφου. Η XML επιτρέπει στους μηχανικούς λογισμικού να ορίζουν σαφώς τα απαραίτητα elements και να κωδικοποιούν τις σχέσεις τους. Η XML καθορίζει τη σύνταξη που πρέπει να ακολουθήσουν γλώσσες markup κάθε περιοχής γνώσης, όπως οι MusicML, MathML, GML και SVG. Παρ' ότι είναι αρκετά ευέλικτη στον ορισμό των elements, είναι μάλλον αυστηρή από άλλες απόψεις. Παρέχει κανόνες γραμματικής για τα XML έγγραφα περιγράφοντας την κανονική τους δομή που επιτρέπει την ανάπτυξη XML parsers (λεκτικών αναλυτών) που μπορούν να διαβάσουν κάθε XML έγγραφο. Τα έγγραφα που ικανοποιούν αυτή τη γραμματική θεωρούνται well-formed.

1.2 ΣΥΝΤΑΞΗ ΤΗΣ XML

Ενα αρχείο XML δεν αποτελεί τίποτα παραπάνω από ένα αρχείο απλού κειμένου (plain text) το οποίο ακολουθεί αυστηρή σύνταξη και δομή. Η XML χρησιμοποιεί εν γένει κωδικοποίηση Unicode (UTF-8 ή UTF-16 Unicode Encoding) προσφέροντας έτσι δυνατότητα καταγραφής χαρακτήρων από οποιαδήποτε γλώσσα και αλφάβητο. Στον συνδυασμό των παραπάνω χαρακτηριστικών στηρίζεται άλλωστε και η διαπλατφομικότητα και διαλειτουργικότητατα που προσφέρει η XML, καθώς οποιαδήποτε εφαρμογή οποιουδήποτε υπολογιστικού συστήματος μπορεί απ'ευθείας να διαβάσει και να επεξεργαστεί ένα αρχείο κειμένου χωρίς να έχει πρόβλημα να ερμηνευσει τους χαρακτήρες και τα δεδομένα που περιέχει.

Στην XML χρησιμοποιούνται ετικέτες (tags) (λέξεις μέσα σε γωνιακές αγκύλες '<' και '>') και γνωρίσματα (τύπου όνομα = "τιμή") μόνο για να οριοθετήθούν κομμάτια δεδομένων ενώ η ερμηνεία των δεδομένων πραγματοποιείται από την εφαρμογή που τα διαβάζει.

1.2.1 ΔΟΜΙΚΑ ΣΤΟΙΧΕΙΑ

Element

Τα έγγραφα της XML είναι δέντρα που αποτελούνται από στοιχεία (*elements*). Κάθε στοιχείο ορίζεται από την ετικέτα αρχής (*start tag*), το ίδιο το δεδομένο και την ετικέτα τέλους (*end tag*) που πλέον είναι απαραίτητο να υπάρχει.

```
<city>Larisa</city>
```

Παράδειγμα: ένα στοιχείο με κείμενο

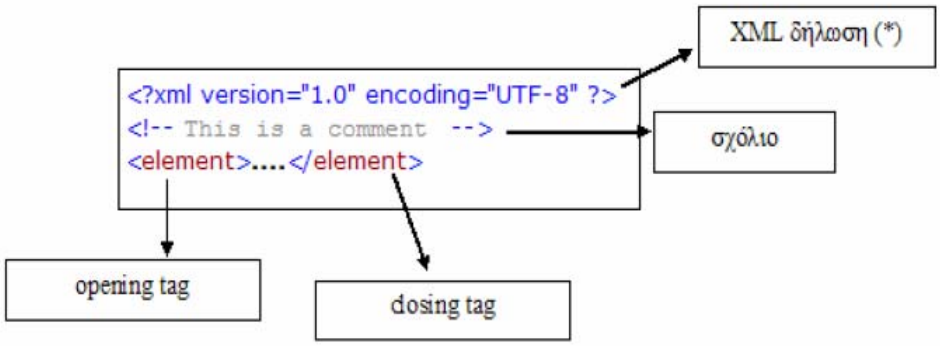
Το περιεχόμενο μπορεί να είναι κείμενο ή άλλα στοιχεία Κενά στοιχεία μπορούν να γραφούν συντομογραφικά με μια μόνο ετικέτα . Επιτρέπονται άπειρα επίπεδα εμφωλευμένων tags ενώ απαγορεύεται οι ετικέτες να ξεκινούν με 'XML' είτε σε πεζά είτε σε κεφαλαία.Με την ενθυλάκωση (nesting) των tags δίνεται η δυνατότητα να κατασκευασθούν στοιχεία με πολύπλοκη εσωτερική δενδρική δομή, π.χ.

```
<?xml version="1.0" encoding="US-ASCII" ?>
<city_info>
    <name>Larisa</name>
    <area_code>241</area_code>
    <latitude>39.38</latitude>
```

```
    <longitude>-22.25</longitude>
    <country>Greece</country>
</city_info>
```
Παράδειγμα: στοιχείο με περιεχόμενο άλλα στοιχεία

```
<alumnus />
```
Παράδειγμα: κενό στοιχείο

Η πρώτη γραμμή ενός XML εγγράφου <?xml version="1.0"?> αποτελεί εντολή προς τον parser (όλες οι εντολές αυτές περικλείονται από <? ?>), δηλ. το πρόγραμμα πού



αποκωδικοποιεί την δομή του XML. Μέσα σ΄αυτήν την εντολή το 'version' είναι μεταβλητή ιδιοτήτων. Το string xml είναι δεσμευμένη λέξη και δεν μπορεί να χρησιμοποιηθεί για attribute, ούτε για tag.

Μεταβλητές Ιδιοτήτων-Attributes

Κάθε στοιχείο (element) δυνατόν να έχει και ιδιότητες (attributes) οι μεταβλητές που παρέχουν επιπλέον πληροφορίατων οποίων μαζύ με τις τιμές τους συντάσσονται σύμφωνα με το παράδειγμα Ένα element μπορέι να έχει μια ή περισσότερες ιδιότητες, τα attributes τα παρέχουν επιπλέον πληροφορία αναφορικά με τα στοιχεία.

Παράδειγμα:

```
<movie type="comedy" rating="for adults" year="1998"> …. </movie>
```

Το tag (movie) συνοδεύεται από ότι ο χρήστης θέλει να περιλάβει σαν ιδιότητες (comedy, rating, year). Κάθε μεταβλητή ιδιότητας φέρει σαν τιμή ένα text string. Η σημασία και χρήση των attributes είναι θέμα του χρήστη ή της εφαρμογής που επεξεργάζεται το XML. Κάθε μεταβλητή ιδιοτήτων θα μπορούσε να αποτελέσει υποστοιχείο (sub element), δηλ. το παραπάνω παράδειγμα θα μπορούσε να γραφεί και με τη μορφή εμφωλευμένων elements:

```
<movie>
        <type>comedy</type>
        <rating>for adults</rating>
        <year>1998</year>
</movie>
```

Το τι θα προτιμηθεί έγκειται συνήθως στην εμπειρία του προγραμματιστή.

XML-Namespaces

Μια σημαντική ιδιότητα των XML-εγγράφων είναι ότι μπορούν να χρησιμοποιηθούν ώστε να συνθέσουν ένα νέο έγγραφο. Ο μηχανισμός των namespaces έχει ως σκοπό την αντιμετώπιση του προβλήματος της σύγκρουσης elements με κοινά ονόματα, τα οποία μπορεί να προέρχονται από διαφορετικά έγγραφα. Συγκεκριμένα ένα element συμπληρώνεται με κάποιο επιπρόσθετο αναγνωριστικό που να του αποδίδει τη μοναδικότητα. Ως αναγνωριστικό χρησιμοποιείται κάποιο Uniform Resource Identifier (URI). Για παράδειγμα στην αρχή ενός XML εγγράφου μπορεί να οριστούν τα namespaces ως εξής :

xmlns:it= "http://www.italian_teams"
xmlns:ru= "http://www.russian_teams.com"
και στο εξής να γίνεται αναφορά στο έγγραφο μέσω it:teams, ru:teams .

XML-Schema

Το XML-Schema είναι μια προδιαγραφή στην οποία κατέληξε η απόπειρα της κοινότητας της XML υπό την ηγεσία του W3C για τη δημιουργία μιας μετα-γλώσσας για την περιγραφή της δομής των XML εγγράφων και των τύπων δεδομένων σε αυτά. Τα XML-schemas είναι και αυτά εκφρασμένα σε XML. Με τον τρόπο αυτό εξαλείφεται η ανάγκη ύπαρξης ειδικών parsers που να τα διαβάζουν. Η ορολογία-λεξιλόγιο του XML-Schema καθορίζεται και αυτή χρησιμοποιώντας κάποιο schema. Το XML-Schema αποτελεί εκτός από τρόπο περιγραφής και τρόπο επικύρωσης της καλής μορφής (well-formedness) και εγκυρότητας (validity) ενός XML εγγράφου.

Entity

Οι οντότητες είναι αφλαριθμητικά που χρησιμοποιούνται όπως οι μακροεντολές, και αναπαριστούν ένα συχνά εμφανιζόμενο κείμενο.

Παράδειγμα:

<!Entity message "Welcome">
Με αυτόν τον τρόπο, όταν γράφουμε &message θα ισοδυναμεί με "Welcome"

XML Data Island

Δεν είναι τίποτε άλλο από ένα XML document μέσα σε ένα αρχείο HTML. Η σύνταξή του είναι όπως ενός XML document μόνο που αυτό δηλώνεται και αναγνωρίζεται μέσα από την XML μέσω ενός XML id.

```
<XML ID="XMLid1">
        <customer>
                <name>Mark Hanson</name>
                <custID>81422</custID>
```

```
        </customer>
    </XML>
```

Εναλλακτικοί τρόποι σύνταξης είναι με εξωτερική αναφορά ή με τη χρήση του tag script.

XMLid1.XMLDocument.documentElement.childNodes.item(0).text

1.3 XML PARSERS

Μια εφαρμογή που χρησιμοποιεί έγγραφα XML πρέπει να είναι σε θέση να αναλύει το έγγραφο στα στοιχεία που το συνθέτουν ώστε να αποσπά από αυτό και την απαραίτητη από το έγγραφο πληροφορία. Αυτό γίνεται μέσω της διαδικασίας του parsing. Το parsing αναλύει το κείμενο σε επιμέρους τμήματα τα οποία είναι αναγνωρίσιμα (start tags, end tags, attributes και τιμές αυτών, κείμενο, σχόλια). Η εφαρμογή μπορεί να κάνει χρήση κάποιου από τα καθορισμένα Application Programming Interfaces (APIs) που υλοποιούν κάποιο μοντέλο parsing. Έχουν αναπτυχθεί δύο APIs για το parsing: το Document Object Module (DOM) και το Simple API for XML (SAX).

Στο DOM ο parser διαβάζει όλο το έγγραφο και παράγει μια δομή δεδομένων (parse tree) που περιγράφει τα περιεχόμενα του εγγράφου (elements, attributes κλπ).

Στο SAX ο parser ενημερώνει την εφαρμογή για τα στοιχεία που συναντά καθώς διαβάζει σειριακά ένα έγγραφο. Αυτό το είδος parsing είναι γνωστό και ως βασισμένο στα συμβάντα (event-based).

Τα APIs αυτά έχουν υλοποιηθεί και σε Java. Η κοινότητα της Java έχει αναπτύξει και την προδιαγραφή JDOM η οποία βασίζεται στη φιλοσοφία του DOM και έχει το πλεονέκτημα ότι χειρίζεται τα parse trees με αντικείμενα περισσότερο συμβατά με τις δομές δεδομένων της Java. Με τη χρήση APIs και άλλων προδιαγραφών που έχουν υλοποιηθεί σε γλώσσες προγραμματισμού όπως η Java, οι εφαρμογές είναι σε θέση εκτός από το parsing, να δημιουργούν και να επεξεργάζονται έγγραφα XML για τις ανάγκες τους.

1.3.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΤΗΣ XML

- Χρησιμοποιεί μια γλώσσα που εύκολα μπορεί κάποιος να κατανοήσει (Χρήση ετικετών αναγνώσιμων & αναγνωρίσιμων από Η/Υ & ανθρώπους)
- Κοινό πρότυπο μεταξύ διαφορετικών πλατφορμών
- Αποθήκευση σε ASCII κείμενο
- Περιγραφή περιεχομένων σε ιεραρχική δομή και ευελιξία στη δομή καθώς ο καθένας δημιουργεί όσες και όποιες ετικέτες θέλει
- Επιτρέπει στο χρήστη, εάν είναι αναγκαίο, να δει τα δεδομένα χωρίς το πρόγραμμα που τα παρήγαγε.
- Είναι βάση για νέες γλώσσες & τεχνολογίες
- Οι περισσότερες εφαρμογές υποστηρίζουν την εξαγωγή και εισαγωγή στοιχείων από έγγραφα XML

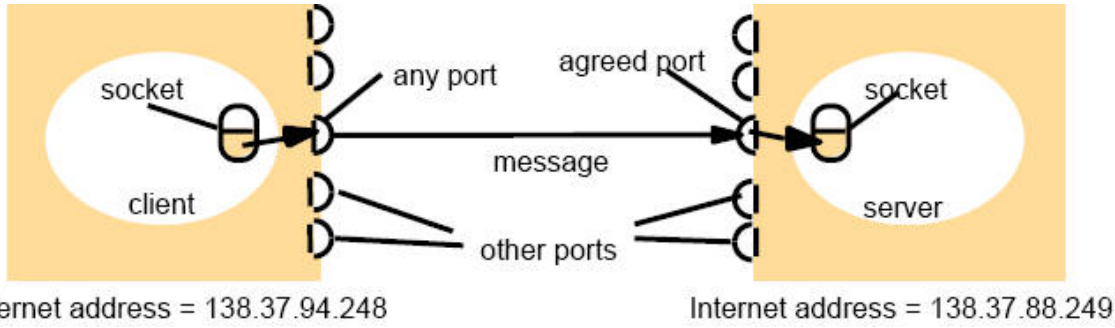
2. ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ

Για την επικοινωνία και τη μεταφορά δεδομένων μεταξύ δύο ή περισσοτέρων υπολογιστών ενός δικτύου χρησιμοποιούνται **πρωτόκολλα** που επιτρέπουν τη μεταφορά δεδομένων με εύκολο και ασφαλή τρόπο. Το πρωτόκολλο αναφέρεται στο σύνολο των κανόνων που διέπουν την επικοινωνία. Η τοπική-διεύθυνση και η απομακρυσμένη-διεύθυνση, προσδιορίζουν την ταυτότητα των υποδικτύων και των υπολογιστών, στους οποίους εκτελούνται οι επικοινωνούσες διεργασίες. Η τοπική-διεργασία και η απομακρυσμένη-διεργασία, προσδιορίζουν την ταυτότητα των διεργασιών που θα επικοινωνούν, καθώς σε έναν υπολογιστή, μπορούν να εκτελούνται περισσότερες της μιας διεργασίες. Κάθε μία από αυτές τις διεργασίες που εκτελούνται στον ίδιο host και που χρειάζονται επικοινωνία μέσω δικτύου, λαμβάνει έναν 16-bit ακέραιο αριθμό, ο οποίος αναπαριστά την θύρα (port number) της διεργασίας και κατ' επέκταση, της υπηρεσίας.
Ένα πρωτόκολλο επίσης ορίζει τη μορφή και τη σειρά των μηνυμάτων που ανταλλάσσονται μεταξύ δύο ή περισσοτέρων οντοτήτων που επικοινωνούν καθώς και τις ενέργειες που εκτελούνται κατά τη μετάδοση ή τη λήψη ενός μηνύματος ή ενός άλλου γεγονότος.

Η μετάδοση της πληροφορίας από υπολογιστή σε υπολογιστή γίνεται μέσω υποδοχών (sockets). Τα "**sockets**" είναι το όνομα που δίνεται, σε ένα συγκεκριμένο προγραμματιστικό μοντέλο, στα τελικά άκρα των συνδέσεων επικοινωνίας ανάμεσα στις διεργασίες. Εξ αιτίας της δημοτικότητας του συγκεκριμένου προγραμματιστικού μοντέλου το όνομα επαναχρησιμοποιήθηκε και σε άλλες περιοχές, συμπεριλαμβανομένης και της Java. Όταν οι διεργασίες επικοινωνούν πάνω από ένα δίκτυο, η Java χρησιμοποιεί και πάλι το μοντέλο ρευμάτων της (stream model). Στην περίπτωση αυτή, ένα socket κατέχει δύο ρεύματα, ένα ρεύμα εισόδου και ένα ρεύμα εξόδου. Μία διεργασία στέλνει τα δεδομένα της σε μία άλλη διεργασία μέσω του δικτύου γράφοντας απλά στο ρεύμα εξόδου που σχετίζεται με το socket. Η διεργασία διαβάζει δεδομένα που γράφονται από τη διεργασία στο «άλλο άκρο» της σύνδεσης διαβάζοντας απλά από το ρεύμα εισόδου που σχετίζεται με το socket. Από τη στιγμή που έχει δημιουργηθεί η δικτυακή σύνδεση, η χρήση των ρευμάτων που σχετίζεται με αυτή τη σύνδεση δεν είναι ιδιαίτερα διαφορετική από τη χρήση οποιουδήποτε άλλου ρεύματος.

Πιο συγκεκριμένα, για την επίτευξη της επικοινωνίας δυο διεργασιών μέσω μηνυμάτων αυτές θα πρέπει να υποστηρίζουν τις λειτουργίες επικοινωνίας μηνυμάτων send και receive. Μια διεργασία στέλνει (**send**) μήνυμα σε ένα προορισμό και μια διεργασία στον προορισμό λαμβάνει (**receive**) το μήνυμα. Σε κάθε προορισμό υπάρχει μια ουρά στην οποία αυτός που στέλνει μηνύματα, προς τον αντίστοιχο προορισμό, τα εισάγει και μια διεργασία στον προορισμό λαμβάνει τα μηνύματα από την ουρά. Τα μηνύματα στέλνονται στον προορισμό τους, ο οποίος προσδιορίζεται από ένα ζεύγος της μορφής (internet address, local port). Οι αριθμοί θυρών (local ports) στα συστήματα TCP/IP είναι αριθμοί των 16 bits, συνεπώς είναι στο εύρος 0 – 65535. Στην πράξη, οι αριθμοί θυρών κάτω από το 1024 είναι δεσμευμένοι για προκαθορισμένες υπηρεσίες, και δε θα πρέπει να τους χρησιμοποιείτε παρά μόνο αν θέλετε να επικοινωνήσετε με κάποια από αυτές τις υπηρεσίες (όπως το telnet, το SMTP mail, το ftp κ.ο.κ.). Πρέπει να συμφωνήσετε

σχετικά, εκ των προτέρων, ανάμεσα στα δύο προγράμματα, τον εξυπηρετούμενο (client) που ξεκινάει τη σύνδεση και τον εξυπηρέτη (server) που «αναμένει το τηλεφώνημα». Αν οι αριθμοί των θυρών που χρησιμοποιούνται από τα δύο μέρη του συστήματος δε συμφωνούν δε θα συμβεί ποτέ η επικοινωνία.



2.1 Τα πρωτόκολλα επικοινωνίας UDP και TCP

Τα πρωτόκολλα επικοινωνίας UDP και TCP κάνουν χρήση socket abstraction η οποία παρέχει ένα τελικό σημείο για την επικοινωνία μεταξύ δύο διεργασιών. Η διαδιεργασιακή επικοινωνία επιτυγχάνεται με την μετάδοση ενός μηνύματος ανάμεσα σε ένα socket της μιας διεργασίας και ένα socket της άλλης διεργασίας. Για να λάβει μια διεργασία ένα μήνυμα το socket της ΠΡΕΠΕΙ να είναι δεσμευμένο σε ένα local port και μια από τις Internet addresses (IA) του υπολογιστή στον οποίο εκτελείται Τα μηνύματα που αποστέλλονται σε συγκεκριμένη IA και συγκεκριμένο Port μπορούν να ληφθούν από τη διεργασία που το socket της είναι δεσμευμένο σε αυτή την IA και post number Κάθε υπολογιστής έχει πολλά post numbers (216) Μια διεργασία μπορεί να λαμβάνει από ΠΟΛΛΑ ports, αλλά ΔΕΝ μπορεί να μοιράζεται ports με άλλες διεργασίες, στον ίδιο υπολογιστή. Εξαίρεση αποτελεί η περίπτωση όπου οι διεργασίες χρησιμοποιούν IP multicast Οποιοσδήποτε αριθμός διεργασιών μπορούν να στέλνουν μηνύματα σε ένα port (κάποιου υπολογιστή) Κάθε socket σχετίζεται με ένα συγκεκριμένο πρωτόκολλο, είτε το UDP ή TCP.

Τα βασικά πρωτόκολλα που χρησιμοποιούνται στα δίκτυα υπολογιστών είναι TCP και το UDP.

2.1.2 TCP

Οι κύριοι στόχοι του πρωτοκόλλου TCP είναι να επιβεβαιώνεται η αξιόπιστη αποστολή και λήψη δεδομένων, επίσης να μεταφέρονται τα δεδομένα χωρίς λάθη και με τη σωστή σειρά στον αποδέκτη. Φροντίζει για χαμένα πακέτα, λήψη πακέτων με λανθασμένη σειρά, αν έχουν έρθει περισσότερες από μιά κόπιες ενός πακέτου, κα.

Το πρωτόκολλο ελέγχου μεταφορών (TCP) είναι connection oriented, δηλαδή η μεταφορά δεδομένων γίνεται μέσω σύνδεσης, η οποία οριοθετείται από ένα σήμα έναρξης και ένα σήμα τέλους ή διακοπής.

TCP είναι το ένα από τα δυό διαθέσιμα πρωτόκολλα μεταφοράς (Transport Protocol) στο 'TCP/IP'. Η αποστολή του είναι να παρέχει μιάν αξιόπιστη υπηρεσία σειριακής μεταφοράς δεδομένων - αυτό σημαίνει ότι αν στείλετε τα πακέτα 'a', 'b' και 'c', ο αποδέκτης τους θα τα παραλάβει σαν "abc", όχι "acb", "ac", "ccbac", κτλ. Φροντίζει για χαμένα πακέτα, λήψη πακέτων με λανθασμένη σειρά, αν έχουν έρθει περισσότερες από μιά κόπιες ενός πακέτου, κα.

2.1.3 UDP

Μία εναλλακτική ονομασία του πρωτοκόλλου είναι **Universal Datagram Protocol**. Διάφορα προγράμματα χρησιμοποιούν το πρωτόκολλο UDP για την αποστολή σύντομων μηνυμάτων (γνωστών και ως datagrams) από τον έναν υπολογιστή στον άλλον μέσα σε ένα δίκτυο υπολογιστών.

Ένα από τα κύρια χαρακτηριστικά του UDP είναι ότι δεν εγγυάται αξιόπιστη επικοινωνία. Δεν εγκαθιδρύει σύνδεση ανάμεσα στον αποστολέα και τον παραλήπτη πριν μεταδώσει τις πληροφορίες (σε κομμάτια που λέγονται αυτόνομα πακέτα - datagrams), και δεν περιμένει ούτε και απαιτεί επιβεβαίωση από τον παραλήπτη. Επιπλέον, επειδή τα αυτόνομα πακέτα που μεταδίδονται μέσω UDP είναι ανεξάρτητα μεταξύ τους, ακόμη και όταν αποτελούν τμήματα της ίδιας μετάδοσης, μπορεί να μη φτάσουν με τη σωστή σειρά, διπλά ή να μην φτάσουν καθόλου εάν το δίκτυο έχει μεγάλο φόρτο. Τα αυτόνομα πακέτα, όπως και τα μέσα μετάδοσης χωρίς επιβεβαίωση, είναι ασυνδεσμικά Η έλλειψη των μηχανισμών αυτών από το πρωτόκολλο UDP το καθιστά αρκετά πιο γρήγορο και αποτελεσματικό, τουλάχιστον για τις εφαρμογές εκείνες που δεν απαιτούν αξιόπιστη επικοινωνία.

Οι εφαρμογές audio και video streaming χρησιμοποιούν κατά κόρον πακέτα UDP. Για τις εφαρμογές αυτές είναι πολύ σημαντικό τα πακέτα να παραδοθούν στον παραλήπτη σε σύντομο χρονικό διάστημα ούτως ώστε να μην υπάρχει διακοπή στην ροή του ήχου ή της εικόνας. Κατά συνέπεια προτιμάται το πρωτόκολλο UDP διότι είναι αρκετά γρήγορο, παρόλο που υπάρχει η πιθανότητα μερικά πακέτα UDP να χαθούν. Στην περίπτωση που χαθεί κάποιο πακέτο, οι εφαρμογές αυτές διαθέτουν ειδικούς μηχανισμούς διόρθωσης και παρεμβολής ούτως ώστε ο τελικός χρήστης να μην παρατηρεί καμία αλλοίωση ή διακοπή στην ροή του ήχου και της εικόνας λόγω του χαμένου πακέτου. Σε αντίθεση με το πρωτόκολλο TCP, το UDP υποστηρίζει broadcasting, δηλαδή την αποστολή ενός πακέτου σε όλους τους υπολογιστές ενός δικτύου, και multicasting, δηλαδή την αποστολή ενός πακέτου σε κάποιους συγκεκριμένους υπολογιστές ενός δικτύου. Η τελευταία δυνατότητα χρησιμοποιείται πολύ συχνά στις εφαρμογές audio και video streaming ούτως ώστε μία ροή ήχου ή εικόνας να μεταδίδεται ταυτόχρονα σε πολλούς συνδρομητές.

Μερικές σημαντικές εφαρμογές που χρησιμοποιούν πακέτα UDP είναι οι εξής: Domain Name System (DNS), IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) και τα παιχνίδια που παίζονται ζωντανά μέσω του Διαδικτύου.

2.2. Το μοντέλο δικτύωσης της Java

Στη Java οι socket συνδέσεις TCP/IP υλοποιούνται με κλάσεις από το πακέτο java.net. Πιο κάτω υπάρχει ένα διάγραμμα του τι συμβαίνει στην πλευρά του εξυπηρέτη και τι

στην πλευρά του εξυπηρετούμενου. Ο εξυπηρέτης αναθέτει έναν αριθμό θύρας. Όταν ο εξυπηρετούμενος αιτείται μία σύνδεση, ο εξυπηρέτης ανοίγει τη σύνδεση socket με τη μέθοδο accept().

- Ο εξυπηρετούμενος εγκαθιδρύει μία σύνδεση με το host στη θύρα port #.
- Τόσο ο εξυπηρετούμενος όσο και ο εξυπηρέτης επικοινωνούν χρησιμοποιώντας ένα InputStream και ένα OutputStream. Ο κώδικας για να υλοποίησης αυτού του είδους της επικοινωνία παρουσιάζεται στις επόμενες σελίδες.

2.2.1 Ελάχιστος εξυπηρετητής TCP/IP

Οι εφαρμογές εξυπηρέτη TCP/IP βασίζονται στις κλάσεις δικτύωσης ServerSocket και Socket που παρέχονται από τη γλώσσα Java. Η κλάση ServerSocket αναλαμβάνει το μεγαλύτερο μέρος της δουλειάς για την εγκαθίδρυση ενός εξυπηρέτη.

```
import java.net.*;
import java.io.*;
public class SimpleServer
{
        public static void main(String args[]) {
        ServerSocket s = null;
        Socket s1;
        String sendingString = "Hello Net World!";
        Int slength = sendString.length();
        OutputStream = s1out;
        DataOutputStream = dos;

        // Εγγραφή της υπηρεσίας στη θύρα 5432
        try
        {
                s = new ServerSocket(5432);
        } catch (IOException e) { }

        // εκτελείται για πάντα ο βρόχος listen/accept
        while (true)
        {
                try
                {
                        // Αναμονή και «ακούει» για σύνδεση
                        s1 = s.accept();

                        // Παίρνει το ρεύμα επικοινωνίας που σχετίζεται με το socket
                        s1out = s1.getOutputStream();
                        dos = new DataOutputStream(s1out)

                        // Στείλτε τη συμβολοσειρά σας
                        dos.writeUTF(sendString);
                        dos.close();

                        // Κλείσε τη σύνδεση, αλλά όχι και το socket του εξυπηρέτη
                        s1out.close();
```

```
                    s1.close();
                } catch (IOException e) { }
        }
}}
```

2.2.2 Ελάχιστος εξυπηρετούμενος TCP/IP

Η πλευρά του εξυπηρετούμενου σε μία εφαρμογή TCP/IP βασίζεται στην κλάση Socket. Και πάλι, μεγάλο μέρος της δουλειάς που έχει να κάνει με την εγκαθίδρυση της σύνδεσης το αναλαμβάνει η κλάση Socket. Αυτός ο εξυπηρετούμενος συνδέεται στον εξυπηρέτη που μόλις περιγράψαμε και παρουσιάζει ό,τι του στέλνει ο εξυπηρέτης στο stdout.

```
import java.net.*;
import java.io.*;
public class SimpleClient
{
        public static void main(String args[]) throws IOException
        {
                int c;
                Socket s1;
                InputStream s1In;
                DataInputStream dis;

                // Ανοίγει τη σύνδεση στον υπολογιστή sunbert, θύρα 5432
                s1 = new Socket("sunbert", 5432);

                // Λαμβάνει ένα χειριστήριο αρχείου εισόδου (input file handle) από το socket και
                // διαβάζει την είσοδο
                s1In = s1.getInputStream();
                dis = new DataInputStream(s1In);
                String st = new String(dis.readUTF());
                System.out.println(st);

                // Όταν τελειώσει απλά κλείνει τη σύνδεση και φεύγει
                dis.close();
                s1In.close();
                s1.close();
        }
}
```

2.2.3 UDP Sockets

Το User Datagram Protocol υποστηρίζεται μέσω δύο κλάσεων της Java, DatagramSocket και DatagramPacket. Το πακέτο (packet) είναι ένα πλήρες μήνυμα που περιέχει πληροφορία σχετικά με τον αποστολέα, το μήκος του μηνύματος και το ίδιο το μήνυμα.

2.2.3.1 DatagramPacket

Το DatagramPacket έχει δύο συναρτήσεις δημιουργίας – μία για να λαμβάνει δεδομένα

και μία για να στέλνει:
· DatagramPacket(byte [] recvBuf, int readLength) – χρησιμοποιείται για να φτιάξει ένα πίνακα από bytes ώστε να λαμβάνει ένα πακέτο UDP. Ο πίνακας των bytes είναι κενός όταν μεταβιβάζεται στη συνάρτηση δημιουργίας και η μεταβλητή int τίθεται στο πλήθος των bytes που πρέπει να διαβαστούν (και δεν είναι μεγαλύτερη από το μήκος του πίνακα).

· DatagramPacket(byte [] sendBuf, int sendLength, InetAddress iaddr, int iport) – χρησιμοποιείται για να προετοιμάσει ένα πακέτο UDP για μετάδοση. Το sendLength δεν είναι μεγαλύτερο από το μήκος του πίνακα bytes sendBuf.

2.2.3.2 DatagramSocket
Το DatagramSocket χρησιμοποιείται για να διαβάζουμε και να γράφουμε πακέτα UDP. Η κλάση έχει τρεις συναρτήσεις δημιουργίας που σας επιτρέπουν να καθορίζετε σε ποια θύρα και διεύθυνση διαδικτύου (internet address) για σύνδεση:

· DatagramSocket() – σύνδεση σε οποιαδήποτε διαθέσιμα θύρα στον τοπικό υπολογιστή.
· DatagramSocket(int port) – σύνδεση στη συγκεκριμένη θύρα στον τοπικό υπολογιστή.
· DatagramSocket(int port, InetAddress iaddr) – σύνδεση στη συγκεκριμένη θύρα στη συγκεκριμένη διεύθυνση.

3. X3D

Τεχνολογίες και Πρωτόκολλα για Εικονικά Περιβάλλοντα

3.1 X3D - VRML
Εισαγωγή
Το πρώτο βήμα για τη σύνδεση του παγκόσμιου ιστού με την εικονική πραγματικότητα έγινε στα μέσα περίπου της προηγούμενης δεκαετίας. Η χρήση του διαδικτύου εξαπλωνόταν με ταχείς ρυθμούς και έτσι γεννήθηκε η ιδέα της σύνδεσης αυτού του επικοινωνιακού μοντέλου με τις τρεις διαστάσεις. Το έτος 1994 γράφτηκε το πρόγραμμα Labyrinth, από τους Mark Pesce και Tony Parisi, το οποίο παρουσίαζε στο χρήστη εικονικούς τρισδιάστατους κόσμους, αφού τους μετέφερε μέσω δικτύου χρησιμοποιώντας τα συνηθισμένα πρωτόκολλα των κοινών ιστοσελίδων. Το πρόγραμμα παρουσιάστηκε, τότε, στο πρώτο παγκόσμιο συνέδριο του W3C. Η γλώσσα περιγραφής των τρισδιάστατων γραφικών ονομάστηκε *VRML* , από τα αρχικά των λέξεων ***V****irtual* ***R****eality* ***M****arkup* ***L****anguage*. Αργότερα η λέξη *Markup* αντικαταστάθηκε από τη λέξη *Modeling*.

Σκοπός της νέας γλώσσας ήταν ένα *format* που θα ήταν εύκολα "μεταφέρσιμο" στο διαδίκτυο. Αποφασίστηκε να ακολουθηθεί το παράδειγμα της γλώσσας *HTML* . Συγκεκριμένα χαρακτηριστικά, όπως η κωδικοποίηση με απλό κείμενο και η εύκολη αναγνωσιμότητα, υιοθετηθήκαν. Παρότι υπήρχε πληθώρα από *formats*, τόσο δυαδικών όσο και *text*, κανένα δε θεωρήθηκε ικανοποιητικό. Η ανάπτυξη μιας νέας κωδικοποίησης

κρίθηκε απαραίτητη. Αρχικά η *VRML* δανείστηκε αρκετά στοιχεία από την κωδικοποίηση που χρησιμοποιούσε η βιβλιοθήκη γραφικών OpenInventor της Silicon Graphics™. Το αποτέλεσμα ήταν το πρότυπο **VRML 1.0**.

Από τη γέννησή της η *VRML* υπόκεινται σε συνεχείς αλλαγές, αναθεωρήσεις, προσθήκες με αποτέλεσμα την απόκτηση νέων χαρακτηριστικών και δυνατοτήτων. Οι κόσμοι της *VRML* 1.0 ήταν στατικοί, χωρίς να δίνουν δυνατότητα αλληλεπίδρασης. Το μόνο που έκανε ο χρήστης ήταν η πλοήγηση και η μεταφορά σε άλλους κόσμους με χρήση υπερσυνδέσμων. Γρήγορα αναπτύχθηκε το πρότυπο **VRML 2.0** , το οποίο εφοδίαζε τη γλώσσα με δυναμικές συμπεριφορές και με την ικανότητα αλληλεπίδρασης με το χρήστη. Τα αντικείμενα ενός κόσμου μπορούσαν να κινούνται, να αλλάζουν χρώμα και άλλες ιδιότητες, να εξαφανίζονται ή και να δημιουργούνται δυναμικά. Μέθοδοι εισόδου από το χρήστη προστέθηκαν και ήταν πλέον δυνατό να ανιχνευθεί η θέση και η κίνηση του χρήστη. Το εικονικό περιβάλλον μπορούσε να προγραμματίζεται σε JavaScript και Java, χάρις τον κόμβο *Script*. Επίσης η γραμματική της γλώσσας εξελίχθηκε ακολουθώντας ένα καλύτερο προγραμματιστικό μοντέλο, το οποίο επέτρεπε πιο συμπαγή και κατανοητό κώδικα.

Η επόμενη έκδοση της VRML ήταν η **VRML97** , η οποία είναι διεθνές πρότυπο (ISO/IEC 14772:1997) ([40], [41]). Η βασική διαφορά από την προηγούμενη έκδοση είναι η εισαγωγή ενός API, του EAI (External Authoring Interface), το οποίο επιτρέπει σε εξωτερικές εφαρμογές προγραμματιστική πρόσβαση στα στοιχεία του εικονικού κόσμου. Το API περιγράφεται στα πλαίσια της γλώσσας IDL, οπότε η σύνδεση κόσμου με εφαρμογή είναι ανεξάρτητη γλώσσας. Όμως, οι πλήρως σύμφωνες με το πρότυπο υλοποιήσεις πρέπει να παρέχουν υλοποίηση του μηχανισμού σε γλώσσα Java. Ο ορισμός του API σε Java δίνεται από το ίδιο το πρότυπο, πράγμα που συνεπάγεται μικρότερη πιθανότητα ασυμβατοτήτων.
Η επόμενη έκδοση του προτύπου λέγεται Extensible 3D (X3D) και αποτελεί σύνολο FDIS προτύπων με τους κωδικούς "ISO/IEC FDIS 19775:200x" , "ISO/IEC FDIS 19776:200x" , "ISO/IEC FDIS 19777:200x"[22] ([32], [33], [34], [35], [36], [37], [38]). Το νέο πρότυπο αυξάνει τον αριθμό των *standard* κόμβων από 74 (συν 20 προαιρετικοί) σε 155 (συν 51 αφαιρετικοί κόμβοι ή ορισμοί διασυνδέσεων κόμβων). Εκτός αυτού ορίζεται XML κωδικοποίηση της γλώσσας και δυαδική κωδικοποίηση που βασίζεται στην XML. Ενισχύεται η προγραμματιστική πρόσβαση στους κόσμους με το API SAI (**S**cene **A**ccess **I**nterface). Γενικά το πρότυπο ορίζει αυστηρότερα τις προδιαγραφές, αφήνοντας μικρό περιθώριο αμφισβητήσεων.

3.2 X3D profiles

Πολύ σημαντικό είναι το ότι εισάγει την έννοια του profile, σύνολο, δηλαδή, υποστηριζόμενων κόμβων και λειτουργιών από μια υλοποίηση. Έτσι εφαρμογές που χρησιμοποιούν πιο περιοριστικά profile μπορούν να ικανοποιηθούν από απλούστερες υλοποιήσεις.

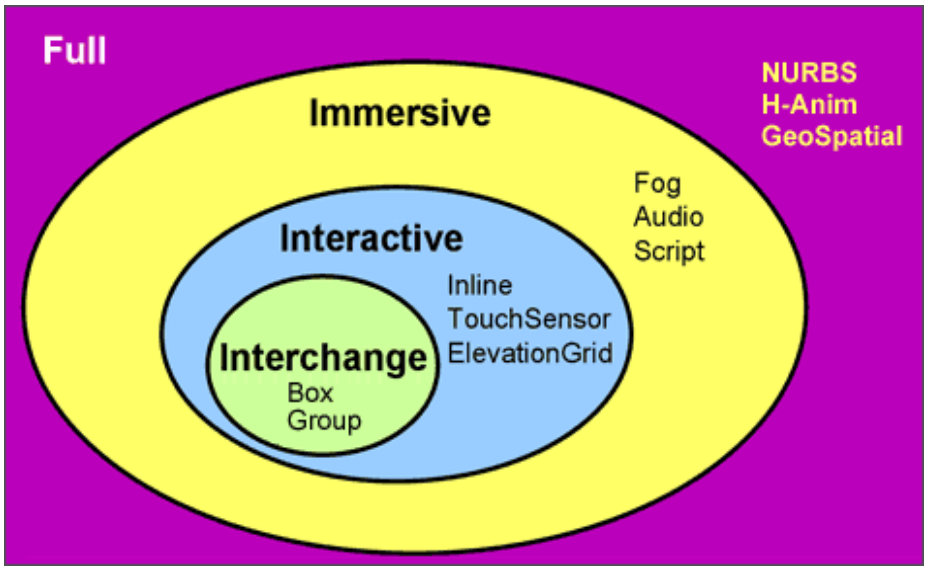
Τα κυριότερα profiles είναι:

Interchange: Είναι το βασικό profile για την επικοινωνία μεταξύ εφαρμογών. Υποστηρίζει γεωμετρία, texturing, φωτισμό, και animation

Interactive: ενεργοποιεί τη δυνατότητα βασική αλληλεπίδραση ενός 3D περιβάλλοντος προσθέτοντας κόμβους αισθητήρων (sensor nodes) για τη πλοήγηση και την αλληλεπίδραση των χρηστών με το 3d περιβάλλον (π.χ., PlanseSensor, TouchSensor, κ.λπ)

Immersive: ενεργοποιεί τη δυνατότητα χρήσης τρισδιάστατα γραφικών και αλληλεπίδρασης συμπεριλαμβανομένης της ακουστικής υποστήριξης και άλλων δυνατοτήτων

Full: Περιλαμβάνει το σύνολο όλων των καθορισμένων κόμβων (nodes) συμπεριλαμβανομένου NURBS, χ- Anim και των συστατικών GeoSpatial.



Οι στόχοι των προτύπων X3D - VRML97 είναι η κάλυψη αναγκών διάφορων εφαρμογών όπως:

- Παρουσίαση τρισδιάστατων κόσμων στο διαδίκτυο (ή σε τοπικό δίκτυο).
- Διαμοιραζόμενοι εικονικοί κόσμοι.
- Απεικονίσεις δεδομένων στον επιστημονικό και τεχνικό τομέα.
- Πολυμεσικές παρουσιάσεις και γενικότερα πολυμεσικές εφαρμογές.
- Εκπαιδευτικές και ψυχαγωγικές εφαρμογές.

Φιλοδοξία του WEb3D Consortium, που έχει αναλάβει την ανάπτυξη των προτύπων X3D - VRML97, είναι το X3D να αποτελέσει ένα κοινά αποδεκτό format στο τομέα των τρισδιάστατων γραφικών και των πολυμέσων24. Έτσι ο σχεδιασμός των στοιχείων που αποτελούν το πρότυπο έγινε σεβόμενος ορισμένες, πολλές φορές αντίθετες μεταξύ τους, αρχές, όπως την επεκτασιμότητα, τη συνδεσιμότητα, τη δυνατότητα επαναχρησιμοποίησης κώδικα, την ευκολία συγγραφής κώδικα, την ικανότητα κλιμάκωσης, το διαχωρισμό των δεδομένων από την αρχιτεκτονική του συστήματος χρόνου εκτέλεσης, την απόδοση, την υποστήριξη ποικίλλων κωδικοποιήσεων και

εναλλακτικών προγραμματιστικών διασυνδέσεων, καθώς και την τμηματική οργάνωση των δυνατοτήτων της γλώσσας (*profile*).

Συνοψίζουμε τις βασικές δυνατότητες:

- Αναπαράσταση τρισδιάστατων μοντέλων με πολύγωνα, παραμετρική γεωμετρία, καθορισμένο φωτισμό και texture mapping.
- Απόδοση δυσδιάστατων γραφικών και κειμένου σε επίπεδα του τρισδιάστατου κόσμου.
- Δυνατότητα animation μέσω στοιχείων παρεμβολής (interpolators) και υπολογισμού χρόνου (timers).
- Ενσωμάτωση στοιχείων ήχου και κινούμενης εικόνας στο τρισδιάστατο χώρο.
- Αλληλεπίδραση με το χρήστη, τουλάχιστον, μέσω πληκτρολογίου και ποντικιού, με δυνατότητα επιλογής και μετακίνησης αντικειμένων.
- Ικανότητα πλοήγησης του χρήστη με βασική προσομοίωση βαρύτητας και ανίχνευση συγκρούσεων με αντικείμενα, εγγύτητας σε αυτά ή οπτικής επαφής μαζί τους.
- Καθορισμός νέων αντικειμένων με συγκεκριμένα χαρακτηριστικά και συμπεριφορές (μηχανισμός protos και externprotos).
- Δυναμική συμπεριφορά του κόσμου, η οποία καθορίζεται μέσω προγραμματισμού.
- Διαφάνεια δικτύου. Τα τμήματα ενός κόσμου βρίσκονται σε διάφορους δικτυακούς τόπους και οι κόσμοι συνδέονται με άλλους που βρίσκονται, επίσης, στο δίκτυο.
- Δυνατότητα φυσικών προσομοιώσεων.

3.2 Web3D Consortium

Το Web3D Consortium (http://www.web3d.org) αποτελεί ένα μη κερδοσκοπικό οργανισμό που έχει ως στόχο τη συγγραφή και προώθηση προτύπων και τεχνολογικών λύσεων όσον αφορά τα τρισδιάστατα γραφικά και τον παγκόσμιο ιστό. Στον οργανισμό συμμετέχουν άλλοι οργανισμοί και ηγετικές εταιρίες του χώρου, όπως οι Silicon Graphics, Sony, Apple, Microsoft και Oracle. Οι εργασίες του έχουν ανατεθεί σε διάφορα *working groups* (http://www.web3d.org/WorkingGroups/) , τα οποία παραθέτονται στον κάτωθι πίνακα:

Working Group	**Σχόλιο**
X3D Conformance Program	Καθορισμός διαδικασίας ελέγχου συμμόρφωσης με τα πρότυπα και πιστοποίησης προϊόντων διαθέσιμων στην αγορά
X3D Shaders	Ενσωμάτωση στο πρότυπο δυνατότητας προγραμματιζόμενης , υψηλού επιπέδου, διαδικαστικής σκίασης των τρισδιάστατων μοντέλων (programmable shading)
GeoSpatial	Εκτός από την εύρεση τρόπων αναπαράστασης γεωγραφικών δεδομένων με τρισδιάστατα μοντέλα, έχει ως αποστολή και την πρόσβαση, από το X3D, σε χωρικά δεδομένα μέσω υπηρεσιών ιστού (web services)
Dis-XML (Distributed Interactive Simulation)	Προδιαγραφές για πολυχρηστικούς κόσμους με πολλούς συμμετέχοντες
H-Anim (Humanoid Animation)	Τυποποίηση του τρόπου αναπαράστασης και κίνησης ανθρωπόμορφων avatar ή πρακτόρων

X3D Source & Tool Development	Ανάπτυξη προγραμματιστικών εργαλείων και προσφορά ανοιχτού πηγαίου κώδικα για την υποστήριξη του προτύπου
CAD	Δυνατότητα χρήσης CAD δεδομένων από ποικίλες εφαρμογές, χάρις μια ορισμένη διεπαφή που περιγράφει το πρότυπο X3D
Medical X3D	Περιγραφή της ανθρώπινης ανατομίας και καθορισμός μεθόδων που θα επιτρέπουν οπτικοποιήσεις πραγματικού χρόνου της κατάστασης της υγείας ασθενών
Visual Simulation	Ανάπτυξη του πρωτοκόλλου XMSF (Exensible Modeling and Simulation Framework) για την κατανεμημένη προσομοίωση μοντέλων και την ενσωμάτωση υπαρχόντων εργαλείων και βιβλιοθηκών, που εξειδικεύονται στον τομέα της προσομοίωσης

Πίνακας Ενεργά Working Groups του Web3D Consortium

Ορισμένες από της ομάδες εργασίας του πίνακα 1 συνεχίζουν το έργο παλαιότερων ομάδων εργασίας του WEb3D Consortium, οι οποίες σχετίζονταν με το πρότυπο *VRML* . Ορισμένες από αυτές φαίνονται στον πίνακα 1. Άλλες από αυτές ολοκλήρωσαν το έργο τους με την αποδοχή και ενσωμάτωση των συστάσεων τους από το X3D (κυρίως μέσω του ορισμού νέων κόμβων), ενώ άλλες τερματίστηκαν χωρίς να έχει γίνει αναγνώριση, τουλάχιστον επίσημα, του έργου τους (όπως *ooe-vrml*, *nlp-anim*, *living-worlds*).

Working Group	**Σχόλιο**
vrml-eai (External Authoring Interface)	API αλληλεπίδρασης εξωτερικής εφαρμογής και τρισδιάστατου μοντέλου
geovrml - GeoVRML	Τρόποι αναπαράστασης γεωγραφικών δεδομένων με τρισδιάστατα μοντέλα και επεξεργασία αυτών των δεδομένων
living-worlds (Living Worlds)	Προδιαγραφές για τη σύνταξη πολυχρηστικών κόσμων
vrml-mpeg4 (MPEG-4 Integration)	Χρήση του προτύπου από το MPEG-4 για παρουσίαση κόσμων με περιορισμένη αλληλεπίδραση και δυναμική συμπεριφορά
dis-java-vrml (Distributed Interactive Simulation)	Προδιαγραφές για πολυχηστικούς κόσμους με πολλούς συμμετέχοντες
nlp-anim (Natural Language Processing)	Επεξεργασία φυσικής γλώσσας για αλληλεπίδραση με στοιχεία του κόσμου
ooe-vrml (Object-Oriented Extensions)	Αντικειμενοστρεφείς επεκτάσεις της γλώσσας.
vrml-streams (VRML Streaming)	Τρόποι μεταφοράς και χειρισμού των αντικειμένων ενός κόσμου ως ροή δεδομένων
VRTP - Virtual Reality Transport Protocol	Ανάπτυξη εξειδικευμένου δικτυακού πρωτοκόλλου για την υποστήριξη δικτυακών, πολυχρηστικών και μεγάλης κλίμακας κόσμων
vrml-java3d	Μέθοδοι συνδυασμού της VRML με το Java 3D API

Πίνακας Παλαιότερα Working Groups του Web3D Consortium

3.3 X3D Browsers

Η ερμηνεία, η εκτέλεση, και η παρουσίαση των X3D αρχείων γίνεται χρησιμοποιώντας έναν μηχανισμό γνωστό σαν *browser*, ο οποίος επιδεικνύει τις μορφές και τους ήχους στη γραφική παράσταση σκηνής. Αυτή η παρουσίαση είναι γνωστή ως εικονικός κόσμος και πλοηγείται στον browser από μια ανθρώπινη ή μηχανική οντότητα, γνωστή ως χρήστης. Ο κόσμος επιδεικνύεται από μια ιδιαίτερη θέση, αυτή η θέση και ο προσανατολισμός στον κόσμο, είναι γνωστά σαν *viewer.* Ο *browser* είναι δυνατόν να παρέχει παραδείγματα πλοήγησης (όπως walking ή flying)) που επιτρέπουν στο χρήστη για να κινήσουν τη θέαση (*viewer*) στον εικονικό κόσμο.

Εκτός από τη πλοήγηση, ο *browser* παρέχει έναν περιορισμένο μηχανισμό που επιτρέπει στο χρήστη για να αλληλεπιδράσει με τον κόσμο μέσω των κόμβων αισθητήρων στην ιεραρχία γραφικών παραστάσεων σκηνής. Οι αισθητήρες αποκρίνονται στην αλληλεπίδραση χρηστών με τα γεωμετρικά αντικείμενα στον κόσμο, το πώς κινείται ο χρήστης μέσα στον κόσμο, ή τη μετάβαση του χρόνου. Η X3D διεπαφή πρόσβασης σκηνής (Scene Access Interface, SAI) παρέχει τους μηχανισμούς για την ανάγνωση των δεδομένων που εισάγουν οι χρήστες, και για να την τρέχουσα όψη του κόσμου.

Προκειμένου να παρέχει τις ικανότητες πλοήγησης, ένας viewer μπορεί να χρησιμοποιήσει το SAI για να παρέχει στο χρήστη τη δυνατότητα να πλοηγήσει. Επιπλέον οι δημιουργοί των σκηνών είναι δυνατόν να χρησιμοποιήσουν προγραμματιστικές γλώσσες με διασυνδέσεις στο SAI για να εφαρμόσουν δικούς τους αλγόριθμους πλοήγησης.

Η οπτική παρουσίαση γεωμετρικών αντικειμένων σε έναν X3D κόσμο, ακολουθεί ένα εννοιολογικό πρότυπο, σχεδιασμένο να μοιάσει με τα φυσικά χαρακτηριστικά του φωτός. Το X3D πρότυπο φωτισμού περιγράφει πώς οι ιδιότητες και τα φώτα εμφάνισης στον κόσμο συνδυάζονται για να παραγάγουν τα επιδειχθέντα χρώματα.

3.4 X3D Components

Το X3D περιλαμβάνει ένα σύνολο από προδιαγραφές δομικών μονάδων (components) που επιτρέπουν την ανάπτυξη εφαρμογών που μπορούν να εκτελεστούν σε ένα πλήθος πλατφορμών από σταθμούς εργασίας. Ανάμεσα στα αρχικά components περιλαμβάνεται μια «μηχανή» παραγωγής τρισδιάστατων γραφικών με εξαιρετικές δυνατότητες, έναν ανεξάρτητο από πλατφόρμα τύπο αρχείου και εκτεταμένη ολοκλήρωση με XML. Τα καθορισμένα components δεν είναι τίποτε άλλο από συσχετισμένες συλλογές των διάφορων x3d κόμβων.

3.4.1 Εσωτερικές Δομές

Το X3D σύστημα αποτελείται από τις αφηρημένες μεμονωμένες οντότητες, τα αντικείμενα. Στη διαδικασία χειρισμού ενός εικονικού μοντέλου, το οποίο περιγράφεται σε γλώσσα VRML ή X3D, από σχετικά προγράμματα (προβολής, προσομοίωσης, συγγραφής κτλ), διακρίνονται ορισμένοι βασικοί μηχανισμοί. Διακριτά κομμάτια αυτών των μηχανισμών είναι η ιεραρχία του τρισδιάστατου σκηνικού μοντέλου (scene graph transformation hierarchy) και ο γράφος αλληλεπίδρασης μεταξύ των αντικειμένων του κόσμου (behaviour graph ή route graph). Η πρώτη δομή, από αυτές, περιγράφει τις

οντότητες που απαρτίζουν τον εικονικό κόσμο, ενώ η δεύτερη καθορίζει τον τρόπο εκδήλωσης δυναμικών συμπεριφορών, κατά την παρουσίαση του κόσμου στο χρήστη.

Ο γράφος που συνδέει τα αντικείμενα της σκηνής αποτελείται από κόμβους (nodes), οι οποίοι συνδέονται μεταξύ τους δεντρικά, με σχέσεις γονέα – παιδιού. Οι κόμβοι περιγράφουν, μεταξύ άλλων, γεωμετρικά σχήματα και εμφανισιακά χαρακτηριστικά αυτών (όπως χρώμα, φωτεινότητα και διαφάνεια). Οι συντεταγμένες ενός γεωμετρικού αντικειμένου είναι σχετικές ως προς τις συντεταγμένες του κόμβου γονέα του. Άρα η transformation hierarchy είναι η βασική δομή που καθορίζει τι ακριβώς θα εμφανιστεί στην οθόνη του χρήστη.

Εκτός, όμως, από μια στατική αναπαράσταση τρισδιάστατων γραφικών, χρειάζεται και ένας μηχανισμός δυναμικού χειρισμού της κατάστασης του εικονικού κόσμου. Διαφορετικά τίποτα δε θα άλλαζε, με αποτέλεσμα λιγότερη ρεαλιστικότητα και μειωμένη χρηστικότητα. Η λύση δίνεται με τη χρήση των routes, ιδεατών αγωγών από κάποιο πεδίο ενός κόμβου σε κάποιο πεδίο ενός άλλου κόμβου. Έτσι όταν αλλάζει η τιμή του πεδίου πηγή, η νέα αυτή τιμή διαδίδεται και στο πεδίου του κόμβου στόχου. Αρχικά οι αλλαγές τιμών πυροδοτούνται από ενέργειες του χρήστη (που αντιλαμβάνονται ειδικοί κόμβοι αισθητήρες), από το πέρασμα του χρόνου (κόμβοι με συμπεριφορά βασιζόμενη στο χρόνο), εξωτερικά ερεθίσματα κα. Το σύνολο των routes αποτελεί τον προαναφερθέν route graph.

Πρέπει να σημειωθεί ότι ο route graph και η transformation hierarchy μπορεί να είναι δυο διακριτές δομές, αλλά πιθανόν και μια ενοποιημένη. Εξάλλου η λειτουργία των routes, συνήθως έχει άμεσο αποτέλεσμα την αλλαγή τιμής σε κόμβους του κόσμου. Η απόφαση, για το σχεδιασμό των δομών δεδομένων, εξαρτάται από την υλοποίηση και τις τεχνολογίες που χρησιμοποιούνται. Ίσως, για λόγους απόδοσης, να θεωρείται καλύτερο, ο γράφος με τα αντικείμενα που σχετίζονται άμεσα με τα γραφικά να αποτελεί ξεχωριστή δομή, έτσι ώστε η επεξεργασία του να ευκολότερη και γρηγορότερη.

3.4.1.1 Κόμβοι και Πεδία

Όπως αναφέρθηκε, η περιγραφή ενός εικονικού κόσμου γίνεται με τη σύνταξη ενός δεντρικού σχηματισμού, του οποίου βασικά μέλη είναι αντικείμενα που ονομάζουμε κόμβους (nodes). Οι κόμβοι έχουν ιδιότητες, οι τιμές των οποίων μπορεί να τίθενται από το χρήστη. Τέτοιες, δυναμικές ιδιότητες, λέγονται πεδία (fields) και ίσως να έχουν ως τιμή άλλους κόμβους (έτσι, άλλωστε, δημιουργείται η δεντρική δομή).

Ένα πεδίο μπορεί να περιέχει μια συγκεκριμένη μεταβλητή δεδομένου τύπου ή ένα πίνακα μεταβλητών δεδομένου τύπου. Αυτό αποτελεί και ένα κριτήριο διαχωρισμού για τον τύπο των δεδομένων που έχει σαν τιμή ένα πεδίο. Αν η τιμή του πεδίου είναι μια απλή μεταβλητη τότε στον τύπο δεδομένων της μεταβλητής τοποθετείται το πρόθεμα SF (*πχ. το πεδίο α είναι του τύπου SFVec3f*). Στην περίπτωση που η τιμή του πεδίου είναι ένας πίνακας, τοποθετείται το πρόθεμα MF στην ονομασία των τύπων δεδομένων του πίνακα (*πχ. το πεδίο β είναι του τύπου MFVec3f*)

Κάθε αντικείμενο έχει τα ακόλουθα χαρακτηριστικά:

- **Όνομα τύπου δεδομένων:** Παράδειγμα SFVec3f, MFColor, SFFloat, Group, Background, or SpotLight.
- **Implementation :** Η εφαρμογή κάθε αντικειμένου, καθορίζει το πώς θα αντιδράσει στις μεταβολές των τιμών του.

Κάθε x3dNode έχει τα ακόλουθα χαρακτηριστικά:

➢ **Τιμές στις μεταβλητές των πεδίων:** Οι τιμές των πεδίων κάθε κόμβου αποθηκεύονται στο x3d αρχείο μαζί με τους κόμβους ή με τα πεδία, και καθορίζουν την κατάσταση του εικονικού κόσμου.

➢ **Συμβάντα που μπορούν να στείλουν και να λάβουν συμβάντα:** Κάθε κόμβος μπορεί να λαμβάνει συμβάντα στα πεδία του που θα προκαλέσουν αλλαγή στην κατάσταση του κόμβου. Συμβάντα που δημιουργούνται από ένα κόμβο, μπορούν να συνδεθούν με πεδία άλλων κόμβων για να διαδώσουν τις αλλαγές. Αυτό γίνεται με τη χρήση της δήλωσης ROUTE.

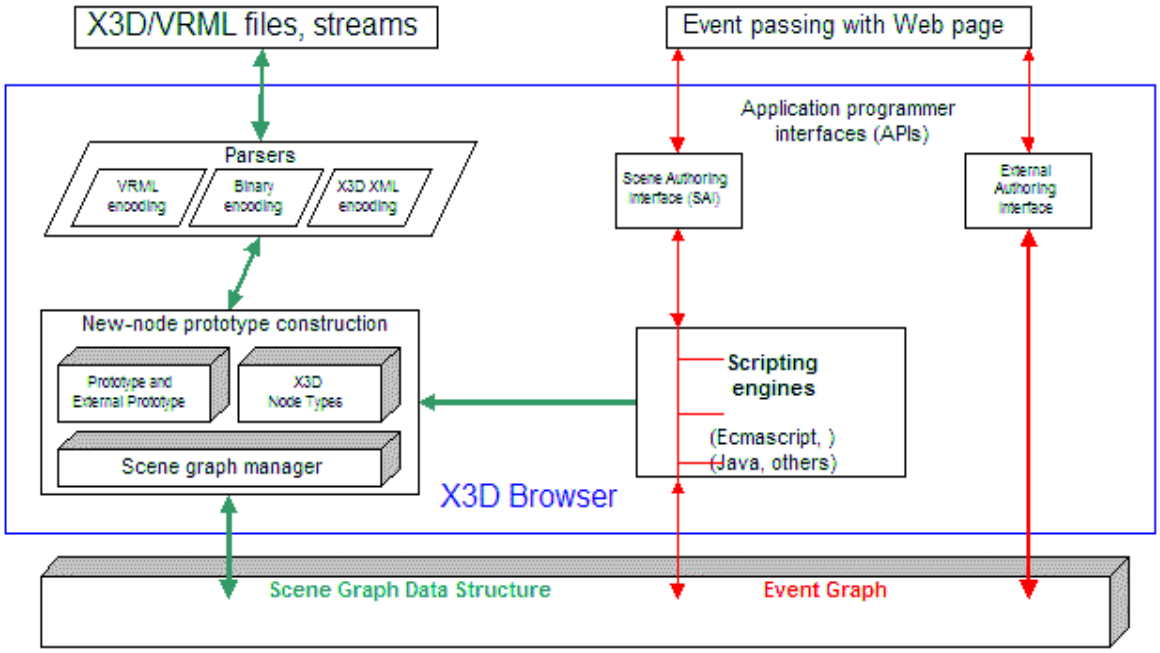
➢ **Όνομα:** Οι κόμβοι μπορούν να ονοματίζονται από το χρήστη χρησιμοποιώντας τη δήλωση def. Το όνομα χρησιμεύει στον εντοπισμό ενός κόμβου από άλλους.

3.4.1.2 Πεδία

Τα πεδία ενός κόμβου είναι αυτά που καθορίζουν την παρούσα κατάσταση του κόμβου και τις τιμές που οι κόμβοι μπορούν να στείλουν ή να λάβουν. Το X3D υποστηρίζει τέσσερις τύπους πρόσβασης στο πεδίο ενός κόμβου:

- **initializeOnly πρόσβαση**: Επιτρέπει την αρχικοποίηση της τιμής ενός πεδίου, αλλά δεν επιτρέπει την αλλαγή της.
- **inputOnly πρόσβαση**: Ο κόμβος μπορεί να λάβει ένα γεγονός για να αλλάξει την της τιμή ενός πεδίου, αλλά η τιμή δεν επιτρέπει το διάβασμα της τιμής
- **outputOnly πρόσβαση**: Ο κόμβος μπορεί να στείλει ένα γεγονός όταν αλλάξει η τιμή ενός δεδομένου πεδίου αλλά δεν μπορεί να αποθηκευτεί η νέα τιμή στο πεδίο.
- **inputOutput πρόσβαση** : Επιτρέπει την πλήρη πρόσβαση στο πεδίο:μπορεί να οριστεί μια αρχική αξία στο πεδίο ενός κόμβου, ο κόμβος μπορεί να λάβει ένα γεγονός που να αλλάξει την τιμή στο πεδίο του , και να στείλει ένα γεγονός όταν η τιμή του πεδίου αλλάζει

Μια επισκόπηση των βασικών μηχανισμών και δομών ενός προγράμματος πλοήγησης σε εικονικούς κόσμους (VRML ή X3D browser) παρουσιάζεται γραφικά στο παρακάτω σχήμα:



Δομή ενός X3D browser

Παρατηρούμε ότι πρώτο βήμα είναι η ανάγνωση των αρχείων (ή έστω ροών δεδομένων) με την περιγραφή του εικονικού κόσμου. Το format μπορεί να ποικίλλει. Έπειτα κατασκευάζεται ο scene graph σταδιακά, καθώς προχωρεί η ανάλυση των δεδομένων εισόδου. Στον ορισμό των νέων κόμβων λαμβάνονται υπόψιν και μηχανισμοί προτυποποίησης (τα prototypes) για τους οποίους θα αναφερθούμε στη συνέχεια. Η προγραμματιστική πρόσβαση στη δομή και τη συμπεριφορά του εικονικού κόσμου, επιτυγχάνεται μέσω δυο APIs, του SAI (Scene Authoring Interface) και του EAI (External Authoring Interface).

3.5 Σύνταξη X3D

Ένα x3d αρχείο, μπορεί να εκφραστεί είτε με το κλασσική VRML είτε σε XML κωδικοποίηση. Τα παρακάτω παραδείγματα αναπαριστούν την ίδια σκηνή και με τις 2 κωδικοποιήσεις:

```
#VRML V2.0 utf8
  Group {
    children [
       Shape {
        appearance DEF BROWN Appearance {
        material Material {
           diffuseColor 0.8 0.6 0.3
       }
      }
       geometry Cylinder {
```

```
        radius 2
       }
      }
   Transform {
      translation 0 2 0
     children [
     Shape {
          appearance USE BROWN
         geometry Cone {
            bottomRadius 2.5
         }
        }
      ]
    }
    ]
}
```

VRML Encoding

```
<?xml version="1.0" encoding="UTF-8"?>                    (1)
<X3D version='3.0' profile='Immersive'>                   (2)

 <head>
   <meta name='filename' content='x3d.x3d/>                (3)
   <meta name='created' content='1 January 2000'/>
   <meta name='author' content='Martha Lasithiotaki'/>
   <meta name='description' content='X3D example in XML encoding'/>
 </head>

  <Scene>
     <Group>
                 <Shape>
                         <Cylinder radius='2'/>
                         <Appearance DEF="BROWN">
                                 <Material diffuseColor='0.8 0.6 0.3'/>
          </Appearance>
                 </Shape>


            <Transform translation='0 2 0'>
                  <Shape>
                         <Cone bottomRadius='2.5'/>
                         <Appearance USE='BROWN'/>
                  </Shape>
           </Transform>

     </Group>
  </Scene>
</X3D>
```

XML Encoding

- Στην πρώτη γραμμή **(1)** ορίζεται η έκδοση της xml που χρησιμοποιείται (version="1.0") καθώς και η κωδικοποίηση που χρησιμοποιείται (encoding="UTF-8")

- Η δεύτερη γραμμή **(2)** αποτελεί την ετικέτα έναρξης του X3d. Εδώ δίνεται τιμή και στην την ιδιότητα profile (profile='Immersive'), το οποίο θα χρησιμοποιηθεί στην X3D εφαρμογή.

- Στην τρίτη γραμμή **(3)** ορίζεται το head tag το οποίο περιλαμβάνει περιεχόμενο metadata για το X3D αρχείο.

- Η τέταρτη γραμμή **(4)** με τις metadata πληροφορίες που παρέχονται, μας επιτρέπει να έχουμε μια καθαρή εικόνα για το αρχείο, δίνοντας μας πληροφορίες για το όνομα του αρχείου, την ημερομηνία δημιουργίας του, το δημιουργό του και μια περιγραφή του αρχείου. Μπορούν να προστεθούν κι άλλες πληροφορίες ανάλογα με τις ανάγκες μας. Αυτό είναι μια σημαντική βελτίωση σε σχέση με τη VRML η οποία παρείχε στο χρήστη τη σύνταξη για σχόλια σκηνής αλλά όχι σαν δομημένα στοιχεία όπως επιτρέπονται μέσω της χρήσης των ετικετών XML.

- Η σκηνή ξεκινάει με τον ορισμό ενός κόμβου τύπου "*Group*", ο οποίος ομαδοποιεί τους υπόλοιπους κόμβους.

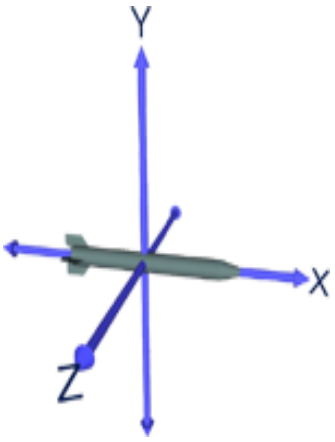
Σημειώνεται ότι σε περιπτώσεις όπως αυτή, όπου όλοι οι κόμβοι είναι παιδιά ενός συγκεκριμένου κόμβου (αυτό δεν είναι απαραίτητο), αυτός ο κόμβος ονομάζεται ριζικός (root node).

- Ως πρώτο παιδί του κόμβου τύπου "*Group*" ορίζεται ένας κόμβος τύπου "*Shape*" που θα περιγράψει το σχήμα που ορίζεται στο επόμενο tag και εμφανισιακά χαρακτηριστικά που θα ορίζει η τιμή του πεδίου "*Appearance*".

- Ο κόμβος τύπου "*Appearance*" ορίζεται και αποκτά το όνομα "BROWN", για να είναι δυνατόν να γίνει αναφορά σε αυτόν από οποιοδήποτε σημείο στη συνέχεια της περιγραφής. Η απόδοση ονόματος γίνεται με τη χρήση της δεσμευμένης λέξης DEF

- Ο κόμβος ομαδοποίησης τύπου "*Transform*" μετακινεί τα παιδιά του, σε σχέση με τις απόλυτες συντεταγμένες του, όσο υποδεικνύει η τιμή του πεδίου "*translation*". Δηλαδή κατά 2 μέτρα στον άξονα Y. Τελική θέση ύψους είναι 2 μέτρα, αφού δεν υπάρχει άλλος μετασχηματισμός θέσης ψηλότερα στην ιεραρχία.

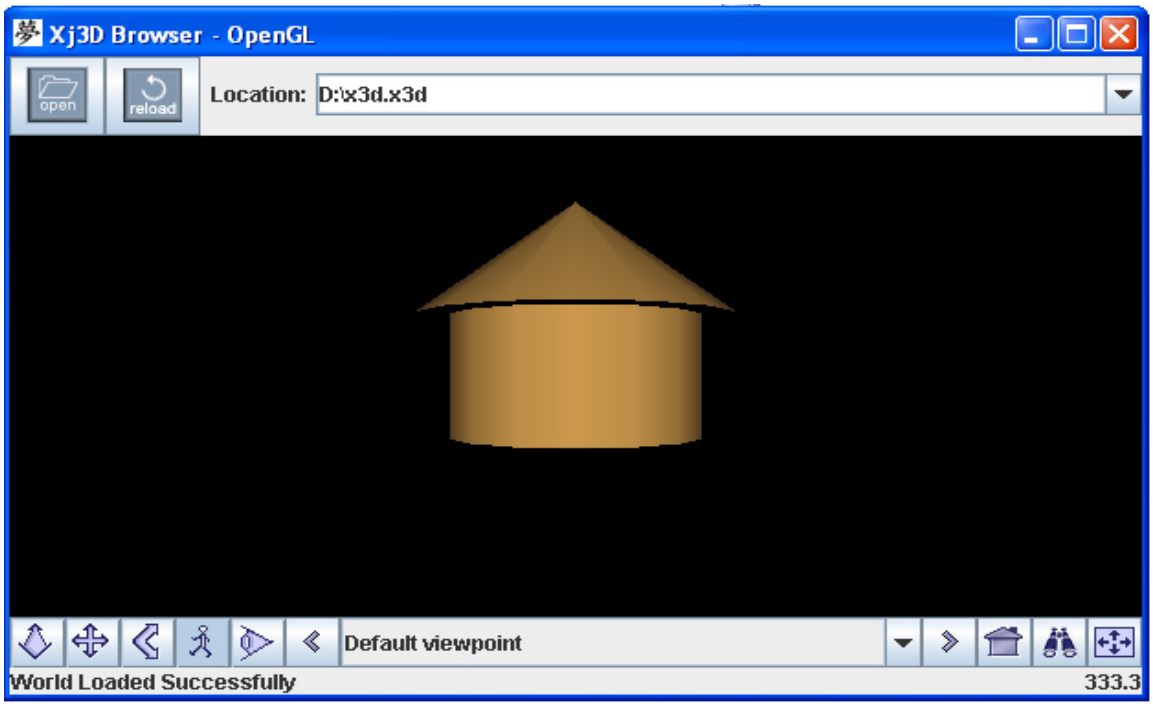
Σημειώνεται ότι το σύστημα συντεταγμένων είναι ένα Καρτεσιανό δεξιόστροφο τριών αξόνων (X, Y, Z), με τον άξονα X να εκτείνεται προς τα δεξιά, τον Y προς τα πάνω, τον Z προς το χρήστη και αρχή όλων το κέντρο της οθόνης (τουλάχιστον στην αρχή πριν ξεκινήσει την περιήγηση στον κόσμο ο χρήστης) και κατά 10 -"εικονικά"- μέτρα πίσω από το σημείο όπου υποτίθεται ότι βρίσκεται ο χρήστης. Η μετακίνηση κατά τον Z θεωρήθηκε σωστή ώστε να μπορεί ο χρήστης άμεσα να εποπτεύσει έναν κόσμο, ο οποίος πιθανόν να επαναχρησιμοποιηθεί ως τμήμα άλλου κόσμου, και για αυτό δε θα έπρεπε να περιέχει μετασχηματισμούς που σχετίζονται με την παρουσίαση και όχι με το περιεχόμενο.



- Παράδειγμα επαναχρησιμοποίησης ενός κόμβου, ο οποίος έχει ονοματιστεί προηγουμένως (σε οποιοδήποτε επίπεδο στην ιεραρχία) με το μηχανισμό της DEF. Την επαναχρησιμοποίηση δηλώνει η λέξη κλειδί **USE** (χρήση: USE <όνομα κόμβου>). Σημαντικό είναι ότι δε γίνεται αντιγραφή, αλλά χρήση του ίδιου αντικειμένου σε πολλά σημεία: ο κόμβος με όνομα "BROWN" αποκτά δύο γονείς και αν κάποια στιγμή αλλάξει το χρώμα του, αυτή η αλλαγή πρέπει να φανεί και στα δυο σχήματα που τον χρησιμοποιούν.

Η εμβέλεια ενός ονόματος κόμβου εκτείνεται από το σημείο που ορίζεται έως το τέλος του ίδιου αρχείου, ορισμού ενός prototype, αλφαριθμητικό ορισμού ενός νέου κόσμου (πχ για δυναμική παραγωγή περιεχομένου από κόμβους τύπου Script). Εάν χρησιμοποιείται το ίδιο όνομα σε πολλές προτάσεις DEF, τότε, στη VRML97 ισχύει ο τελευταίος ορισμός, ενώ στο X3D προτείνεται να αποφεύγεται αυτή η τεχνική, υπονοώντας ότι η περιγραφή μπορεί να είναι άκυρη. Ούτως ή άλλως το ίδιο όνομα περισσότερο από μια φορά μπορεί να προκαλέσει σύγχυση, οπότε σωστό είναι να αποφεύγεται.

Στο ακόλουθο σχήμα φαίνεται ένα στιγμιότυπο του εικονικού κόσμου που περιγράφτηκε:



Εικόνα Ένας απλός εικονικός X3D κόσμος

Οι τύποι των πεδίων των κόμβων όπως προαναφέρθηκε, χωρίζονται σε δυο κατηγορίες:

✓ όσους επιτρέπουν μια απλή τιμή για το συγκεκριμένο πεδίο και όσους επιτρέπουν απόδοση τιμής ως μια διατεταγμένη πλειάδα απλών τιμών, απεριόριστου μήκους. Τα ονόματα των τύπων απλής τιμής έχουν το πρόθεμα "SF", ενώ

✓ οι τύποι πολλαπλών τιμών έχουν όνομα που ξεκινάει με το πρόθεμα "MF". Στον ακόλουθο πίνακα αναγράφονται όλοι οι δυνατοί τύποι πεδίων:

Όνομα τύπου	**Αρχική τιμή**	**Σχόλιο**
SFBool	*FALSE*	Σημαία δυαδικής λογικής (TRUE | FALSE)
MFBool	[]	
SFColor	(0 0 0)	Χρώμα σε μορφή RGB ([0,1]3)
MFColor	[]	
SFColorRGBA	(0 0 0 0)	Χρώμα με Διαφάνεια RGBA ([0,1]4) Τύπος X3D
MFColorRGBA	[]	Τύπος X3D
SFDouble	0.0	Αριθμός κινητής υποδιαστολής διπλής ακρίβειας
MFDouble	[]	
SFFloat	0.0	Αριθμός κινητής υποδιαστολής απλής ακρίβειας
MFFloat	[]	
SFImage	(0 0 0)	Ασυμπίεστη εικόνα ως σειρά bytes. Οι τρεις πρώτοι αριθμοί είναι το πλάτος, ύψος και format της εικόνας
MFImage	[]	Τύπος X3D
SFInt32	0	Ακέραιος με αναπαράσταση στα 32 bits
MFInt32	[]	
SFMatrix3d	[1 0 0 0 1 0 0 0 1]	3x3 πίνακας μετασχηματισμού στις δυο διαστάσεις. Χρησιμοποιείται αριθμητική διπλής ακρίβειας. Τύπος X3D
MFMatrix3d	[]	Τύπος X3D
SFMatrix3f	[1 0 0 0 1 0 0 0 1]	Όπως το SFMatrix3d, μόνο που χρησιμοποιείται αριθμητική απλής ακρίβειας Τύπος X3D
MFMatrix3f	[]	Τύπος X3D
SFMatrix4d	[1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1]	4x4 πίνακας μετασχηματισμού στις τρεις διαστάσεις. Χρησιμοποιείται αριθμητική διπλής ακρίβειας Τύπος X3D

MFMatrix4d	[]	Τύπος X3D
SFMatrix4f	[1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1]	Τύπος X3D
MFMatrix4f	[]	Τύπος X3D
SFNode	*NULL*	Κόμβος
MFNode	[]	
SFRotation	(0 0 1 0)	Περιστροφή. Οι τρεις πρώτοι αριθμοί ορίζουν ένα κανονικοποιημένο διάνυσμα και ο τέταρτος δίνει σε ακτίνια τη δεξιόστροφη περιστροφή γύρω από αυτό το διάνυσμα
MFRotation	[]	
SFString	""	Αλφαριθμητικό με κωδικοποίηση UTF-8
MFString	[]	
SFTime	-1	Αριθμός δευτερολέπτων από τις 00:00:00, την 1-1-1970 στο μεσημβρινό του Γκρίνουϊτς. Ο αριθμός είναι κινητής υποδιαστολής διπλής ακριβείας.
MFTime	[]	
SFVec2d	(0 0)	Δισδιάστατο διάνυσμα (διπλή ακρίβεια) Τύπος X3D
MFVec2d	[]	Τύπος X3D
SFVec2f	(0 0)	
MFVec2f	[]	
SFVec3d	(0 0 0)	Τύπος X3D
MFVec3d	[]	Τύπος X3D
SFVec3f	(0 0 0)	
MFVec3f	[]	

Πίνακας Σύνολο πεδίων X3D-VRML

Η αρχική τιμή δεν αναφέρεται στην αρχική τιμή των πεδίων των κόμβων. Αυτές οι τιμές καθορίζονται στον ορισμό του τύπου του κόμβου και δίνονται, από το συγγραφέα του κόσμου, άλλες (αν χρειάζεται) στη δήλωση του κόμβου στο αρχείο. Οι παραπάνω τιμές είναι οι εναλλακτικές που αποδίδονται όταν κάποιο πρόγραμμα δημιουργεί νέα αντικείμενα αυτού του τύπου. Διότι κάθε τύπος αντιστοιχεί με μια τάξη στις προγραμματιστικές διεπαφές (SAI, EAI). Αυτές οι τιμές, επίσης, δίνονται σε πεδία εισόδου (η έννοια θα εξηγηθεί παρακάτω), όταν αυτά διαβάζονται από κάποιο Script κόμβο, χωρίς να έχει τεθεί η τιμή τους από κάποιο route.

Τα πεδία κάθε τύπου κόμβου κατατάσσονται σε τέσσερις κατηγορίες, αναλόγως με τη λειτουργία τους σε σχέση με το μηχανισμό δυναμικού χειρισμού του κόσμου. Τα πεδία

που ορίζουν σταθερές ιδιότητες του κόμβου, δηλαδή ιδιότητες που τίθενται κατά την ανάγνωση της περιγραφής του κόσμου και δεν αλλάζουν έκτοτε, δηλώνονται ως "**fıeld**". Τα πεδία που λειτουργούν ως είσοδοι και αυτά που λειτουργούν ως έξοδοι πληροφοριών δηλώνονται αντίστοιχα ως "**eventIn**" και "**eventOut**". Υπάρχουν και πεδία τα οποία συνδυάζουν τις τρεις προηγούμενες δυνατότητες και δηλώνονται ως "exposedField". Οι διαφορές θα τονιστούν με την εξήγηση για τα routes στη συνέχεια. Στο πρότυπο X3D όπως έχει προαναφερθεί τα ονόματα για τις δηλώσεις είναι αντίστοιχα "**initializeOnly**", "**inputOnly**", "**outputOnly**" και "**inputOutput**".

3.6 *Επεξεργασία Συμβάντων*

Ήδη έχει αναφερθεί ότι τα routes είναι ιδεατοί αγωγοί μεταφοράς συμβάντων από πεδία εξόδους κόμβων προς πεδία εισόδου άλλων κόμβων. Όταν σε ένα πεδίο, δηλωμένο ως eventOut ή exposedField (outputOnly ή inputOutput), αποδίδεται νέα τιμή, δημιουργείται ένα συμβάν και η νέα τιμή διαδίδεται, μέσω του route, σε ένα άλλο πεδίο ίδιου τύπου, το οποίο είναι δηλωμένο ως eventIn ή exposedField (inputOnly ή inputOutput).

Έστω ότι υπάρχει ένας κόμβος με όνομα NameA, που έχει δυο πεδία eventOut: τα fa1 (SFInt32) και fa2_changed (SFString). Έστω, επίσης, ότι υπάρχει ένας κόμβος NameB με δυο πεδία eventIn: τα fb1 (SFInt32) και set_fb2 (SFString). Τότε οι δυνατοί ορισμοί των routes είναι:

- ROUTE NameA.fa1 TO NameB.fb1
- ROUTE NameA. fa1_ changed TO NameB.fb1
- ROUTE NameA.fa1 TO NameB.fb1_set
- ROUTE NameA. fa1_ changed TO NameB.fb1_set
- ROUTE NameA.fa2_changed TO NameB.set_fb2
- ROUTE NameA.fa2 TO NameB.set_fb2
- ROUTE NameA.fa2_changed TO NameB.fb2
- ROUTE NameA.fa2 TO NameB.fb2

Στην περίπτωση, όπου το πεδίο έχει τύπο πολλαπλών τιμών (MF...) μπορεί στον ορισμό του route να γίνει αναφορά σε συγκεκριμένη απλή τιμή και αυτή η τιμή να συνδεθεί με ένα πεδίο αντίστοιχου τύπου (SF...). Πχ:

```
ROUTE nodeA.MFfield[2] TO nodeB.Sffield
```

Ειδικά με τους κόμβους με πεδία τύπου SFNode ή MFNode, το συντακτικό μπορεί να γίνει ακόμη πιο πολύπλοκο. Πχ:

```
ROUTE nodeA.MFNode-field[2].SFfield TO nodeB.SFNode-field.SFfield
```

Οι προτάσεις "*ROUTE*" μπορούν να εμφανίζονται είτε κάπου μέσα στο αρχείο, εκτός των δηλώσεων των άλλων κόμβων, είτε στο σώμα κόμβων, στις θέσεις που δηλώνονται οι τιμές των πεδίων.

Όπως αναφέρθηκε, ο route graph είναι ο βασικός μηχανισμός που προσδίδει δυναμικότητα στον εικονικό κόσμο και επιτρέπει την πραγμάτωση των

αλληλεπιδραστικών στοιχείων του. Η διαδικασία λειτουργίας αυτού του μηχανισμού είναι επαναλαμβανόμενη και περικλείει την ανάγνωση τιμών από αισθητήρες ή από την κλήση συναρτήσεων που δηλώνονται σε κόμβους τύπου Script. Με το που παράγεται ένα event από έναν αισθητήρα ή ένα κόμβο τύπου Script, το event προωθείται από το πεδίο που παρήγαγε το event μέσω του route σε άλλους κόμβους. Υπάρχει περίπτωση αυτοί οι κόμβοι να αντιδράσουν παράγοντας κι εκείνοι συμβάντα, μέχρι να αποκτήσουν τιμές όλα τα routes. Η διαδικασία αυτή ονομάζεται **event cascade.**

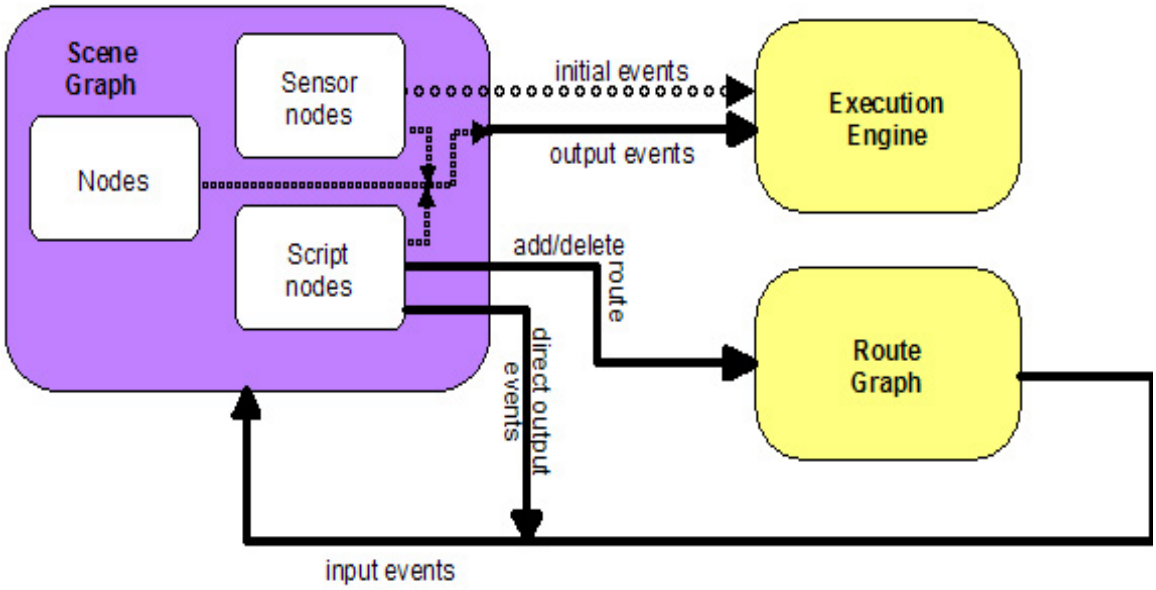
Σε κάθε συμβάν προσάπτεται μια **χρονοσφραγίδα** (timestamp) και τίθεται όριο στην ελάχιστη διαφορά μεταξύ δυο οποιοδήποτε χρονοσφραγίδων: είναι το simulation tick, το οποίο καθορίζει το ρυθμό της προσομοίωσης. Στα συμβάντα που παράγονται κατά τη διάρκεια ενός **event cascade** ορίζεται το ίδιο timestamp με το αρχικό γεγονός, δεδομένου ότι όλα θεωρούνται ότι συνέβησαν την ίδια χρονική στιγμή.

Τα βήματα της διαδικασίας παρουσιάζονται παρακάτω:

1. Καθορισμός τιμής για την ισχύουσα χρονοσφραγίδα
2. Ανανέωση των γραφικών που προβάλλονται στο χρήστη σύμφωνα με την παρούσα θέση του
3. Μετάδοση των συμβάντων που έχουν προστεθεί στην ουρά μέσα από τα αντίστοιχα routes
4. Κλήση της συνάρτησης *shutdown()* για τους κόμβους Script που καταργούνται
5. Μετάδοση τελικών συμβάντων από κόμβους αισθητήρες που καταργούνται
6. Δημιουργία ή/και διαγραφή routes ως αποτέλεσμα της κλήσης των συναρτήσεων *addRoute*, *deleteRoute* από εκτέλεση συναρτήσεων κόμβων τύπου Script
7. Κλήση της συνάρτησης *eventsProcessed* στους κόμβους , που δέχθηκαν συμβάντα κατά τα προηγούμενα βήματα
8. Μετάδοση αρχικών συμβάντων από κόμβους αισθητήρες που ενεργοποιούνται
9. Κλήση της συνάρτησης *initialize* στους κόμβους τύπου Script που μόλις ενεργοποιήθηκαν
10. Έλεγχος για δημιουργία συμβάντων κατά τα βήματα 3 έως 9 και προσθήκη στην ουρά συμβάντων. Αν για κάποιο πεδίο εξόδου υπάρχει ήδη συμβάν με τη τρέχουσα χρονοσφραγίδα, τότε δεν προστίθεται το νέο για αποφυγή ατέρμονων βρόγχων κατά την εκτέλεση.
11. Αν η ουρά συμβάντων δεν είναι άδεια, τότε επιστρέφουμε στο βήμα 3

Όταν ένα event (συμβάν) καταλήγει μέσω πολών routes σε πολλά πεδία (fan-out) ή όταν πολλά routes οδηγούν το ίδιο πεδίο την ίδια χρονική στιγμή (fan-in), τότε η σειρά επεξεργασίας δε θεωρείται σημαντική. Καλό είναι να αποφεύγονται τέτοιες καταστάσεις, ώστε να μην υπάρχουν προβλήματα ασυμβατοτήτων μεταξύ browsers.

Το όλο σύστημα και τα μέρη που αποτελούν το μηχανισμό παρουσιάζονται εποπτικά στο επόμενο σχήμα:



Εικόνα Μοντέλο επεξεργασίας συμβάντων

3.7 Δημιουργία κόμβων X3D με Prototypes

Το X3D διαθέτει ένα μηχανισμό επέκτασης του συνόλου των κόμβων που υποστηρίζει: τα prototypes και external prototypes. Δίνεται η δυνατότητα να οριστούν νέοι τύποι κόμβων και να χρησιμοποιηθούν όπως οι πρότυποι.

Η δήλωση *ProtoDeclare* επιτρέπει τον καθορισμό ενός νέου τύπου κόμβου χρησιμοποιώντας στοιχεία από ήδη ορισμένους prototyped κόμβους. Από τη στιγμή που θα καθοριστεί ένας κόμβος Prototype στη σκηνή, μπορεί να χρησιμοποιηθεί πολλές φορές όπως και τους προκαθορισμένους κόμβους που παρέχει το X3D. Στην περίπτωση που ο prototype κόμβος έχει οριστεί σε ένα εξωτερικό X3D αρχείο, ο συγγραφέας της σκηνής πρέπει να γράψει μια δήλωση ExternProtoDeclare πριν τη χρήση του νέου κόμβου με τη δήλωση *ProtoInstance.* Εκτός αυτού, ο καθορισμός του κόμβου πρέπει να δηλωθεί και στο εξωτερικό X3D έγγραφο καθώς επίσης και X3D στη σκηνή χρησιμοποιώντας το πρωτότυπο. Αυτός ο πλεονασμός είναι μια χαρακτηριστική πηγή λάθους κατά τη διάρκεια της δημιουργίας μιας νέας εφαρμογής. Περαιτέρω στα αντικειμενοστρεφή χαρακτηριστικά γνωρίσματα, όπως η κληρονομιά, ο πολυμορφισμός, μια ασφαλής έννοια τύπων κ.λπ. θα ήταν επιθυμητή. Επομένως τα πρωτότυπα είναι σε αντίθεση με τους ενσωματωμένους δεύτερης θέσης κόμβους κόμβων.

Ακολουθεί ένα παράδειγμα:

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D version='3.0' profile='Immersive'>

  <Scene>
    <ProtoDeclare name='PrototypeSphere'>                                    (1)

      <ProtoInterface>
        <field name='SphereColor' type='SFColor' value='.1 .1 .0' accessType='initializeOnly'/>
      </ProtoInterface>

      <ProtoBody>
       <Shape>
        <Appearance>
          <Material DEF='SphereMaterial'>
            <IS>
              <connect nodeField='diffuseColor' protoField='SphereColor'/>      (2)
            </IS>
          </Material>
        </Appearance>
        <Sphere radius='1.2'/>
       </Shape>
      </ProtoBody>

  </ProtoDeclare> <!-- End of prototype -->

  <ProtoInstance name='PrototypeSphere'>                                     (3)
    <fieldValue name='SphereColor' value='0 0 1'/>
  </ProtoInstance>

 </Scene>
</X3D>
```

Πίνακας Παράδειγμα χρήσης του μηχανισμού prototypes

(1) Χρήση της λέξης "PROTO" για ορισμό ενός νέου τύπου κόμβου. Το όνομα αυτού του τύπου είναι PrototypeSphere. Ορίζεται, επίσης, ότι θα υπάρχει ένα πεδίο τύπου SFColor, το οποίο θα έχει όνομα " SphereColor " και θα μπορεί μόνο να πάρει αρχική τιμή κατά τη δήλωση ενός κόμβου του τύπου PrototypeSphere (field).

(2) Στο σώμα του ορισμού το πεδίο "diffuseColor" συνδέεται με το πεδίο, του κόμβου PrototypeSphere, "SphereColor". Ουσιαστικά το SphereColor είναι ψευδώνυμο για το diffuseColor.

(3) Στη συνέχεια δηλώνεται ένας νέος κόμβος τύπου BoxWithColor και το πεδίο SphereColor αρχικοποιείται σε μια τιμή διαφορετική από την προορισμένη.
Αν το προηγούμενο αρχείο είχε όνομα "*ProtoExample.wrl*" και χρειάζεται να δημιουργηθεί ένας κόμβος τύπου BoxWithColor σε άλλο αρχείο, τότε μπορεί να χρησιμοποιηθεί ο μηχανισμός των external prototypes για να γίνει αναφορά στον υπάρχον ορισμό:

```
#VRML V2.0 utf8

 EXTERNPROTO BoxFromLibrary [field SFColor color]


["ProtoExample.wrl#BoxWithColor"]

 BoxFromLibrary { color 1 1 0 }
```

Πίνακας Παράδειγμα μηχανισμού external prototypes

- Χρήση της λέξης "EXTERNPROTO" για ορισμό ενός νέου τύπου κόμβου, ο οποίος ορίζεται επακριβώς σε εξωτερικό αρχείο. Το όνομα αυτού του τύπου θα είναι "BoxFromLibrary", ανεξαρτήτως του ονόματος τύπου που χρησιμοποιείται για στον αρχικό ορισμό. Ορίζεται, επίσης, ότι θα υπάρχει ένα πεδίο τύπου SFColor, το οποίο θα έχει όνομα "color" και θα μπορεί μόνο να πάρει αρχική τιμή κατά τη δήλωση ενός κόμβου του τύπου *BoxFromLibrary* (field). Σε περίπτωση που δε υπήρχε ανάγκη αλλαγής της τιμής του πεδίου *color*, θα μπορούσε να παραληφθεί η ρητή αναφορά σε αυτό και το σώμα των πεδίων να μείνει κενό.
- Δίνεται το *URL* του αρχείο αναφοράς ακολουθούμενο από το όνομα του τύπου που αναφέρεται.
- Δηλώνεται ένας νέος κόμβος τύπου BoxFromLibrary και το πεδίο color αρχικοποιείται σε μια τιμή διαφορετική από την προορισμένη.

Το πρότυπο X3D διευκολύνει το μηχανισμό των external prototypes, ορίζοντας τις προτάσεις "Import" και "Export". Με τη χρήση αυτών δε χρειάζεται η αναλυτική παράθεση των χρησιμοποιούμενων πεδίων ενός EXTERNPROTO. Έτσι διευκολύνεται η επαναχρησιμοποίηση κώδικα και μειώνεται ο όγκος των αρχείων (πράγμα αρκετά σημαντικό για χρήση στον ιστό).

3.8 Scripting

Ο τύπος κόμβου Script είναι πολύ σημαντικός γιατί επιτρέπει την εκτέλεση προγραμμάτων ορισμένων από το συγγραφέα του εικονικού κόσμου. Είναι αντίστοιχο με το tag script της HTML. Χωρίς αυτόν η όποια δυναμική συμπεριφορά του εικονικού κόσμου θα περιοριζόταν από τη λογική των προτυποποιημένων κόμβων, δηλαδή στους κόμβους αισθητήρες (συμπεριλαμβανομένου και του TimeSensor) και στους κόμβους παρεμβολής.

Ο κόμβος Script προσφέρει τη δυνατότητα να εκτελεστούν προγράμματα γραμμένα σε οποιαδήποτε γλώσσα υποστηρίζει ο VRML-X3D browser. Κατ' ελάχιστον υποστηρίζεται η JavaScript (ή πιο σωστά η γλώσσα ECMAScript, σύμφωνα με το πρότυπο ISO/IEC DIS 16262 – http://www.ecma.ch) και η Java.

Οι ανεξαρτήτως γλώσσας υπηρεσίες, που παρέχονται στον προγραμματιστή, από τον κόμβο Script, ορίζονται από το ίδιο το πρότυπο, για τις δυο γλώσσες, στη VRML97, οπότε και η υποστήριξη για άλλες γλώσσες θα πρέπει να ακολουθεί το παράδειγμα των

δυο γλωσσών, με κριτήριο την εύκολη μετατροπή κώδικα από ήδη υποστηριζόμενη γλώσσα στη στοχευόμενη γλώσσα. Το X3D ορίζει άμεσα μια, ανεξαρτήτως γλώσσας υλοποίησης, προγραμματιστική διασύνδεση: το SAI (***S****cene* ***A****uthoring* ***I****nterface*), το οποίο αποτελεί κοινό σύνολο υπηρεσιών για προγράμματα κόμβων τύπου Script, αλλά και για εξωτερικά προγράμματα που αλληλεπιδρούν με το σύστημα χρόνου εκτέλεσης του εικονικού κόσμου.

3.9 Αισθητήρες X3D

Όπως έχει ήδη αναφερθεί ένα αρχείο x3d περιγράφει 2 είδη γράφων. Το γράφο ιεραρχίας του τρισδιάστατου σκηνικού μοντέλου (**scene graph transformation hierarchy**) και το γράφο αλληλεπίδρασης μεταξύ των αντικειμένων του κόσμου (**behaviour graph** ή **route graph**).

Σκοπός μας είναι να αναλύσουμε το γράφο αλληλεπίδρασης του αρχικού X3D αρχείου, ο οποίος μπορεί να έχει χωριστά δέντρα γεγονότων (να αποτελείται από γεγονότα που δεν σχετίζονται μεταξύ τους).

Όταν ένας χρήστης αλληλεπιδρά με ένα 3D κόσμο, μπορούν να δημιουργηθούν ένα ή πολλαπλά γεγονότα, με το να έχουμε διαδοχικές αντιδράσεις ανταλλαγής δεδομένων μεταξύ των κόμβων. Οι κόμβοι που δέχονται γεγονότα μπορεί να αντιδράσουν αλλάζοντας την κατάσταση του κόσμου ή αν δημιουργήσουν νέα γεγονότα. Με αυτό το τρόπο, το γεγονός που παράγεται από την αλληλεπίδραση χρηστών διαδίδεται στο σύνολο ή το μέρος του τρισδιάστατου κόσμου και αλλάζει την κατάσταση του συστήματος.

Το X3D διαθέτει τους κόμβους αισθητήρες (Sensor Nodes) οι οποίοι είναι υπεύθυνοι για την ανίχνευση ενεργειών που έχουν γίνει από τους χρήστες, αλλαγών στον τρόπο εξέτασης του περιβάλλοντος και μηνυμάτων από το δίκτυο. Με άλλα λόγια στο γράφο ιεραρχίας (**scene graph**) ενός X3D κόσμου, ο αρχικός κόμβος (**root node**) ενός δέντρου γεγονότων είναι ένας κόμβος αισθητήρα (**sensor node**). Υπάρχουν πέντε τύποι από sensor nodes:

3.9.1 Pointing device sensors (Αισθητήρας συσκευής δείκτη)

Οι αισθητήρες συσκευής δείκτη ανιχνεύουν συμβάντα από σημεία που υποδεικνύει ο χρήστης μέσα σε ένα ορισμένο γεωμετρικό χώρο (πχ TouchSensor). Οι ακόλουθοι τύποι κόμβων, είναι αισθητήρες συσκευής κατάδειξης:

- **CylinderSensor**

Ο κόμβος κυλινδρικού αισθητήρα (CylinderSensor) χαρτογραφεί την κίνηση δεικτών του χρήστη (π.χ., ποντίκι ή ράβδος) σε μια περιστροφή πάνω σε έναν αόρατο κύλινδρο που ευθυγραμμίζεται με τον Y-άξονα του τοπικού συστήματος συντεταγμένων. Ο

CylinderSensor χρησιμοποιεί την γεωμετρία που κληρονομεί από τους κόμβους των γονέων του για να καθορίσει εάν είναι πρέπει ν α παραγάγει γεγονότα.

Πεδία του CylinderSensor

```
CylinderSensor : X3DDragSensorNode {
  SFBool     [in,out] autoOffset         TRUE
  SFString   [in,out] description       ""
  SFFloat    [in,out] diskAngle         π/12  (0,π/2)
  SFBool     [in,out] enabled           TRUE
  SFFloat    [in,out] maxAngle           -1    [-2π,2π]
  SFNode     [in,out] metadata           NULL  [X3DMetadataObject]
  SFFloat    [in,out] minAngle           0     [-2π,2π]
  SFFloat    [in,out] offset            0     (-∞,∞)
  SFBool     [out]
  SFBool     [out]    isOver
  SFRotation [out]    rotation_changed
  SFVec3f    [out]    trackPoint_changed
}
```

Το πεδίο enabled μπορεί να ενεργοποιήσει ή να θέσει εκτός λειτουργίας τον κόμβο CylinderSensor. Οι τιμές που μπορεί να πάρει ο αισθητήρας είναι TRUE και FALSE. Στην πρώτη περίπτωση ο αισθητήρας αντιδρά κατάλληλα στα γεγονότα χρηστών, ενώ στη δεύτερη ο αισθητήρας δεν ανιχνεύει την εισαγωγή δεδομένων του χρήστη ή δεν στέλνει τα γεγονότα.

Ο CylinderSensor παράγει γεγονότα όταν ενεργοποιείται η συσκευή δείκτη και ο δείκτης δείχνει σε κόμβους που είναι παιδιά της ομάδας γονέων του αισθητήρα.

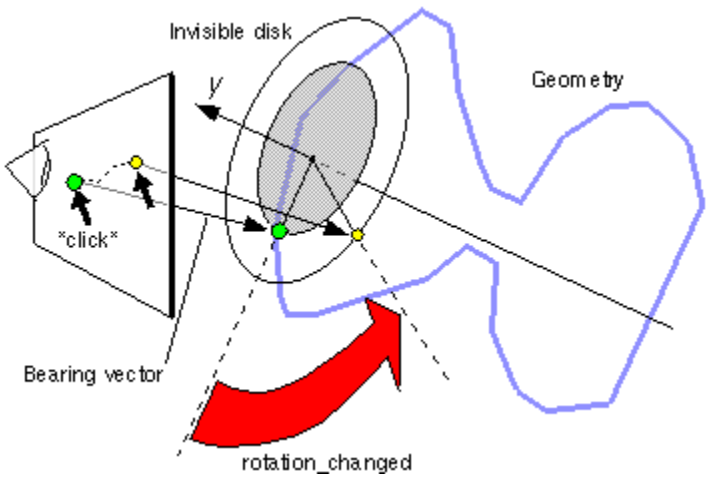
Με την ενεργοποίηση της συσκευής δείκτη και καθώς υποδεικνύεται η γεωμετρία του αισθητήρα, στέλνεται ένα γεγονός *isActive* με την τιμή TRUE. Η αρχική οξεία γωνία μεταξύ του φέροντος διανύσματος διόπτευσης[1] και του τοπικού Y-άξονα του κόμβου CylinderSensor καθορίζει εάν οι πλευρές του αόρατου κυλίνδρου χρησιμοποιούνται για χειρισμό.

Εάν η αρχική γωνία είναι μικρότερη από την τιμή του πεδίου *diskAngle*, ο γεωμετρικός χώρος του αισθητήρα αντιμετωπίζεται ως απείρως μεγάλος δίσκος που βρίσκεται στο τοπικό Y=0 και συμπίπτει με το αρχικό σημείο διατομής. Η κίνηση συρσίματος του αισθητήρα χαρτογραφείται σε μια περιστροφή γύρω το διάνυσμα του άξονα Y. Το κάθετο διάνυσμα από το αρχικό σημείο διατομής με τον άξονα Y καθορίζει τη μηδενική περιστροφή γύρω από τον άξονα. Για κάθε επόμενη θέση του δείκτη στέλνεται ένα γεγονός *rotation_changed* που ισούται με το άθροισμα της περιστροφής γύρω από τον άξονα Y (από την αρχική προς τη νέα διατομή) συν την τιμή του πεδίου offset.

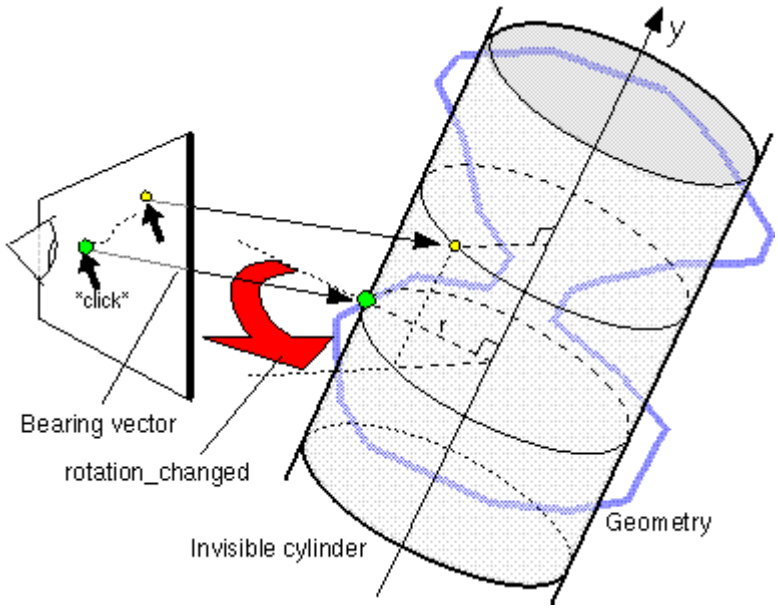
Τα συμβάντα trackPoint_changed ανιχνεύουν το θέση συρσίματος στην επιφάνεια αυτού του δίσκου. Όταν η συσκευή δείκτη απενεργοποιείται και η τιμή του πεδίου

[1] Διάνυσμα Διόπτευσης είναι το διάνυσμα που προκύπτει από την προβολή του σημείου όπου ανιχνεύθηκε κίνηση του χρήστη, στον φανταστικό κύλινδρο του αισθητήρα.

autoOffset είναι TRUE, το πεδίο OFFSET παίρνει την τελευταία τιμή του *rotation_changed* και παράγεται ένα συμβάν *offset_changed.*



Εάν η αρχική οξεία γωνία μεταξύ του φέροντος διανύσματος διόπτευσης και του τοπικού Y-άξονα του κόμβου CylinderSensor είναι μεγαλύτερη ή ίση με τη γωνία diskAngle, τότε ο αισθητήρας συμπεριφέρεται όπως έναν κύλινδρο. Η μικρότερη απόσταση μεταξύ του σημείου της διατομής (μεταξύ της γεωμετρίας της διατομής και του αισθητήρα) και του άξονα Y και του συστήματος συντεταγμένων καθορίζει την ακτίνα ενός αόρατου κυλίνδρου που χρησιμοποιείται για να χαρτογραφήσει την κίνηση του δείκτη και χαρακτηρίζει μια μηδενική αξία περιστροφής. Για κάθε επόμενη θέση του δείκτη στέλνεται ένα *rotation_changed* που ισούται με το άθροισμα της δεξιόστροφη περιστροφής από την αρχική διατομή σε αναφορά με το διάνυσμα του άξονα Y συν την τιμή του πεδίου *offset.*



Τα *trackPoint_changed* απεικονίζουν τη σταθερή θέση συρσίματος πάνω στη επιφάνεια του αόρατου κυλίνδρου. Όταν η συσκευή δείκτη απενεργοποιείται και η τιμή του πεδίου *autoOffset* είναι TRUE, το πεδίο *offset* παίρνει την τελευταία τιμή της γωνίας περιστροφής και παράγεται ένα συμβάν *offset_changed.*

Όταν ο αισθητήρας παράγει ένα συμβάν isActive με τιμή TRUE, συλλέγει όλα τα μετέπειτα γεγονότα κινήσεων της συσκευής δείκτη μέχρι η συσκευή να απελευθερωθεί και ένα ψεύτικο isActive (isActive=FALSE) συμβάν. Η κίνηση του δείκτη, για όσο η συσκευή δείκτη είναι ενεργοποιημένη αναφέρεται σαν λειτουργία συρσίματος ("drag" operation)

Ενώ η συσκευή δείκτη ενεργοποιείται, trackPoint_changed και rotation_changed γεγονότα παράγονται και ερμηνεύονται από την κίνηση του δείκτη βασισμένα στο τοπικό σύστημα συντεταγμένων του αισθητήρα. Τα γεγονότα trackPoint_changed αντιπροσωπεύουν τα σταθερά σημεία διατομής στην επιφάνεια του αόρατου κυλίνδρου ή του δίσκου. Αν η αρχική γωνία οδηγεί σε κυλινδρική περιστροφή (σε αντιδιαστολή με τη συμπεριφορά δίσκων) και η συσκευή δείκτη συρθεί μακριά από τον κύλινδρο ενώ είναι ενεργοποιημένη, οι browsers μπορούν να το μεταφράσουν με ποικίλους τρόπους (πχ να σταθεροποιεί όλες τις τιμές στον κύλινδρο και να συνεχίζει την περιστροφή όσο ο δείκτης είναι μακριά από τον κύλινδρο). Κάθε κίνηση της συσκευής δείκτη όσο το πεδίο *isActive* έχει την τιμή TRUE δημιουργεί *trackPoint_changed* και *rotation_changed* συμβάντα.

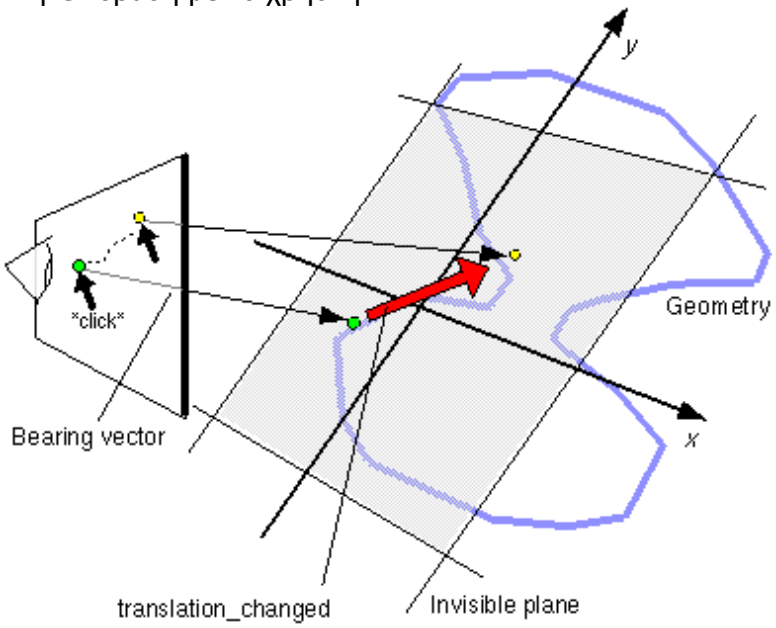
Τα πεδία *minAngle* και *maxAngle* σταθεροποιούν τα γεγονότα *rotation_changed* σε ένα εύρος τιμών. Αν η τιμή του πεδίου *minAngle* είναι μεγαλύτερη από την τιμή του πεδίου *maxAngle,* τα συμβάντα *rotation_changed* δεν σταθεροποιούνται. Τα πεδία *minAngle* και *maxAngle* περιορίζονται στο ευρος τιμών [-2π,+2π].

- **PlaneSensor**

Ο κόμβος PlaneSensor χαρτογραφεί την κίνηση του δείκτη σε ένα δισδιάστατο χώρο όπου πάντα το Z λογίζεται μηδέν στο τοπικό σύστημα συντεταγμένων. Ο κόμβος PlaneSensor χρησιμοποιεί τη γεωμετρία που κληρονομεί από το γονικό κόμβο για να καθορίσει έαν υποχρεούται να παράγει συμβάντα.

```
PlaneSensor : X3DDragSensorNode {
  SFBool   [in,out] autoOffset         TRUE
  SFString [in,out] description        ""
  SFBool   [in,out] enabled            TRUE
  SFVec2f  [in,out] maxPosition         -1 -1 (-∞,∞)
  SFNode   [in,out] metadata            NULL  [X3DMetadataObject]
  SFVec2f  [in,out] minPosition         0 0   (-∞,∞)
  SFVec3f  [in,out] offset             0 0 0 (-∞,∞)
  SFBool   [out]    isActive
  SFBool   [out]    isOver
  SFVec3f  [out]    trackPoint_changed
  SFVec3f  [out]    translation_changed
}
```

Το πεδίο enabled μπορεί να ενεργοποιήσει ή να θέσει εκτός λειτουργίας τον κόμβο PlaneSensor. Εάν το πεδίο *enabled* έχει την τιμή *TRUE* ο αισθητήρας αντιδρά κατάλληλα στα γεγονότα χρηστών, ενώ στην περίπτωση που έχει την τιμή *FALSE* ο αισθητήρας δεν μπορεί να ανιχνεύσει κινήσεις του χρήστη ή να στείλει τα γεγονότα. Αν το πεδίο *enabled* λάβει ένα γεγονός με τιμή *FALSE* και το πεδίο *isActive* έχει την τιμή *TRUE*, ο αισθητήρας τίθεται εκτός λειτουργίας, παράγει ένα γεγονός γεγονός *FALSE* στο πεδίο *isActive*. Στη περίπτωση που λάβει ξανά ένα γεγονός *TRUE* ενεργοποιείται και είναι έτοιμος για αλληλεπίδραση με το χρήστη.



Ο κόμβος PlaneSensor δημιουργεί γεγονοτα όταν η συσκευή δείκτη είναι ενεργοποιημένη και ο δείκτης δείχνει σε Απόγονους γεωμετρικούς κόμβους της ομάδας γονέων του αισθητήρα.

Με την ενεργοποίηση της συσκευής δείκτη (κλικ ποντικιού) δείχνοντας τη γεωμετρία του αισθητήρα ένα γεγονός *sActive* TRUE στέλνεται. Η κίνηση δεικτών χαρτογραφείται σε σχετική μετατόπιση στο επίπεδο. (ένα παράλληλο επίπεδο στο τοπικό επίπεδο του αισθητήρα με Z=0 που συμπίπτει με το αρχικό σημείο της διατομής). Για κάθε επόμενη μετακίνηση του διανύσματος διόπτευσης ένα γεγονός translation_changed παράγεται το οποίο αντιστοιχεί στο άθροισμα της σχετικής μετατόπισης από το αρχικό σημείο διατομής στο σημείο διατομής του νέου διανύσματος διόπτευσης στο επίπεδο συν την τιμή του πεδίου offset.Το σημείο της μετατόπισης καθορίζεται από το επίπεδο συντεταγμένων του αισθητήρα όπου το Z πάντα ισούται με μηδέν. Όταν η συσκευή δείκτη απενεργοποιείται και η τιμή του πεδίου *autoOffset* είναι *TRUE,* η τιμή του πεδίου *offset* αλλάζει στην τελευταία τιμή του πεδίου *translation_changed* και παράγεται ένα γεγονός *offset_changed.*

Με το που ενεργοποιείται ο αισθητήρας (παράγει ένα γεγονός *isActive = TRUE*), συλλέγει όλα τα μετέπειτα γεγονότα κίνησης από την συσκευή δείκτη έως ότου απενεργοποιηθεί (παράγοντας ένα γεγονός *isActive = FALSE*). Άλλοι αισθητήρες συσκευής δείκτη δεν παράγουν γεγονότα κατά τη διάρκεια αυτής της περιόδου. Η κίνηση του δείκτη όσο η

συσκευή δείκτη είναι ενεργοποιημένη (*isActive = TRUE*) αναφέρεται σαν λειτουργία έλξης.

Τα πεδία *minPosition* και *maxPosition δίνουν ένα όριο στις τιμές που μπορούν να πάρει το πεδίο translation_changed* (πάντα το επίπεδο Z έχει την τιμή μηδέν) και κατά συνέπεια τίθεται ένα όριο στα γεγονότα *translation_changed* που μπορούν να παραχθούν. Αν η τιμές X και Y του πεδίου *minPosition* είναι μεγαλύτερες από τις αντίστοιχες τιμές του πεδίου *maxPosition,* τότε είναι δυνατόν να δηλωθεί ένα σταθερό εύρος τιμών που μπορεί να πάρει το πεδίο *translation_changed.* Στην περίπτωση που οι τιμές X και Y του πεδίου *minPosition* είναι ίσες με τις αντίστοιχες τιμές του πεδίου *maxPosition* τότε το πεδίο προσαρμόζεται στην τιμή που δίνεται από το χρήστη.

Συνοψίζοντας:
Όταν η συσκευή δείκτη είναι ενεργή και ο δείκτης έχει μετακινηθεί από το χρήστη παράγονται γεγονότα *trackPoint_changed* και *translation_change.* Τα *trackPoint_changed* αναπαριστούν τα μη σταθερά σημεία διατομής στην επιφάνεια του επιπέδου. Κάθε κίνηση της συσκευής δείκτη όσο το πεδίο *isActive = TRUE*, παράγει γεγονότα *trackPoint_changed* και *translation_changed.*

```
<X3D profile="Immersive" >
<Scene>
        <PlaneSensor DEF="PLANE_SENSOR"/>                    (1)

        <Transform DEF="BOX">
      <Shape>
        <Box size='3 3.2 1.0'/>
        <Appearance>
            <Material diffuseColor="1 0.2 0.7"/>
        </Appearance>
      </Shape>
        </Transform>


   <ROUTE fromField="translation_changed" fromNode="PLANE_SENSOR"      (2)
toField="set_translation" toNode="BOX"/>

</Scene>
</X3D>
```

Αρχείο X3D με αισθητήρα PlaneSensor

(1) Εδώ δημιουργείται ο αισθητήρας επιπέδου (PlaneSensor) με το όνομα PLANE_SENSOR (DEF = "PLANE_SENSOR")
(2) Εδώ έχουμε τη μεταφορά συμβάντων από το πεδίο *translation_changed* του κόμβου *PLANE_SENSOR* στο πεδίο *set_translation* του κόμβου *BOX.* Πιο συγκεκριμένα μεταφέρονται οι συνταγμένες (X-Y) που συνέλεξε ο αισθητήρας από την κίνηση του δείκτη, στο γεωμετρικό κόμβο BOX. Ο κόμβος BOX έχει τώρα τις συντεταγμένες που έλαβε από τον αισθητήρα.

- **SphereSensor**

Ο κόμβος SphereSensor χαρτογραφεί την κίνηση του δείκτη σε ένα μια σφαιρική περιστροφή στο τοπικό σύστημα συντεταγμένων. Ο κόμβος SphereSensor χρησιμοποιεί τη γεωμετρία που κληρονομεί από το γονικό κόμβο για να καθορίσει έαν υποχρεούται να παράγει συμβάντα.

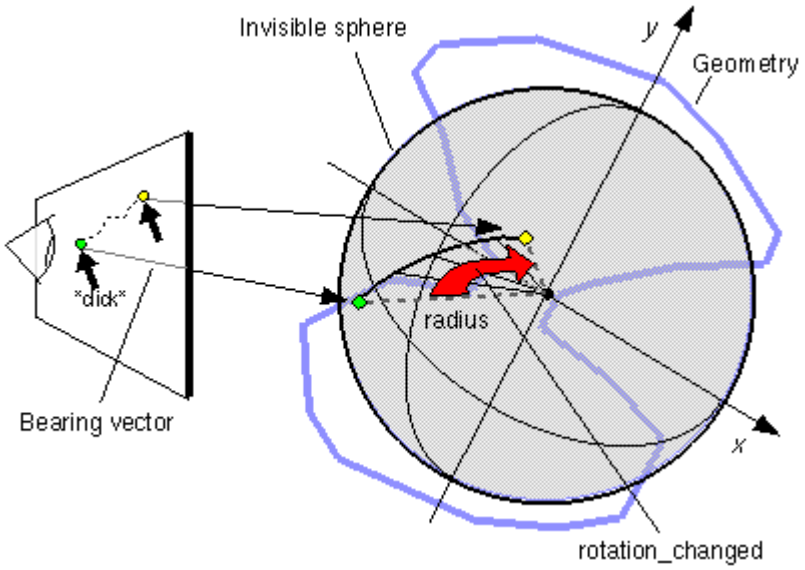
```
SphereSensor : X3DDragSensorNode {
  SFBool     [in,out] autoOffset         TRUE
  SFString   [in,out] description       ""
  SFBool     [in,out] enabled           TRUE
  SFNode     [in,out] metadata           NULL    [X3DMetadataObject]
  SFRotation [in,out] offset            0 1 0 0 [-1,1],(-∞,∞)
  SFBool     [out]    isActive
  SFBool     [out]    isOver
  SFRotation [out]    rotation_changed
  SFVec3f    [out]    trackPoint_changed
}
```

Το πεδίο *enabled* του αισθητήρα μπορεί να ενεργοποιήσει η να απενεργοποίησει τον αισθητήρα. Στην περίπτωση που το πεδίο αυτό έχει την τιμή TRUE, ο αισθητήρας μπορεί να αντιδράσει κατάλληλα στα συμβάντα του χρήστη, σε αντίθετη περίπτωση (*enabled = FALSE*) δεν μπορεί να ανιχνεύσει κινήσεις του χρήστη ή να στείλει τα γεγονότα. Ο συγκεκριμένος αισθητήρας παράγει γεγονότα όταν η συσκευή δείκτη και ο δείκτης δείχνει σε γεωμετρικά σχήματα που η και ο δείκτης δείχνει σε απόγονους γεωμετρικούς κόμβους της ομάδας γονέων του αισθητήρα.

Το διάνυσμα που ορίζεται από το αρχικό σημείο της διατομής της γεωμετρίας του αισθητήρα και την τοπική προέλευση καθορίζει την ακτίνα της σφαίρας που χρησιμοποιείται για να χαρτογραφήσει την επόμενη δείχνοντας κίνηση συσκευών σέρνοντας.

Η γωνία που ορίζεται από το αρχικό σημείο διατομής της γεωμετρίας του αισθητήρα, με τις συντεταγμένες του σημείου προέλευσης και κορυφή το κέντρο της σφαίρας δημιουργούν ένα τόξο οι μοίρες του οποίου καθορίζουν το χρησιμοποιείται για να χαρτογραφήσει επόμενες κινήσεις του δείκτη καθώς σύρεται. Η εικονική σφαίρα που ορίζεται από αυτή τη γωνία χρησιμοποιείται για να ερμηνεύσει επόμενες κινήσεις της συσκευής δείκτη και δεν επηρεάζεται από οποιεσδήποτε αλλαγές του συστήματος συντεταγμένων του αισθητήρα όσο ο αισθητήρας είναι ενεργός.

Για κάθε θέση του διανύσματος διόπτευσης, στέλνεται ένα *rotation_changed* γεγονός το οποίο ισούται με το άθροισμα της σχετικής περιστροφής από το αρχικό σημείο διατομής συν την τιμή του πεδίου offset. Τα συμβάντα *trackPoint_changed* απεικονίζουν τη σταθερή θέση συρσίματος στην επιφάνεια της σφαίρας. Όταν η συσκευή δείκτη απενεργοποιείται δηλαδή το *autoOffset* παίρνει την τιμή *true,* το *offset* παίρνει την τελευταία τιμή του πεδίου *rotation_changed* και παράγεται ένα *offset_changed* γεγονός.



Όταν ο αισθητήρας παράγει ένα συμβάν isActive με τιμή TRUE, συλλέγει όλα τα μετέπειτα γεγονότα κινήσεων της συσκευής δείκτη μέχρι η συσκευή να απελευθερωθεί και ένα ψεύτικο isActive (isActive=FALSE) συμβάν. Η κίνηση του δείκτη, για όσο η συσκευή δείκτη είναι ενεργοποιημένη αναφέρεται σαν λειτουργία συρσίματος ("drag" operation)

Συνοψίζοντας:
Όταν η συσκευή δείκτη είναι ενεργή και ο δείκτης έχει μετακινηθεί από το χρήστη παράγονται γεγονότα *trackPoint_changed* και *rotation_changed.* Τα *trackPoint_changed γεγονότα* αναπαριστούν τα μη σταθερά σημεία διατομής στην επιφάνεια της αόρατης σφαίρας.

```
<X3D profile="Immersive" >

 <Scene>
       <SphereSensor DEF='Sphere_SENSOR'/>                    (1)
        <Transform DEF='MYBOX'>
          <Shape>
            <Appearance>
              <Material diffuseColor='1 1 0'/>
            </Appearance>
            <Box/>
          </Shape>
        </Transform>

    <ROUTE fromField='rotation_changed' fromNode='Sphere_SENSOR'  (2)
toField='set_rotation' toNode='MYBOX'/>

</Scene>
</X3D>
```

Αρχείο X3D με αισθητήρα SphereSensor

(1) Εδώ δημιουργείται ο σφαιρικός αισθητήρας (SphereSensor) με το όνομα 'Sphere_SENSOR' (DEF='Sphere_SENSOR')

(2) Εδώ έχουμε τη μεταφορά συμβάντων από το πεδίο rotation_changed του κόμβου *Sphere_SENSOR* στο πεδίο *set_rotation* του κόμβου *MYBOX.* Πιο συγκεκριμένα, κάθε φορά που ο χρήστης προκαλεί κίνηση στο γεωμετρικό κόμβο "παιδί" (BOX) του αισθητήρα, η τιμή της μεταβολής της περιστροφής μεταφέρεται στο πεδίο set_rotation του κόμβου *MYBOX.*

- **TouchSensor (Αισθητήρας αφής)**

Ένας κόμβος TouchSensor ακολουθεί τη θέση και την κατάσταση της συσκευής δείκτη και ανιχνεύει πότε τα σημεία χρηστών που έχει επιλέξει ο χρήστης περιλαμβάνονται στη γεωμετρία του γονέα του κόμβου TouchSensor. Ένας κόμβος αισθητήρα αφής μπορεί να ενεργοποιηθεί ή να απενεργοποιηθεί με το να του αποστέλλει ένα συμβάν με τιμή TRUE ή FALSE. Αν ο αισθητήρας αφής είναι ανενεργός, δεν ακολουθεί τη θέση της συσκευής δείκτη και δεν μπορεί να στείλει γεγονότα.

Ο αισθητήρας αφής παράγει γεγονότα, όταν η συσκευή δείκτη δείχνει σε γεωμετρικούς κόμβους που είναι απόγονοι της ομάδας γονέων του.

```
TouchSensor : X3DTouchSensorNode {
  SFString [in,out] description         ""
  SFBool   [in,out] enabled             TRUE
  SFNode   [in,out] metadata              NULL [X3DMetadataObject]
  SFVec3f  [out]    hitNormal_changed
```

```
  SFVec3f  [out]    hitPoint_changed
  SFVec2f  [out]    hitTexCoord_changed
  SFBool   [out]    isActive
  SFBool   [out]    isOver
  SFTime   [out]    touchTime
}
```

Ο πεδίο **isOver** απεικονίζει την κατάσταση της συσκευής δείκτη αναφορικά με το αν η συσκευή δείχνει προς τη γεωμετρία του κόμβου TouchSensor ή όχι. Όταν η συσκευή δείκτη αλλάζει κατάσταση από μια θέση της οποίας η διόπτευση δεν τέμνει κανένα γεωμετρικό κόμβο που να είναι απόγονος των γονέων του TouchSensor, σε μια άλλη η οποία τέμνει, ένα αληθές συμβάν *isOver* (*isOver =* TRUE) δημιουργείται. Όταν η συσκευή δείκτη μετακινείται από ένα σημείο του οποίου το διάνυσμα διόπτευσης τέμνει τη γεωμετρία σε ένα άλλο το οποίο δεν την τέμνει ή καποία άλλη γεωμετρία εμποδίζει τη γεωμετρία του αισθητήρα, τότε παράγεται ένα ψευδές συμβάν *isOver* (*isOver =* FALSE). Τα παραπάνω γεγονότα παράγονται μόνο όταν η συσκευή δείκτη και έχει αλλάξει κατάσταση over (μετάβαση από το να δείχνει σε κόμβο τύπου TouchSensor στο να μη δείχνει, και το αντίθετο). Τα γεγονότα αυτά επίσης δεν παράγονται αν από μόνος του ο γεωμετρικός κόμβος κινείται χωρίς να απαιτείται παρέμβαση από το χρήστη.

Όσο ο χρήστης μετακινεί το σημείο διόπτευσης στη γεωμετρία του αισθητήρα αφής, καθορίζεται το σημείο διατομής ανάμεσα στο σημείο διόπτευσης και της γεωμετρίας. Κάθε κίνηση του δείκτη όσο το πεδίο *isOver* έχει την τιμή TRUE, δημιουργεί *hitPoint_changed*, *hitNormal_changed και hitTexCoord_changed* συμβάντα. Τα hitPoint_changed συμβάντα περιέχουν το τρισδιάστατο σημείο του γεωμετρικού κόμβο πάνω στο οποίο έκανε κλικ ο χρήστης.
Το πεδίο *touchTime* ενεργοποιείται όταν όλες από τις παρακάτω συνθήκες αληθεύουν:
1) Η συσκευή δείκτη έδειχνε γεωμετρικό κόμβο (απόγονο των γονέων του αισθητήρα αφής) όταν ενεργοποιήθηκε αρχικά (isActive =TRUE)
2) Η συσκευή δείκτη δείχνει ακόμη στον παραπάνω γεωμετρικό κόμβο (isOver =TRUE).
3) Η συσκευή δείκτη απενεργοποιείται (παράγεται ένα συμβάν isActive = FALSE).

Η τιμή του πεδίου *touchTime* ισούται με την ώρα κατά την οποία ο χρήστης έκανε κλικ στο γεωμετρικό κόμβο.

3.9.2 **Environment sensors** (Αισθητήρες Περιβάλλοντος)

Οι αισθητήρες περιβάλλοντος είναι κόμβοι που στέλνουν συμβάντα βασισμένα σε κάποιο συμβάν που εμφανίζεται μέσα στο περιβάλλον, συνήθως μια αλληλεπίδραση μεταξύ δύο στοιχείων μέσα στον κόσμο.

Τα περισσότερα συμβάντα που προκαλούνται από Environment sensors (αισθητήρες περιβάλλοντος), δημιουργούνται λόγω μιας αλληλεπίδρασης μεταξύ του θεατή και του κόσμου. Εντούτοις, ένα γεγονός αισθητήρων περιβάλλοντος μπορεί να εμφανιστεί και λόγω μιας αλληλεπίδρασης μεταξύ ενός κομματιού του υλικού (π.χ., ένα ρολόι) και του κόσμου, ή ενός γεγονότος πέρα από το δίκτυο.

Οι τύποι κόμβων που ακολουθούν ανήκουν στην κατηγορία των Αισθητήρων Περιβάλλοντος:

- **Collision (Αισθητήρας Σύγκρουσης)**

Ο κόμβος σύγκρουσης είναι ένας κόμβος ομαδοποίησης που διευκρινίζει τις ιδιότητες ανίχνευσης σύγκρουσης για τα παιδιά του (και τους απογόνους τους), διευκρινίζει τα αναπληρωματικά αντικείμενα που αντικαθιστούν τα παιδιά του κατά τη διάρκεια της ανίχνευσης σύγκρουσης, και στέλνει τα γεγονότα που επισημαίνουν ότι μια σύγκρουση μεταξύ των κόμβων έχει εμφανιστεί.

- **ProximitySensor (Αισθητήρας εγγύτητας)**

Ο κόμβος ProximitySensor παράγει τα γεγονότα όταν ο χρήστης μπαίνει, βγαίνει και κινείται μέσα σε μια συγκεκριμένη περιοχή στη σκήνη. Η περιοχή αυτή καθορίζεται από ένα κουτί.

- **VisibilitySensor (Αισθητήρας ορατότητας)**

Ο αισθητήρας ορατότητας χρησιμοποιείται χαρακτηριστικά για να ανιχνεύσει πότε ο χρήστης μπορεί να δει ένα συγκεκριμένο αντικείμενο ή μια περιοχή της σκηνής προκειμένου να ενεργοποιηθούν ή να απενεργοποιηθούν άλλα αντικείμενα. Σκοπός του συγκεκριμένου αισθητήρα είναι συχνά να προσελκύσει την προσοχή του χρήστη ή να βελτιώσει την απόδοση.

3.9.3 Key device sensors (Αισθητήρες Πληκτρολογίου)

Οι αισθητήρες πληκτρολογίου είναι κόμβοι που στέλνουν συμβάντα βασισμένα σε κάποιο συμβάν που δημιουργείται από το χρήστη χρησιμοποιώντας το πληκτρολόγιο.

Οι τύποι κόμβων που ακολουθούν ανήκουν στην κατηγορία Αισθητήρων Πληκτρολογίου:

- **Key Sensor (Αισθητήρας πλήκτρων)**

Οι αισθητήρες πλήκτρων ενεργοποιούνται και παράγουν γεγονότα με το πάτημα ενός πλήκτρου στο πληκτρολόγιο. Όταν το κουμπί στο πληκτρολόγιο απελευθερώνεται από το χρήστη, τότε ο αισθητήρας απενεργοποιείται. Η ταυτότητα που πλήκτρου που πατήθηκε, επιστρέφεται από το γεγονός που παράγεται.

- **String Sensor**

Ένας κόμβος αισθητήρων συμβολοσειράς παράγει τα γεγονότα καθώς ο χρήστης πιέζει πλήκτρα στο πληκτρολόγιο. Το τοπικό λειτουργικό σύστημα ορίζει μια τιμή, η οποία όταν εισάγεται από το χρήστη σηματοδοτείται η λήξη της πληκτρολόγησης. Όταν αναγνωρίζεται λήξη της πληκτρολόγησης από το λειτουργικό σύστημα, παράγεται ένα συμβάν με τιμή την συμβολοσειρά που εισήγαγε ο χρήστης.

3.9.3 Load sensors

3.9.4 Time Sensors (Χρονικοί αισθητήρες)

Οι κόμβοι TimeSensor είναι αισθητήρες που παράγουν τα γεγονότα με το πέρασμα του χρόνου.

Μπορούν να χρησιμοποιηθούν για πολλούς λόγους συμπεριλαμβανομένου:
1) Την δημιουργία μια συνεχούς προσομοίωσης ή κίνησης
2) Τον έλεγχο και τη δημιουργία περιοδικών δραστηριοτήτων σε ένα κόμβο (πχ περιστροφή ενός κόμβου, μια φορά το λεπτό)
3) Την έναρξη ενιαίων γεγονότων σε περίπτωση περιστατικού (πχ ένας συναγερμός)

```
TimeSensor : X3DTimeDependentNode, X3DSensorNode {
  SFTime  [in,out] cycleInterval    1     (0,∞)
  SFBool  [in,out] enabled          TRUE
  SFBool  [in,out] loop             FALSE
  SFNode  [in,out] metadata         NULL  [X3DMetadataObject]
  SFTime  [in,out] pauseTime        0     (-∞,∞)
  SFTime  [in,out] resumeTime       0
  SFTime  [in,out] startTime        0     (-∞,∞)
  SFTime  [in,out] stopTime         0     (-∞,∞)
  SFTime  [out]    cycleTime
  SFTime  [out]    elapsedTime
  SFFloat [out]    fraction_changed
  SFBool  [out]    isActive
  SFBool  [out]    isPaused
  SFTime  [out]    time
}
```

Ο κόμβος TimeSensor περιέχει δύο ιδιαίτερα outputOnly πεδία, το πεδίο isActive και το πεδίο cycleTime. Το πεδίο εξόδου isActive στέλνει την τιμή TRUE, όταν ο κόμβος TimeSensor ξεκινά να τρέχει, και FALSE όταν σταματά. Το πεδίο εξόδου cycleTime στέλνει χρονικό συμβάν στο πεδίο startTime και κάθε φορά που ένας καινούριος κύκλος ξεκινά. Τα υπόλοιπα πεδία εξόδου δημιουργούν συνεχή γεγονότα. Το πεδίο fraction_changed, ένας αριθμός SFFloat που μπορεί να πάρει τιμές από 0 έως και 1, στέλνει το ολοκληρωμένο μέρος του τρέχοντος κύκλου. Το πεδίο εισόδου εξόδου time στέλνει μια απόλυτη τιμή για κάθε στιγμιότυπο προσομοίωσης.

Αν το πεδίο enabled έχει την τιμή TRUE, ο κόμβος Timesensor ενεργοποιείται και μπορεί να τρέξει. Αν το πεδίο *set_enabled* λάβει την τιμή FALSE καθώς ο Timesensor τρέχει, ο αισθητήρας εκτελεί τις ακόλουθες ενέργειες:

1) Αξιολογεί και στέλνει όλα τα σχετικά αποτελέσματα
2) Στέλνει την τιμή FALSE στο πεδίο isActive
3) Απενεργοποιείται μόνος του

Τα γεγονότα εισαγωγής στα πεδία του Timesensor (e.g., *set_startTime)* υποβάλλονται σε επεξεργασία και τα αντίστοιχα πεδία εξόδου (e.g., startTime_changed) στέλνονται ανεξάρτητα από την κατάσταση του ενεργοποιημένου πεδίου.

Ο "κύκλος" ενός κόμβου TimeSensor διαρκεί για τα δευτερόλεπτα cycleInterval.

Το πεδίο εξόδου *cycleTime* μπορεί να χρησιμοποιηθεί για σκοπούς συγχρονισμού σε μια εφαρμογή, όπως για παράδειγμα ήχου με κίνηση. Η τιμή ενός γεγονότος *cycleTime* ισούται με την ώρα που έγινε η εκκίνηση του κύκλου. Ένα γεγονός cycleTime παράγεται

στην αρχή κάθε κύκλου, συμπεριλαμβανομένου του κύκλου που αρχίζει σε startTime. Το πρώτο γεγονός cycleTime για έναν κόμβο TimeSensor μπορεί να χρησιμοποιηθεί ως συναγερμός (ενιαίος σφυγμός σε έναν καθορισμένο χρόνο).

Όταν ένας κόμβος TimeSensor ενεργοποιείται παράγει συμβάν isActive = TRUE και αρχίζει να δημιουργεί συμβάντα χρόνου *fraction_changed,* και *cycleTime*, τα οποία μπορούν να καθοδηγηθούν σε άλλους κόμβους για να οδηγήσουν στη δημιουργία κίνησης.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
  "http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D profile="Immersive" >
 <head>
  <meta content="Simple X3D example" name="description"/>
 </head>
 <Scene>

  <Viewpoint description="The World" orientation="0 1 0 1.57" position="10 0 0"/>
  <Viewpoint description="Closer" orientation="0 1 0 1.57" position="6 0 0"/>
  <Viewpoint description="Near" orientation="0 1 0 1.57" position="2 0 0"/>
  <Viewpoint description="Heart of Darkness" orientation="0 1 0 1.57" position="0 0 0"/>
  <Transform rotation="0 1 0 1.57" DEF='TheWorldTurns' >
   <Shape>
    <Sphere/>
    <Appearance>
     <ImageTexture url="earth-topo.png"/>
    </Appearance>
   </Shape>
  </Transform>

<TimeSensor DEF='TimeThisTurn' cycleInterval='12.0' loop='true'
      pauseTime='0' isPaused='' resumeTime='0' fraction_changed=''/>                    (1)

<OrientationInterpolator DEF='TurnTheWorld' key='0.00 0.25 0.50 0.75 1.00'
      keyValue='0 1 0 0, 0 1 0 1.5708, 0 1 0 3.14159, 0 1 0 4.7123889, 0 1 0 6.2831852'/>    (2)

  <ROUTE fromNode='TimeThisTurn' fromField='fraction_changed' toNode='TurnTheWorld'
toField='set_fraction'/>                                                              (3)

  <ROUTE fromNode='TurnTheWorld' fromField='value_changed' toNode='TheWorldTurns'
toField='rotation'/>

 </Scene>
</X3D>
```

(1) Εδώ έχουμε τη δημιουργία ενός χρονικού αισθητήρα με διάρκεια κάθε κύκλου 12 δευτερόλεπτα cycleInterval='12.0'.

Interpolators: Χρησιμοποιούνται για να παρεμβάλουν στην κίνηση δηλαδή κίνηση στην οποία τα ιδιαίτερα χοντρά κομμάτια μιας μετακίνησης αποκαλούμενα "key values" (βασικές τιμές), υπολογίζονται κατά μέσο όρο για να δημιουργήσουν μια ενιαία και ομαλή κίνηση. Για παράδειγμα:

(2) Ένας OrientationInterpolator αλλάζει τον προσανατολισμό ενός αντικειμένου έτσι ώστε όταν στέλνονται οι τιμές του στο πεδίο περιστροφής (orientation) ενός αντικειμένου, το αντικείμενο να περιστρέφεται.

(3) Οι τιμές από το πεδίο 'fraction_changed' του κόμβου Time Sensor στέλνονται στον κόμβο orientation interpolator και στο πεδίο 'set_fraction' . Στη συνέχεια στέλνεται η τιμή του πεδίου 'value_changed' από τον κόμβο orientation interpolator στο πεδίο 'rotation' του κόμβου 'TheWorldTurns'

4. Scene Authoring Interface

Το **Scene Authoring Interface (SAI)** προδιαγράφει αλληλεπιδράσεις με το γράφο σκηνής στη περίπτωση όπου αυτή η αλληλεπίδραση γίνεται μέσω εξωτερικών εφαρμογών. Μέσω αυτών των προδιαγραφών, δίνεται ανεξαρτήτου γλώσσας το σύνολο των ενεργειών που μπορούν να πραγματοποιηθούν από μια εξωτερική εφαρμογή διαμέσου της συγκεκριμένης διεπαφής (SAI).

Το SAI δημιουργεί μια κοινή διεπαφή που μπορεί να χρησιμοποιηθεί είτε για διαχείριση του browser και του γράφου σκηνής από μια εξωτερική εφαρμογή ή μέσα από το γράφο σκηνής με τη χρήση συγκεκριμένου προγραμματιστικού κόμβου, του script node. Οι δύο αυτοί τρόποι διαχείρισης διαφέρουν στη δομή του αντίστοιχου κώδικά τους, μη επιτρέποντας αυτόματη εκτέλεση τμημάτων κώδικα από εξωτερική εφαρμογή σε script node του γράφου και ανάποδα. Μέσω του προτύπου παρέχεται ένα μοναδικό, ενοποιημένο προγραμματιστικό περιβάλλον διεπαφής καθώς και περιορισμοί που εξαρτώνται από το περιβάλλον υλοποίησης του κώδικα.

Οι προδιαγραφές του SAI επιτρέπουν πέντε διαφορετικούς τύπους πρόσβασης στη σκηνή X3D:

- Πρόσβαση στη λειτουργικότητα του browser
- Λήψη ειδοποιήσεων ενεργειών του browser, όπως λάθος URL, εκκίνηση και τερματισμός.
- Αποστολή γεγονότων σε κατάλληλα πεδία κόμβων που δέχονται είσοδο μέσα στο γράφο σκηνής
- Διερμηνείας της γραφικής παράσταση σκηνής
- Δημιουργίας και να καταστροφής κόμβων
- Αποστολής γεγονότων στους κόμβους
- Δημιουργίας συνδέσεων μεταξύ των κόμβων (διαδρομές)
- Ανάγνωση τελευταίας τιμής που απεστάλη από πεδία εξόδου κόμβων της σκηνής
- Ειδοποίηση σχετικά με γεγονότα που αλλάζουν τιμές πεδίων κόμβων στη σκηνή

Το X3D παρέχει ένα σύνολο διεπαφών προγραμματιστών εφαρμογής (APIs), αποκαλούμενα διεπαφή πρόσβασης σκηνής (SAI), η οποία καθορίζει την πρόσβαση στη σκηνή κατά την εκτέλεση.

4.1 Βασικές λειτουργίες SAI στη Java

❖ **Φόρτωση ενός X3D κόσμου**

Για να προσθέσουμε μιας X3D σκηνής στην εφαρμογή μας πρέπει να ακολουθηθούν οι παρακάτω διαδικασίες:

❶ Προσθήκη του τρισδιάστατου παραθύρου

Αρχικά πρέπει να δημιουργηθεί ένα X3D component με τη χρήση της κλάσης SAI **BrowserFactory.** Το νέο αυτό component στη συνέχεια προστείθεται στην εφαρμογή. Ακολουθεί ο κώδικας:

```
import org.web3d.x3d.sai.*;

public class SimpleSAIDemo extends JFrame {

    public SimpleSAIDemo() {
        Container contentPane = getContentPane();

        // Δημιουργία του SAI component
        X3DComponent x3dComp = BrowserFactory.createX3DComponent(null);

        // Προσθήκη του παραπάνω component στην εφαρμογή
        JComponent x3dPanel = (JComponent)x3dComp.getImplementation();
        contentPane.add(x3dPanel, BorderLayout.CENTER);
```

Προσθήκη **X3D** component Java με τη χρήση SAI

❷ Φόρτωση του X3D αρχείου

Προκειμένου να φορτωθεί ένα X3D αρχείο στην εφαρμογή μας, απαραίτητη είναι η εισαγωγή του X3D Browser. Ο Browser είναι η βασική διεπαφή που χρειαζόμαστε για την αλληλεπίδραση με την εφαρμογή. Επιτρέπει ενέργειες, όπως τη φόρτωση των αρχείων, τη δημιουργία των νέων κόμβων, μπορεί να μας δώσει πληροφορίες για το ρυθμό των frames του x3d. Η κλάση Browser είναι διαθέσιμη και στα εσωτερικά και εξωτερικά προγράμματα SAI.

Η διεπαφή ExternalBrowser επιτρέπει σε εξωτερικά προγράμματα να έχουν πρόσβαση σε περισσότερες λειτουργίες όπως την παύση του rendering και την προσθήκη Browser

listeners. Αυτές οι διαδικασίες μπορούν να γίνουν μόνο με τη χρήση εξωτερικού κώδικα. Προκειμένου να φορτωθεί ένα αρχείο πρέπει να εκτελεστεί η παρακάτω εντολή:

```
// Με αυτήν την εντολή προσθέτουμε τον εξωτερικό Browser
ExternalBrowser x3dBrowser = x3dComp.getBrowser();
```

Για τη φόρτωση του x3d αρχείου στον Browser, είναι απαραίτητη και η παραπάνω αναφορά του εξωτερικού Browser. Για να γίνει αυτό, εκτός από την παραπάνω αναφορά, χρησιμοποιείται και η κλήση της μεθόδου createX3DFromURL η οποία δεν επιστρέφει τίποτα έως ότου μεταφορτωθεί το αρχείο και είναι έτοιμο για επίδειξη.

```
//Δημιουργία μιας X3D σκηνής με τη φόρτωση ενός αρχείου
X3DScene mainScene = x3dBrowser.createX3DFromURL(new String[]
{"moving_box.x3dv" });

//Αντικατάσταση του τρέχοντος κόσμου με νέο
x3dBrowser.replaceWorld(mainScene);
```

❖ Δημιουργία νέου κόμβου

Το SAI εκτός από την δυνατότητα φόρτωσης αρχείων, επιτρέπει και τη δημιουργίας της σκηνής από την αρχή. Το πρώτο στάδιο της διαδικασίας για τη δυναμική δημιουργία ενός κόσμου είναι η δημιουργία ενός X3DScene. Κάθε X3DScene πρέπει να προ-δηλώνει ποιο Profile και ποια Components θα χρησιμοποιήσει. Αν το ένα Profile που αιτείται, δεν υποστηρίζεται από το browser τότε θα εμφανιστεί το Exception : NotSupportedException.

```
ProfileInfo profile = null;

try {
    profile = x3dBrowser.getProfile("Immersive");
    } catch(NotSupportedException nse) {
            System.out.println("Immersive Profile not supported");
            System.exit(-1);
        }
```

Δημιουργία X3D σκηνής – Στάδιο Δημιουργίας Profile

Μετά το στάδιο της δημιουργίας του Profile της σκηνής, ακολουθέι το στάδιο της δημιουργίας της. Η μέθοδος **createScene** δημιουργεί μια κενή σκηνή και επιστρέφει ένα αντικείμενο **X3DScene.**

```
X3DScene mainScene = x3dBrowser.createScene(profile, null);
```

Δημιουργία X3D σκηνής – Στάδιο Δημιουργίας Σκηνής

Μετά το στάδιο δημιουργίας της κενής σκηνής, σειρά έχει η δημιουργία κόμβων και η προσθήκη τους στην κενή σκηνή που μόλις δημιουργήθηκε. Η μέθοδος **createNode** χρησιμοποιείται για τη δημιουργία των κόμβων. Αν ο κόμβος είναι άγνωστος θα τότε θα εμφανιστεί το InvalidNodeException. Η κλάση createX3DFromURLTo παρακάτω παράδειγμα απεικονίζει τη διαδικασία:

```
X3DNode shape = mainScene.createNode("Shape");                (1)
SFNode shape_geometry = (SFNode) (shape.getField("geometry"));
X3DNode box = mainScene.createNode("Box");                    (2)

shape_geometry.setValue(box);                                 (3)
mainScene.addRootNode(shape);                                 (4)
```

Δημιουργία X3D σκηνής – Στάδιο Δημιουργίας κόμβων

(1) Στο παραπάνω παράδειγμα παρουσιάζει τη δημιουργία ενός κόμβου Shape.
(2) Στη συνέχεια, δημιουργείται ένας κόμβος Box και
(3) Τοποθετείται σαν τιμή στο γεωμετρικό πεδίο του κόμβου Shape.
(4) Το μόνο που απομένει είναι να προστεθεί ο κόμβος αυτός στο Root της δημιουργημένης σκηνής.

❖ "Άκουσμα" των αλλαγών στις τιμές των πεδίων

Ένα πρόγραμμα SAI μπορεί να ακούει τις αλλαγές στις τιμές σε κάθε πεδίο. Αυτό επιτρέπει στην εφαρμογή να πραγματοποιεί αλλαγές, με συνεχή εμφάνιση των αλλαγών αυτών στην τρισδιάστατη προσομοίωση.

```
#X3D V3.0 utf8

PROFILE Interactive

DEF TG Transform {
  rotation 0 1 0 0.78
  children [
     Shape {
    geometry Box {}
    appearance Appearance {
      material Material {
        diffuseColor 1 0 0
      }
    }
  }
  DEF TOUCH_SENSOR TouchSensor {}  ]
}
```

X3D αρχείο με TouchSensor

Το παραπάνω X3D αρχείο έχει ένα κόμβο TouchSensor. Με τον παρακάτω κώδικα SAI γίνεται ανάγνωση των αλλαγών για το πεδίο touchTime του κόμβου TouchSensor. θα χρησιμοποιήσει τη μέθοδο addX3DEventListener για να αφουγκραστεί τις αλλαγές στον πεδίο touchTime του κόμβου TouchSensor. Θα χρειαστεί επίσης και η χρήση του **X3DFieldEventListener** Interface.

```
// Δημιουργία μιας X3D σκηνής με τη φόρτωση ενός αρχείου
X3DScene mainScene = x3dBrowser.createX3DFromURL(new String[] { "box.x3d" });

// Αντικατάσταση του παρόντος κόσμου με καινούριο
x3dBrowser.replaceWorld(mainScene);

// Εύρεση του κόμβου TouchSensor με όνομα TOUCH_SENSOR
X3DNode touch = mainScene.getNamedNode("TOUCH_SENSOR");
if (touch == null)
{
  System.out.println("Couldn't find TouchSensor named: TOUCH_SENSOR");
  return;
}

SFTime ttime = (SFTime) touch.getField("touchTime");

// Προσθήκη Listener για την ανίχνευση των αλλαγών στο πεδίο touchTime
ttime.addX3DEventListener(this);
}

public void readableFieldChanged(X3DFieldEvent evt) {
        System.out.println("Stop touching me!");
}
```

Κώδικας SAI – Ανίχνευση συμβάντων στον κόμβο TouchSensor

Με τον κώδικα SAI γίνεται αναζήτηση του κόμβου TouchSensor (στο αρχείο box.x3d) με όνομα «TOUCH_SENSOR», στη συνέχεια γίνεται προσθήκη ενός Listener για να υπάρχει η δυνατότητα καταγραφής των γεγονότων που συμβαίνουν στο πεδίο touchTime του κόμβου. Στην περίπτωση ανίχνευσης κάποιου γεγονότος ένα μήνυμα «Stop touching me!» Εμφανίζεται στο χρήστη.

5. Ανάλυση Εφαρμογής

Περιγραφή Εφαρμογής

Θέμα
Το θέμα της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη μιας εφαρμογής απεικόνισης τρισδιάστατης X3D σκηνής. Η εφαρμογή θα δίνει στο χρήστη τη δυνατότητα αλληλεπίδρασης με την τρισδιάστατη σκηνή που έχει προφορτώσει ο χρήστης. Η παρούσα εφαρμογή θα παρέχει επίσης τη δυνατότητα (μέσω δικτύου) σε έναν απομακρυσμένο χρήστη να μπορεί να παρακολουθήσει ζωντανά τις κινήσεις του χρήστη, καθώς και τη δυνατότητα αλληλεπίδρασης του απομακρυσμένου χρήστη με τη σκηνή.

5.1 Βασικός αλγόριθμος υλοποίησης εργασίας

Η βασική ιδέα για τη λύση του προβλήματος και την επίτευξη της ζωντανής παρακολούθησης μιας κίνησης στην x3d σκηνή ενός χρήστη από έναν άλλον απομακρυσμένο χρήστη ήταν να παρακολουθήσουμε τις αλλαγές στη σκηνή του χρήστη που δημιουργεί την κίνηση και στη συνέχεια να τις αποστείλουμε στον απομακρυσμένο χρήστη. Εκείνος με τη σειρά του θα αξιοποιήσει τις λαμβανόμενες πληροφορίες και θα τις προσθέσει στη δική του σκηνή. Με αυτόν τον τρόπο ο απομακρυσμένος χρήστης παρακολουθεί ζωντανά τις κινήσεις του χρήστη που δημιούργησε το συμβάν.

Ακολουθούν τα βήματα από τα οποία αποτελείται ο αλγόριθμος:

ΔΗΜΙΟΥΡΓΙΑ ΠΕΡΙΒΑΛΛΟΝΤΟΣ X3D
Το πρώτο βήμα για την ανάπτυξη της εφαρμογής, είναι η εισαγωγή στην εφαρμογή ενός browser που θα χρησιμοποιηθεί για την απεικόνιση της x3d σκηνής που θα εισάγει ο χρήστης. Για την συγκεκριμένη εφαρμογή χρησιμοποιήθηκε ο Xj3D Browser. Με την εισαγωγή του browser, οι χρήστες είναι ικανοί να φορτώσουν και να αλληλεπιδράσουν με X3D σκηνές.

> Τα γεγονότα που παράγονται σε έναν κόμβο από την αλληλεπίδραση του χρήστη με τη σκηνή προέρχονται αποκλειστικά και μόνο από αισθητήρες καθώς είναι οι μόνοι που μπορούν να επιτρέψουν στο χρήστη να αλλάξει τα δεδομένα στους κόμβους από τους οποίους αποτελείται η σκηνή. Οι αισθητήρες που μελετώνται στην παρούσα πτυχιακή είναι οι εξής: **CylinderSensor, PlaneSensor, SphereSensor, TouchSesnor** και θα αναλυθούν λεπτομερώς παρακάτω.

ΠΡΟΣΘΗΚΗ LISTENERS ΣΤΟΥΣ ΑΙΣΘΗΤΗΡΕΣ ΟΛΩΝ ΤΩΝ ΤΥΠΩΝ
Για να είναι δυνατή η ανίχνευση των γεγονότων που παράγει ο χρήστης με τη χρήση κάποιου από τους αισθητήρες απαραίτητη ήταν η προσθήκη ενός **Listener** σε κάθε έναν από τους αισθητήρες, ώστε όταν ο χρήστης αλληλεπιδράσει με κάποιον από τους

αισθητήρες, μπορούμε να πάρουμε πληροφορίες που χρειαζόμαστε, όπως ποιος αισθητήρας ενεργοποιήθηκε και σε ποιον κόμβο είναι συνδεδεμένος ο αισθητήρας που ενεργοποίησε ο χρήστης.

ΑΝΙΧΝΕΥΣΗ ΚΑΙ ΑΝΑΛΥΣΗ ΓΕΓΟΝΟΤΟΣ – ΕΥΡΕΣΗ ΝΕΩΝ ΣΥΝΤΕΤΑΓΜΕΝΩΝ

Με την ανίχνευση κάποιου γεγονότος από τον browser, η εφαρμογή αναλύει το γεγονός κάνοντας τα ακόλουθα. Ανιχνεύει ποιος αισθητήρας προκάλεσε το γεγονός και μέσω αυτού του αισθητήρα, καταλήγει στον κόμβο στον οποίο προκλήθηκε η αλλαγή στα δεδομένα. Γνωρίζοντας τον κόμβο στον οποίο προκλήθηκαν οι αλλαγές, βρίσκει τις νέες συντεταγμένες που απέκτησε ο κόμβος.

ΑΠΟΣΤΟΛΗ ΤΩΝ ΣΥΝΤΕΤΑΓΜΕΝΩΝ ΣΤΟΝ ΑΠΟΜΑΚΡΥΣΜΕΝΟ ΧΡΗΣΤΗ ΜΕ ΧΡΗΣΗ XML

Γνωρίζοντας πια τον αισθητήρα που ενεργοποίησε ο χρήστης και τις νέες συντεταγμένες που απέκτησε ο κόμβος ,έχουμε όλες τις πληροφορίες που χρειαζόμαστε για να περιγράψουμε τη συνέβη στη σκηνή μας. Το επόμενο στάδιο του αλγορίθμου, είναι η αποστολή των προαναφερθέντων πληροφοριών στην X3D σκηνή του απομακρυσμένου χρήστη. Για την αποστολή των δεδομένων χρησιμοποιούνται xml αρχεία τα οποία πεδία των οποίων παίρνουν σαν τιμές το όνομα του αισθητήρα, και τις νέες συντεταγμένες του κόμβου με τον οποίο είναι συνδεδεμένος ο αισθητήρας.

Παραπάνω αναλύθηκε η διαδικασία με την οποία ανιχνεύεται ένα γεγονός της σκηνής και μέσω του γεγονότος την εύρεση των νέων συντεταγμένων. Στη συνέχεια αναλύεται ο αλγόριθμος κατά τον οποίο αξιοποιούνται οι πληροφορίες που έλαβε ο απομακρυσμένος χρήστης, ώστε να μπορέσει τελικά να παρακολουθήσει τις κινήσεις του χρήστη που δημιούργησε τα γεγονότα.

ΛΗΨΗ XML ΑΡΧΕΙΟΥ ΑΠΟ ΤΟΝ ΑΠΟΜΑΚΡΥΣΜΕΝΟ ΧΡΗΣΤΗ

- Με την λήψη του xml αρχείου από τον απομακρυσμένο χρήστη, ο τελευταίος ξεκινά την αποκωδικοποίηση του xml αρχείου. Στην αποκωδικοποίηση ο χρήστης κρατάει το όνομα του αισθητήρα το οποίο προκάλεσε το γεγονός και τις συντεταγμένες του κόμβου με τον οποίο συνδέεται ο αισθητήρας.

- Γνωρίζοντας το όνομα του αισθητήρα, πραγματοποιείται αναζήτηση σε όλους τους κόμβους της σκηνής με σκοπό την εύρεση του κόμβου που έχει σαν όνομα, το όνομα του αισθητήρα που αποκωδικοποιήθηκε από το xml αρχείο.

- Με την εύρεση του ομώνυμου κόμβου, και με τη χρήση του ιδεατού αγωγού μεταφοράς συμβάντων (route) με τον οποίο είναι συνδεδεμένος, καταλήγουμε στον κόμβο προορισμού του συμβάντος (destination node).

- Στη συνέχεια, αποδίδονται οι τιμές των νέων συντεταγμένων που αποκωδικοποιήθηκαν από το xml αρχείο στον κόμβο προορισμού του συμβάντος.

5.2 Επεξήγηση του βασικού αλγορίθμου – Κώδικας υλοποίησης

ΔΗΜΙΟΥΡΓΙΑ ΠΕΡΙΒΑΛΛΟΝΤΟΣ X3D

- Ανοίγοντας την εφαρμογή ο χρήστης καλείται να επιλέξει ένα x3d αρχείο από μια λίστα ενός φακέλου το αρχείο που επιθυμεί να φορτώσει.

```
/* Κώδικας στο πρόγραμμα*/
fileToOpen = choosefile();
```

Το πρόγραμμα καλεί τη συνάρτηση **choosefile** η οποία διαβάζοντας τα x3d αρχεία που περιέχονται στην τοποθεσία του δίσκου stringUrl, επιστρέφει το όνομα του αρχείου που επέλεξε ο χρήστης.

```
public String choosefile()
   {
      int counter=0;
      JOptionPane chooseFiles = (JOptionPane) new JOptionPane();

     //  Variable stringUrl contains the url in the disk that contains the x3d files
      File dir = new File(stringUrl);
      File[] files = dir.listFiles();

      String fileNames []= new String[files.length];

      for (int p=0; p<files.length; p++)
      {
         String temp= files[p].getName();
         temp = temp.trim();

         //... Find the position of the last dot.  Get extension.
         int dotPos = temp.lastIndexOf(".");

         if (dotPos!=-1)
         {
            String extension = temp.substring(dotPos);
             if (extension.equals(".x3d"))
            {
               fileNames[counter]=files[p].getName();
               counter++;
            }
         }
      }

      Object selectedX3dFile = chooseFiles.showInputDialog(null,"Choose the X3D File you want
to load...","Choose file",JOptionPane.INFORMATION_MESSAGE,null,fileNames,fileNames);

      return selectedX3dFile.toString();
   }
```

Για να πραγματοποιηθεί η φόρτωση του αρχείου που επιλέχθηκε, η εφαρμογή καλεί έναν εξωτερικό browser x3d. Δημιουργείται ένα συστατικό X3D και στη συνέχεια καλείται ο X3DBrowser.

```
x3dComp=BrowserFactory.createX3DComponent(requestedParameters);
x3dBrowser = x3dComp.getBrowser();
```

Στη συνέχεια δημιουργείται η σκηνή μας φορτώνοντας το αρχείο που έχει επιλέξει ο χρήστης. Το αρχείο φωρτώνεται καλώντας τη συνάρτηση createX3DFromURL η οποία παίρνει σαν όρισμα την απόλυτο url του αρχείου σε μορφή συμβολοσειράς (String).

```
mainScene=x3dBrowser.createX3DFromURL(new String[] {"file:///"+stringUrl+"//"+fileToOpen});
x3dBrowser.replaceWorld(mainScene);
```

Εκτός από την εισαγωγή του browser και τη φόρτωση του αρχείου στη σκηνή δημιουργούνται και τα sockets που θα χρησιμοποιηθούν για την αποστολή και τη λήψη νέων γεγονότων από τον απομακρυσμένο χρήστη, καθώς και το βασικό Interface της εφαρμογής.

```
try
    {
    //Create the socket that will be used to receive XML files
    socket = new DatagramSocket(portR);
    sizeSocket = new DatagramSocket(portRsize);
    address = InetAddress.getByName(IPAddress);
    }
catch(Exception cdf) { cdf.printStackTrace(); }
}
```

Για την εφαρμογή χρησιμοποιήθηκαν 2 sockets, ο λόγος θα αναλυθεί παρακάτω.

```
public void interfacex3d()
{

// Setup browser parameters
HashMap requestedParameters = new HashMap();

// Create a SAI component
x3dComp=BrowserFactory.createX3DComponent(requestedParameters);

Container pane = getContentPane();                                    //Get the Content Pane

JPanel panel=(JPanel) new JPanel(new FlowLayout());

JPanel panelbutton=(JPanel) new JPanel(new BorderLayout());          //Panel For button connect
JComponent x3dPanel= (JComponent)x3dComp.getImplementation();      // Add the  component
                                                                          to the UI
JLabel label= (JLabel) new JLabel();

buttonConnect= new JButton("Connect");
try {
```

```
//Mask Formatter for the IP Adress
 MaskFormatter formatter = new MaskFormatter("###'.###'.###'.###");
 final JFormattedTextField IPinput = new JFormattedTextField(formatter);
 formatter.setInvalidCharacters("*");
 formatter.setPlaceholderCharacter('0');

 try {
        //Create the socket for receiving the xml files
        socket = new DatagramSocket(portR);

        //Create the socket for receiving the size of the xml files
        sizeSocket = new DatagramSocket(portRsize);

        //Connect with the IpAddress that the user has entered
        address = InetAddress.getByName(IPAddress);
      }catch(Exception cdf){cdf.printStackTrace();}

//Create actionListener for connection to the remote user
 buttonConnect.addActionListener(new ActionListener()
{
       public void actionPerformed(ActionEvent actionEvent)
       {
         //Get the IP Adress that the user inserted
          String IPAddress = IPinput.getText();

          int validIP=1;

          //if the IP Address the user entered is valid then validIP=1, else validIP=0
          boolean ableToConnect = true;
          for (int IpPosition=0; IpPosition<=15; IpPosition=IpPosition+4 )
          {
               validIP = checkValidIp(IPAddress,IpPosition,IpPosition+3);
               if (validIP==0)
                 ableToConnect=false;
          }

         try {
            //Create the socket that will be used to receive XML files
             socket = new DatagramSocket(portR);
             sizeSocket = new DatagramSocket(portRsize);
             address = InetAddress.getByName(IPAddress);
             }catch(Exception cdf){cdf.printStackTrace();}
       }
    });


IPinput.setColumns(9);
label.setText("Insert IP Address");

panel.add(label);
panel.add(IPinput);
panel.add(panelbutton, BorderLayout.EAST);
panel.add(panelbutton, BorderFactory.createCompoundBorder());
```

```
        panelbutton.add(buttonConnect, BorderLayout.CENTER);
        statusLabel = (JLabel) new JLabel();
        panelbutton.add(statusLabel, BorderLayout.PAGE_END);

        pane.add(x3dPanel, BorderLayout.CENTER);
        pane.add(panel, BorderLayout.PAGE_START);

        } catch(Exception cdf){cdf.printStackTrace();}

        // Get the X3D external browser
        x3dBrowser = x3dComp.getBrowser();

       // The chooseFile function returns the name of the file that the user has selected to open In
       String format
       fileToOpen = choosefile();

       // Create an X3D scene by loading the demo file (fileTo open)
       mainScene=x3dBrowser.createX3DFromURL(new String[] {"file:///"+stringUrl+"//"+fileToOpen});

       // Replace the current world with the new one
       x3dBrowser.replaceWorld(mainScene);

}
```

ΠΡΟΣΘΗΚΗ LISTENERS ΣΤΟΥΣ ΑΙΣΘΗΤΗΡΕΣ ΟΛΩΝ ΤΩΝ ΤΥΠΩΝ

- **Εύρεση όλων των αισθητήρων της σκηνής**

Πρωταρχικός ρόλο στην παρούσα πτυχιακή παίζουν οι αισθητήρες που παρέχει το X3d, αφού αυτοί είναι υπεύθυνοι για τη δημιουργία αλληλεπίδρασης της σκηνής με το χρήστη. Για να είναι εφικτή η ανίχνευση των γεγονότων που παράγονται από το χρήστη αρχικά πρέπει να βρεθούν όλοι οι αισθητήρες που περιέχει μια οποιαδήποτε σκηνή που είναι πιθανόν να φορτώσει ο χρήστης και στη συνέχεια σε κάθε αισθητήρα, προστεθεί ένας Listener.

Μια σκηνή είναι δυνατόν να περιέχει τέσσερα από τα βασικά είδη αισθητήρων. Βασικά είδη θεωρούνται οι εξής αισθητήρες : TouchSensor, PlaneSensor, SphereSensor και CylinderSensor.

Σε όλους τους αισθητήρες που περιέχονται σε μια σκηνή αναγκάστηκα έχει οριστεί ένα όνομα για τον απλούστατο λόγο ότι κάθε αισθητήρας που συνδέεται με έναν κόμβο χρησιμοποιεί απαραιτήτως έναν ιδεατό αγωγό μεταφοράς συμβάντων (route) ώστε να μπορέσει να γίνει η σύνδεση. Ο ιδεατός αγωγός (route) με τη σειρά του για να συνδέσει έναν αισθητήρα με έναν κόμβο πρέπει να γνωρίζει ποιόν ακριβώς αισθητήρα συνδέει με ποιον κόμβο, και αφού το όνομα ενός κόμβου είναι αυτό που καθορίζει μοναδικά τον κόμβο στη σκηνή, ο ιδεατός αγωγός μεταφοράς συμβάντων χρησιμοποιεί τα ονόματα του αισθητήρα και του κόμβου για τη δημιουργία επιτυχούς σύνδεσης.

Αρχικά πρέπει ψάξουμε σε όλους τους κόμβους της σκηνής και να βρούμε ποιοι από αυτούς τους κόμβους ανήκουν στη κατηγορία των αισθητήρων και σε ποιο ακριβώς είδος αισθητήρα. Το SAI που παρέχεται από το X3D δίνει τη δυνατότητα εύρεσης όλων των κόμβων μιας σκηνής για τους οποίους έχει οριστεί ένα όνομα.

Για να βρεθούν λοιπόν όλοι οι κόμβοι που ανήκουν σε μια από τις κατηγορίες των αισθητήρων που αναφέρθηκαν παραπάνω, δημιουργήθηκε ένας πίνακας που περιέχει όλους τους ονοματισμένους κόμβους της σκηνής και στη συνέχεια για κάθε στοιχείο του πίνακα, έγινε αναζήτηση για τον αν ανήκει σε έναν από τους τέσσερις τύπους αισθητήρων που μελετάμε (TouchSensor, PlaneSensor, SphereSensor, CylinderSensor).

```
// Η παρακάτω συνάρτηση τοποθετεί στον πίνακα named Nodes όλους τους ονοματισμένους κόμβους
namedNodes = mainScene.getNamedNodes();
```

Διαβάζοντας σειριακά τον πίνακα και εκτελώντας για κάθε στοιχείο την παρακάτω εντολή βρίσκουμε τα είδη των κόμβων που περιέχονται μέσα στο x3d αρχείο που φόρτωσε ο χρήστης.

```
// Η παρακάτω εντολή επιστρέφει στο String ftype το είδος του κόμβου που περιέχεται σε κάθε θέση του πίνακα nn. Ο κόμβος nn όπως φαίνεται και στον κώδικα παρακάτω είναι ένας προσωρινός κόμβος που παίρνει κάθε φορά την τιμή ενός από τα στοιχεία του πίνακα namedNodes
ftype = mainScene.getNamedNode(nn).getNodeName();
```

Στην περίπτωση που το στοιχείο του πίνακα namedNodes που 'διαβάζεται' είναι κόμβος τύπου TouchSensor η τιμή του ftype θα είναι "TouchSensor". Με αυτόν τον τρόπο είμαστε σε θέση να γνωρίζουμε τον τύπο κάθε κόμβου του αρχείου x3d. Αυτό που μας ενδιαφέρει προς το παρόν είναι οι αισθητήρες. Οπότε όπως φαίνεται και παρακάτω στο πρόγραμμα κάνουμε τον έλεγχο μόνο για τους αισθητήρες.

- **Ανίχνευση των γεγονότων που δημιούργησε ο χρήστης**

Για να μπορούν να ανιχνευθούν γεγονότα από τους αισθητήρες χρειάζονται ειδικοί x3dEventListeners. Ένας x3dEventListeners είναι ένας listener ακούει τα γεγονότα στα πεδία x3d κόμβων. Οι listeners αυτοί τοποθετούνται σε όλα τα είδη των αισθητήρων που περιέχει η σκηνή. Κάθε τύπος αισθητήρα χρειάζεται τον δικό του μοναδικό listener γιατί κάθε αισθητήρας προκαλεί διαφορετικό είδος κίνησης στον κόμβο με τον οποίο συνδέεται.

1. Αισθητήρας PlaneSensor

Ο x3dEventListener που θα προστεθεί σε όλους τους κόμβους τύπου *PlaneSensor*, πρέπει να 'συγχρονισθεί' ώστε να ακούει τα γεγονότα που παράγονται από το πεδίο *translation_changed* του αισθητήρα.

Για να γίνει αυτό, το πρόγραμμα ψάχνει όλους τους κόμβους - αισθητήρες τύπου *PlaneSensor* της σκηνής και για καθέναν από αυτούς διαβάζει την τιμή του πεδίου *translation_changed* και την καταχωρεί σε ένα τρισδιάστατο διάνυσμα (SFVec3f), αποτελούμενο από τρεις single precision αριθμούς κινητής υποδιαστολής (*planeTranslation*).

```
SFVec3f planeTranslation = (SFVec3f) plnSensor.getField("translation_changed");
```

Με την παρακάτω εντολή αντιστοιχίζεται κάθε γεγονός που πρόκειται να παραχθεί από το συγκεκριμένο πεδίο του κόμβου που εξετάζεται με το ίδιο το DEF όνομα του κόμβου - αισθητήρα. Αυτό που προκύπτει από την παραπάνω διαδικασία είναι να δώσουμε μια ταυτότητα στο γεγονός που πρόκειται να παράγει ο αισθητήρας. Σε μετέπειτα στάδιο του προγράμματος θα έχουμε τη δυνατότητα με την ανίχνευση γεγονότος που προκάλεσε ένας αισθητήρας του τύπου PlaneSensor, να βρούμε ποιος ακριβώς από όλους τους αισθητήρες του τύπου αυτού το προκάλεσε.

```
planeTranslation.setUserData(planeSensorList[k]);
```

```
if (ftype=="PlaneSensor")
{
    planeSensorList[k]=nn;

   // Find the Plane Sensor Node
    plnSensor=mainScene.getNamedNode(planeSensorList[k]);
    SFVec3f planeTranslation = (SFVec3f) plnSensor.getField("translation_changed");

  // Accosiate user data with the translation_changed field
    planeTranslation.setUserData(planeSensorList[k]);

    // Add listener to plane Sensor
    planeTranslation.addX3DEventListener(this);
    k++;
}
```

Το πρόγραμμα εκτελεί την παραπάνω διαδικασία και για τους υπόλοιπους τύπους αισθητήρων.

2. Αισθητήρας TouchSensor

Ο x3dEventListener που θα προστεθεί σε όλους τους κόμβους τύπου *TouchSensor*, πρέπει να 'συγχρονισθεί' ώστε να ακούει τα γεγονότα που παράγονται από το πεδίο *touchTime* του αισθητήρα.

Και εδώ όπως και προηγουμένως για όλους τους αισθητήρες τύπου TouchSensor διαβάζεται η τιμή του πεδίου '*touchTime*' και καταχωρείται σε μια μεταβλητή τύπου SFTime.

```
SFTime touchTranslation = (SFTime) tchSensor.getField("touchTime");
```

Ενώ στη συνέχεια δίνουμε ένα μοναδικό όνομα - ταυτότητα στο γεγονός που πρόκειται να παράγει ο κάθε αισθητήρας, το οποίο δεν είναι άλλο από το όνομα του ίδιου του αισθητήρα.

```
touchTranslation.setUserData(touchSensorList[h]);
```

Με την παρακάτω εντολή το πρόγραμμα προσθέτει τον Listener στον αισθητήρα.

```
touchTranslation.addX3DEventListener(this);
```

```
if (ftype=="TouchSensor")
{

    touchSensorList[h]=nn;

    //Find the Touch Sensor
    tchSensor=mainScene.getNamedNode(touchSensorList[h]);
    SFTime touchTranslation = (SFTime) tchSensor.getField("touchTime");

    //Add listener to the sensor
    touchTranslation.addX3DEventListener(this);
    touchTranslation.setUserData(touchSensorList[h]);
    h++;
}
```

3. Αισθητήρας SphereSensor

Ο x3dEventListener που θα προστεθεί σε όλους τους κόμβους τύπου *SphereSensor*, πρέπει να 'συγχρονισθεί' ώστε να ακούει τα γεγονότα που παράγονται από το πεδίο *rotation_changed* του αισθητήρα.

Και εδώ όπως και προηγουμένως για όλους τους αισθητήρες τύπου *SphereSensor* διαβάζεται η τιμή του πεδίου ' *touchTime* ' και καταχωρείται σε μια μεταβλητή τύπου SFTime.

```
SFRotation sphereTranslation = (SFRotation)phereSensor.getField("rotation_changed");
```

Ενώ στη συνέχεια δίνουμε ένα μοναδικό όνομα - ταυτότητα στο γεγονός που πρόκειται να παράγει ο κάθε αισθητήρας, το οποίο δεν είναι άλλο από το όνομα του ίδιου του αισθητήρα.

```
sphereTranslation.setUserData(sphereSensorList[o]);
```

Με την παρακάτω εντολή το πρόγραμμα προσθέτει τον Listener στον αισθητήρα.

```
sphereTranslation.addX3DEventListener(this);
```

```
if (ftype=="SphereSensor")
{
    sphereSensorList[o]=nn;

    //Find the Sphere Sensor
    sphereSensor=mainScene.getNamedNode(sphereSensorList[o]);
    SFRotation sphereTranslation = (SFRotation)phereSensor.getField("rotation_changed");
    X3DFieldDefinition [] m=sphereSensor.getFieldDefinitions();
    sphereTranslation.setUserData(sphereSensorList[o]);

    //add listener to sphereSensor
    sphereTranslation.addX3DEventListener(this);
    o++;
}
```

4. Αισθητήρας CylinderSensor

Ο x3dEventListener που θα προστεθεί σε όλους τους κόμβους τύπου *SphereSensor*, πρέπει να 'συγχρονισθεί' ώστε να ακούει τα γεγονότα που παράγονται από το πεδίο *rotation_changed* του αισθητήρα.

Και εδώ όπως και προηγουμένως για όλους τους αισθητήρες τύπου *SphereSensor* διαβάζεται η τιμή του πεδίου '*touchTime*' και καταχωρείται σε μια μεταβλητή τύπου SFTime.

```
SFRotation cylTranslation = (SFRotation) cylSensor.getField("rotation_changed");
```

Ενώ στη συνέχεια δίνουμε ένα μοναδικό όνομα - ταυτότητα στο γεγονός που πρόκειται να παράγει ο κάθε αισθητήρας, το οποίο δεν είναι άλλο από το όνομα του ίδιου του αισθητήρα.

```
cylTranslation.setUserData(cylSensorList[o]);
```

Με την παρακάτω εντολή το πρόγραμμα προσθέτει τον Listener στον αισθητήρα.

```
cylTranslation.addX3DEventListener(this);
```

```
if (ftype=="CylinderSensor")
{
    cylSensorList[o]=nn;

    //Find the Cylinder Sensor
    cylSensor=mainScene.getNamedNode(cylSensorList[o]);
    X3DFieldDefinition d[]=cylSensor.getFieldDefinitions();
    SFRotation cylTranslation = (SFRotation) cylSensor.getField("rotation_changed");
    cylTranslation.setUserData(cylSensorList[o]);

    //add listener to CylinderSensor
```

```
        cylTranslation.addX3DEventListener(this);
        o++;
    }
```

ΑΝΙΧΝΕΥΣΗ ΚΑΙ ΑΝΑΛΥΣΗ ΓΕΓΟΝΟΤΟΣ

Με την διαδικασία προσθήκης listeners στους αισθητήρες της σκηνής, το πρόγραμμα είναι πλέον σε θέση να ανιχνεύσει γεγονότα που προκάλεσε ο χρήστης.

Με την ανίχνευση ενός γεγονότος το πρόγραμμα ξεκινάει την εκτέλεση της συνάρτησης **readableFieldChanged.** Η συνάρτηση ανιχνεύει το είδος του αισθητήρα που παρήγαγε το γεγονός καθώς και το ίδιο το όνομα του αισθητήρα.

- Ανίχνευση αισθητήρα PlaneSensor που παρήγαγε το γεγονός

Η συνάρτηση έχει σαν όρισμα το event που ανιχνεύθηκε από τον x3dEventListener. Χρησιμοποιώντας το event αυτό στην παρακάτω εντολή καταχωρήθηκε το αντικείμενο στο οποίο προκλήθηκε το γεγονός στη μεταβλητή source, τύπου SFVec3f .

```
SFVec3f source=(SFVec3f) event.getSource();
```

Έχοντας το αντικείμενο σαν σημείο αναφοράς και με την παρακάτω εντολή λαμβάνουμε το πεδίο του αισιητήρα που προκάλεσε το γεγονός, που δεν μπορεί να είναι άλλο από το πεδίο translation_changed, αφού σε αυτό το πεδίο προστέθηκε ο x3dEventListener.

```
eventName=source.getDefinition().getName();
```

Με την παρακάτω εντολή το πρόγραμμα φτάνουμε στο στόχο μας αφού το πρόγραμμα βρίσκει πλέον το DEF όνομα του αισθητήρα που προκάλεσε το γεγονός που εξετάζεται. Η συγκεκριμένη εντολή μπορεί και παρέχει τη δυνατότητα αυτή λόγω της εντολής planeTranslation.setUserData(planeSensorList[k]); που χρησιμοποιήθηκε προηγουμένως στον x3deventListener.

```
whichSensor = event.getData().toString();
```

Η ανίχνευση του ονόματος για τους αισθητήρες CylinderSensor, SphreSensor, Touchsensor που παρήγαγαν γεγονότα πραγματοποιείται από το πρόγραμμα ακριβώς με τον τρόπο που περιγράφηκε παραπάνω για τον αισθητήρα PlaneSensor

ΕΥΡΕΣΗ ΝΕΩΝ ΣΥΝΤΕΤΑΓΜΕΝΩΝ

Η ενεργοποίηση κάποιου από τους αισθητήρες της σκηνής προκαλεί και την αλλαγή στη θέση του κόμβου με τον οποίο είναι συνδεδεμένος ο αισθητήρας που ενεργοποιήθηκε.

Λόγω του διαφορετικού πεδίου από το οποίο παράγει γεγονός το κάθε είδος αισθητήρα, θα έχουμε και διαφορετικούς τρόπους εύρεσης των νέων συντεταγμένων για τους κόμβους που υπέστησαν μεταβολή της θέσης του στη σκηνή.

Το πρόγραμμα χρησιμοποίησε τη συνάρτηση parseX3D για να βρει τα ονόματα των κόμβων κόμβους με τους οποίους συνδέονται οι αισθητήρες TouchSensor, SphereSensor και CylinderSensor

Η παρακάτω συνάρτηση αναλύει το X3D αρχείο που έχει φορτώσει ο χρήστης στη σκηνή. Στόχος της συνάρτησης είναι να καταχωρήσουμε σε 2 πίνακες τους κόμβους που συνδέει κάθε ROUTE. Έτσι με τον παρακάτω κώδικα δημιουργούνται 2 public String πίνακες, ο fromNode και ο toNode. Στον πίνακα fromNode καταχωρούνται τα DEF ονόματα των κόμβων προορισμού του κάθε ROUTE της σκηνής ενώ στον πίνακα toNode και στις ισότιμες σειρές του πίνακα καταχωρούνται τα ονόματα των κόμβων πηγής του ROUTE.
Μελλοντικά με τη χρήση της συνάρτησης και γνωρίζοντας απλά το όνομα ενός κόμβου, θα μπορούμε να λαμβάνουμε πληροφορίες, χρησιμοποιώντας αυτούς τους πίνακες για το αν αυτός ο κόμβος βρίσκεται σε σύνδεση με κάποιον άλλον μέσω ROUTE και αν ναι, θα είναι εύκολο να βρεθεί και το όνομα του άλλου κόμβου.

```
public void parseX3D()
{
//The function parses the current X3D file
  try
     {

    // Get the loaded file
     File f  = new File(mainScene.getWorldURL()+fileToOpen);
     DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();
     DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
     Document doc = docBuilder.parse(f.getPath());
     doc.getDocumentElement().normalize();

     //Get the Elements named X3D
     NodeList root = doc.getElementsByTagName("X3D");
     Element myNode1  = (Element)root.item(0);

     // Gets all the Elements in the scene named Route
     NodeList root1 = myNode1.getElementsByTagName("ROUTE");
     int k=0;

     // Create fromNode Array that will be used to store the source node names of every Route in
     the scene
     fromNode = new String[root1.getLength()];

     // Create toNode Array that will be used to store the destination node names of every Route
     in the scene
     toNode = new String[root1.getLength()];
     while(k<root1.getLength())
     {
          Element myNode = (Element) root1.item(k);
          Attr myAttr1 = myNode.getAttributeNode("fromNode");
          fromNode[k] = myAttr1.getValue();
          Attr myAttr2 = myNode.getAttributeNode("toNode");
          toNode[k] = myAttr2.getValue();
```

```
    k++;
}
}catch(Exception e){e.printStackTrace();}
}
```

- **Planesensor**

Για την εύρεση των νέων συντεταγμένων του κόμβου με τον οποίο συνδέεται ο αισθητήρας Planesensor που ενεργοποιήθηκε, το πρόγραμμα αρχικά ελέγχει αν η το γεγονός παρήχθη από πεδίο τύπου translation_changed, αφού αν ο χρήστης ενεργοποιήσει αισθητήρα αυτού του είδους παράγεται ένα transaltion_changed γεγονός.

Και τέλος εκτελώντας την παρακάτω εντολή λαμβάνεται το αντικείμενο που μετακίνησε ο χρήστης και μετατρέπεται (cast) σε τύπο SFVec3f.

```
SFVec3f source=(SFVec3f) event.getSource();
```

Έχοντας όλα τα παραπάνω και με την παρακάτω εντολή, το πρόγραμμα βρίσκει τις νέες συντεταγμένες και τις αποθηκεύει στον Float πίνακα data.

```
SFVec3f source=(SFVec3f) event.getSource();
```

```
// If the user has activated a PlaneSensor node a transaltion_changed event is generated
if (eventName=="translation_changed")
{
        // Get the object that has generated the event and cast it to (SFVec3f)
        SFVec3f source=(SFVec3f) event.getSource();

        // Get the value of the SFVec3f (get the coordinates of the node that is accosiated with
        the PlaneSensor that the user has activated)
        source.getValue(data);
}
```

- **CylinderSensor και SphereSensor**

Για την εύρεση των νέων συντεταγμένων του κόμβου με τον οποίο συνδέονται οι αισθητήρες CylinderSensor και SphereSensor , το πρόγραμμα εκτελεί την ίδια διαδικασία με παραπάνω με την μόνη διαφορά ότι ο έλεγχος για την παραγωγή γεγονότος γίνεται στο πεδίο rotation_changed.

Το πρόγραμμα χρησιμοποιεί την συνάρτηση sphereCoords η οποία γνωρίζοντας το όνομα του αισθητήρα που ενεργοποιήθηκε επιστρέφει το όνομα του κόμβου που μετακίνησε ο αισθητήρας.

```
public String sphereCoords(String whichSensor)
   {
       //Destination node
       String destiny="";
       String sphereNode="";
       for (int k=0; k<fromNode.length; k++)
       {
```

```
        if (whichSensor.equals(fromNode[k]))
        {
           sphereNode=toNode[k];
        }
      }
      return sphereNode;
   }
```

Γνωρίζοντας το όνομα του κόμβου που μετακινήθηκε, παίρνει τον κόμβο χρησιμοποιώντας την εντολή

```
X3DNode l = mainScene.getNamedNode(k);
```

και με τρόπο όμοιο με αυτόν του PlaneSensor παίρνει το πεδίο rotation του κόμβου από το όποιο και βρίσκει τις νέες συντεταγμένες του κόμβου.

```
// If the user has activated a CylinderSensor or a SphereSensor a rotation event is generated
  if (eventName=="rotation_changed")
  {
     String k = sphereCoords(whichSensor);
     X3DNode l = mainScene.getNamedNode(k);
     SFRotation sfSphere = (SFRotation) l.getField("rotation");
     sfSphere.getValue(rotationdata);

     data[0]=rotationdata[0];
     data[1]=rotationdata[1];
     data[2]=rotationdata[2];
  }
```

- **TouchSensor**

Για την εύρεση των νέων συντεταγμένων του κόμβου που μετακινήθηκε από αισθητήρα TouchSensor το πρόγραμμα αναζητεί τον κόμβο PositionInterpolator με τον οποίο συνδέεται ο αισθητήρας TouchSensor που ενεργοποιήθηκε. Η εύρεση του κόμβου αυτού είναι ζωτικής σημασίας, καθώς ο PositionInterpolator είναι ο κόμβος που καθορίζει τις συντεταγμένες και την τροχιά που θα ακολουθήσει ο τελικός κόμβος με τον οποίο είναι συνδεδεμένος ο TouchSensor.

Με τη συνάρτηση touchCoords που δημιουργήθηκε, η οποία επιστρέφει το DEF όνομα του PositionInerpolator, δίδεται τη δυνατότητα εύρεσης του κόμβου PositionInterpolator.

```
X3DNode PositionInterpolatorNode=mainScene.getNamedNode(interpolatorDef);
```

Γνωρίζοντας τον PositionInterpolator του TouchSensor που ενεργοποιήθηκε η διαδικασία εύρεσης των νέων συντεταγμένων, είναι πανομοιότυπη με την διαδικασία που υλοποιήθηκε για τους PlaneSensor αισθητήρες.

Στην περίπτωση του αισθητήρα touchSensor η αποστολή των συντεταγμένων γίνεται αμέσως μόλις βρεθούν οι νέες συντεταγμένες του κόμβου με τον οποίο συνδέεται

στη σκηνή. Παρακάτω θα εξηγηθεί αναλυτικά η διαδικασία αποστολής για όλα τα είδη των αισθητήρων.

```
if (eventName=="touchTime")
  {
      String  interpolatorDef="";
      flag=1;
      int counter=0;
      while (flag==1)
      {
         //touchCoords function returns the name of the sensor that has generated the event
         interpolatorDef=touchCoords(whichSensor);

         X3DNode PositionInterpolatorNode=mainScene.getNamedNode(interpolatorDef);
         SFVec3f fractionFloat = SFVec3f)PositionInterpolatorNode.getField("value_changed");

         dataBefore[0] = data[0];
         dataBefore[1] = data[1];
         dataBefore[2] = data[2];

         fractionFloat.getValue(data);

         dataAfter[0] = data[0];
         dataAfter[1] = data[1];
         dataAfter[2] = data[2];

         if (dataBefore[0] == dataAfter[0] && dataBefore[1] == dataAfter[1] &&
         dataBefore[2] == dataAfter[2])
        {
            //an io times tou timesensor einai panw apo 4 fores idies
            if (counter == 4)
            {
               flag = 0;
            }
            counter=counter+1;
         }
            else
            flag = 1;

         //concatenate strings
         String x=Float.toString(data[0]);
         String y=Float.toString(data[1]);
         String z=Float.toString(data[2]);


makeXML(mainScene.getNamedNode(interpolatorDef).getNodeName(),interpolatorDef,x,y,z,"0");
send();
      }
    }
```

ΑΠΟΣΤΟΛΗ ΤΩΝ ΣΥΝΤΕΤΑΓΜΕΝΩΝ ΣΤΟΝ ΑΠΟΜΑΚΡΥΣΜΕΝΟ ΧΡΗΣΤΗ

Βρισκόμαστε πλέον στο στάδιο όπου έχει παραχθεί ένα γεγονός, έχει ανιχνευθεί το είδος του γεγονότος, έχει βρεθεί ο κόμβος ο οποίος (λόγω της σύνδεσης του με το γεγονός) έχει αλλάξει τη θέση του στη σκηνή και τέλος έχουν βρεθεί και οι νέες συντεταγμένες του κόμβου αυτού.

Το επόμενο βήμα για να φθάσουμε στο στόχο μας είναι η αποστολή των παραπάνω στοιχείων στον απομακρυσμένο χρήστη και στη συνέχεια κάτι που θα δούμε σε επόμενο βήμα, ο απομακρυσμένος χρήστης να αξιοποιήσει αυτές τις πληροφορίες, ανάλογα.

Πρώτου αρχίσει η διαδικασία δημιουργίας του αρχείου XML είναι απαραίτητο να αναφερθεί ο τα΄

1) ΔΗΜΙΟΥΡΓΙΑ ΑΡΧΕΙΟΥ XML

Η αποστολή των παραπάνω δεδομένων στον απομακρυσμένο χρήστη γίνεται με τη βοήθεια αρχείων xml, που αποθηκεύονται τοπικά στο δίσκο του χρήστη από όπου θα διαβαστούν στο επόμενο βήμα για να μεταβιβασθούν στον απομακρυσμένο χρήστη. Τα αρχεία αυτά δομούνται κατάλληλα ώστε να παρέχουν πληροφορίες που αφορούν:

- ✓ Το είδος του αισθητήρα που ενεργοποιήθηκε
- ✓ Το DEF όνομα του αισθητήρα
- ✓ Τις νέες συντεταγμένες του κόμβου και τέλος
- ✓ Η περιστροφή του κόμβου

```
<?xml version="1.0" encoding="UTF-8"?>
<X3d>
   <MODIFICATION>
      <SENSOR>
               PositionInterpolatorDEF="WindowTimeSensor"Coordinates=0.0_1.0_0.0Rotat0069on=0
          </SENSOR>
   </MODIFICATION>
</X3d>
```

Παράδειγμα XML αρχείου που θα σταλεί στον απομακρυσμένο χρήστη

Η συνάρτηση δημιουργίας του αρχείου, καλείται με τα παρακάτω ορίσματα κατά σειρά:

- ➢ Είδος του αισθητήρα
- ➢ DEF όνομα του αισθητήρα που ενεργοποιήθηκε
- ➢ Συντεταγμενένη X του κόμβου που μετακινήθηκε
- ➢ Συντεταγμενένη Y του κόμβου που μετακινήθηκε
- ➢ Συντεταγμενένη Z του κόμβου που μετακινήθηκε
- ➢ Περιστροφή του κόμβου που μετακινήθηκε

```
makeXML(mainScene.getNamedNode(interpolatorDef).getNodeName(),interpolatorDef,x,y,z,"0");
```

Στην περίπτωση ενεργοποίησης αισθητήρα TouchSensor το πρώτο όρισμα δεν θα πάρει την τιμή του είδους του αισθητήρα που ενεργοποιήθηκε (TouchSensor) αλλά την τιμή του PositionInterpolator.

Αυτό γίνεται γιατί η συνάρτηση είναι σχεδιασμένη με τέτοιο τρόπο ώστε γνωρίζοντας το DEF όνομα του αισθητήρα που προκάλεσε το γεγονός, να οδηγούμαστε αμέσως μέσω του Route με το οποίο είναι συνδεδεμένος ο αισθητήρας στον κόμβο προορισμού (τον κόμβο του οποίου άλλαξε η θέση στη σκηνή).

Ο αισθητήρας TouchSensor δεν λειτουργεί με τον τρόπο που λειτουργούν οι υπόλοιποι, λόγω του ότι δεν ανήκει στην κατηγορία των αισθητήρων αφής. Ο κόμβος του αισθητήρα χρησιμοποιεί δυο συνδέσεις για να φτάσει στον τελικό του κόμβο. Η πρώτη είναι μεταξύ timesensor και positionInterpolaor και η δευτερη μεταξύ positionInterpolator και κόμβου της σκηνής.

Ακολουθεί η συνάρτηση δημιουργίας των xml αρχείων.

```
public void makeXML(String KindOfSensor,String DEFSensorName,String x,String y, String
z,String rotation)
{

    if (KindOfSensor=="CylinderSensor")
    {
       x="0";
       y="0";
       z="0";
    }

    String sensorDEFString= KindOfSensor+" DEF="+"'"'"+DEFSensorName+"'"'";
    try
       {
       FileWriter out = new FileWriter(xmlFileUrl+"\\Acreated.xml");
       out.flush();
       d = new DocumentImpl();
       Element root = d.createElement("X3d");
       d.insertBefore(root,null);
       OutputFormat o = new OutputFormat(d);
       o.setIndent(5);
       o.setIndenting(true);
       XMLSerializer X = new XMLSerializer(o);
       X.setOutputCharStream(out);

       NodeList s1 = d.getElementsByTagName("X3d");
       firstPersonNode = s1.item(0);
       Element first = (Element)firstPersonNode;

       String mod="MODIFICATION";
       Node modification = (Node) d.createElement(mod);
       Node sensorTag = (Node) d.createElement("SENSOR");

       firstPersonNode.appendChild(modification);
       modification.appendChild(sensorTag);
```

```
        TextImpl nameOfSensor=(TextImpl)d.createTextNode(KindOfSensor+"DEF=\""+DEFSensorName+
              "\"Coordinates="+x+"_"+y+"_"+z+"Rotation="+rotation);
               sensorTag.appendChild(nameOfSensor);

        X.serialize(d);
        out.flush();
        out.close();
    }
    catch(Exception e){e.printStackTrace();}{}
}
```

2) ΑΠΟΣΤΟΛΗ XML ΑΡΧΕΙΟΥ

Μετά την δημιουργία του αρχείου xml, σειρά έχει η αποστολή του στον απομακρυσμένο χρήστη. Για την αποστολή των αρχείων xml στον απομακρυσμένο χρήστη δημιουργήθηκε ένα δίκτυο που για την επικοινωνία του χρησιμοποιεί πρωτόκολλο UDP.

Τα xml αρχεία που πρόκειται να παραληφθούν δεν έχουν ένα ορισμένο μέγεθος. Το μέγεθος ενός αρχείου εξαρτάται από το αρχείο που έχει φορτώσει ο χρήστης κάθε φορά. Η λύση του να οριστεί σαν μέγεθος παραλαμβανόμενου αρχείου ένας πολύ μεγάλος αριθμός, ώστε να αποφευχθεί η αποστολή ενός μόνο μέρους του αρχείου, δεν εξυπηρετεί διότι στην περίπτωση που το αρχείο που δημιουργήθηκε είναι μικρότερου μεγέθους από το ορισμένο, τότε το αρχείο στον ελεύθερο χώρο που του απομένει θα γεμίσει με άχρηστα bytes, και ο απομακρυσμένος χρήστης δεν θα μπορεί να πάρει την πληροφορία που χρειάζεται.

Αφού λοιπόν απομακρυσμένος χρήστης πρέπει να γνωρίζει πόσα ακριβώς bytes πρόκειται να λάβει, η μόνη λύση στο πρόβλημα, είναι να υπολογιστεί από πριν το ακριβές μέγεθος του αρχείου (ο αριθμός δηλαδή των byte που πρόκειται να περάσουν από το δίκτυο) και να σταλεί στον απομακρυσμένο χρήστη πριν την αποστολή του αρχείου.

Για το λόγο αυτό εκτός από το socket που θα χρησιμοποιηθεί για την αποστολή του xml αρχείου, χρειάζεται να δημιουργηθεί κι ένα άλλο socket το οποίο θα στέλνει από πριν τον αριθμό των byte του αρχείου που πρόκειται να λάβει ο απομακρυσμένος χρήστης.

Το socket που χρησιμοποιείται για την αποστολή και στη συνέχεια τη λήψη (από τον απομακρυσμένο χρήστη) των αρχείων ονομάζεται socket, ενώ το socket που χρησιμοποιείται για την αποστολή του αριθμού των byte του αρχείου ονομάζεται sizeSocket.

Ακολουθεί η συνάρτηση αποστολής των αρχείων (send):

```
public void send()
{
      try
      {
          File file=new File(xmlFileUrl+"\\Acreated.xml");
          FileInputStream fileInStream=new FileInputStream(file);
          bytesend = new byte [fileInStream.available()];
          for (int i=0; i<bytesend.length; i++)
```

```
            {
             int sizeint=fileInStream.available();
             if (sizeint>0)
             {
                 String myString = Integer.toString(sizeint);
                 byte kko[]=myString.getBytes();
                 sizeRpacket = new DatagramPacket(kko, kko.length, address, portSsize);

                 sizeSocket.send(sizeRpacket);

                 statusLabel.setText("\n\n\nSending Data to the remote user...");
                 fileInStream.read(bytesend);
                 spacket = new DatagramPacket(bytesend, bytesend.length, address, portS);
                 socket.send(spacket);
                 spacket=null;

             }
           }
       }catch(Exception e){e.printStackTrace();}
}
```

Στη συνέχεια ακολουθεί η συνάρτηση readableFieldChanged η οποία εκτελεί όλα αυτά που λέχθηκαν παραπάνω. Συνοψίζοντας, 'ακούει' το γεγονός που παρήγαγε ο χρήστης, ανιχνεύει ποιος από όλους τους αισθητήρες ενεργοποιήθηκε και στη συνέχεια ανάλογα με τον αισθητήρα που ενεργοποιήθηκε, βρίσκει τις συντεταγμένες του κόμβου που άλλαξε τη θέση του στην x3d σκηνή που φόρτωσε ο χρήστης. Όλα αυτά τα παραπάνω δεδομένα, ομαδοποιούνται σε ένα xml αρχείο και στη συνέχεια αποστέλλεται στον απομακρυσμένο χρήστη.

```
public void readableFieldChanged(org.web3d.x3d.sai.X3DFieldEvent event)
   {
      //Everytime a node is changed by the user, an event is generated.
      float data[]=new float[3];

     //Array that contains the data of the previous Touchsensor position
     float dataBefore[]=new float[3];

     //Array that contains the current data of the Touchsensor's position
     float dataAfter[]=new float [3];

     //Array that contains the data of the
     float rotationdata[]=new float[4];

      String destinationNode="",whichSensor="",positionInterpolator = "";

     //The flag is used to check if the previous Touchsensor data are equal to the current ones
      int flag;

      //According to which sensor the user has activated, it happens one of the following...
      X3DRoute routs[]=mainScene.getRoutes();
```

```
// In case the user has activated a PlaneSensor node
 try
       {

       // Get the object in which the event has occured ( get The sensor that has generated
       the event)
        SFVec3f source=(SFVec3f) event.getSource();

       // Get the field of the object e.g. translation_changed that will be used
       eventName=source.getDefinition().getName();

       // Get the DEF name of the sensor that has generated the event
       whichSensor = event.getData().toString();
       }
       catch(Exception ex){}

// In case the user has activated a TouchSensor node
 try
    {
    SFTime field = (SFTime) event.getSource();
    eventName=field.getDefinition().getName();
     whichSensor = event.getData().toString();
     }
 catch(Exception exe){}


// In case the user has activated a CylinderSensor node
 try
    {
    SFRotation source=(SFRotation) event.getSource();
    eventName=source.getDefinition().getName();
    whichSensor = event.getData().toString();
     }
 catch(Exception ex){}

// In case the user has activated a TouchSensor or a CylinderSensor node
 try
    {
    SFFloat source=(SFFloat) event.getSource();
    eventName=source.getDefinition().getName();
    whichSensor = event.getData().toString();
    }
 catch(Exception ex){}

// If the user has activated a PlaneSensor node a translation_changed event is generated
 if (eventName=="translation_changed")
 {
    // Get the object that has generated the event and cast it to (SFVec3f)
    SFVec3f source=(SFVec3f) event.getSource();

    // Get the value of the SFVec3f (get the coordinates of the node that is accosiated with
    the PlaneSensor that the user has activated)
```

```
    source.getValue(data);
 }

// If the user has activated a CylinderSensor a rotation event is generated
 if (eventName=="rotation_changed")
 {
    String k = sphereCoords(whichSensor);
    X3DNode l = mainScene.getNamedNode(k);
    SFRotation sfSphere = (SFRotation) l.getField("rotation");
    sfSphere.getValue(rotationdata);

    data[0]=rotationdata[0];
    data[1]=rotationdata[1];
    data[2]=rotationdata[2];
 }

 if (eventName=="touchTime")
{
    String  interpolatorDef="";
    flag=1;
    int counter=0;
    while (flag==1)
    {
       //touchCoords function returns the name of the sensor that has generated the event
       interpolatorDef=touchCoords(whichSensor);

       X3DNode PositionInterpolatorNode=mainScene.getNamedNode(interpolatorDef);
       SFVec3f fractionFloat = SFVec3f)PositionInterpolatorNode.getField("value_changed");

       dataBefore[0] = data[0];
       dataBefore[1] = data[1];
       dataBefore[2] = data[2];

       fractionFloat.getValue(data);

       dataAfter[0] = data[0];
       dataAfter[1] = data[1];
       dataAfter[2] = data[2];

       if (dataBefore[0] == dataAfter[0] && dataBefore[1] == dataAfter[1] &&
       dataBefore[2] == dataAfter[2])
      {
          //an io times tou timesensor einai panw apo 4 fores idies
          if (counter == 4)
          {
             flag = 0;
          }
          counter=counter+1;
       }
          else
          flag = 1;

       //concatenate strings
```

```
            String x=Float.toString(data[0]);
            String y=Float.toString(data[1]);
            String z=Float.toString(data[2]);


makeXML(mainScene.getNamedNode(interpolatorDef).getNodeName(),interpolatorDef,x,y,z,"0");
send();
          }
       }


     if (eventName!="touchTime")
       {
          //concatenate strings
          String x=Float.toString(data[0]);
          String y=Float.toString(data[1]);
          String z=Float.toString(data[2]);
          String rotation=Float.toString(rotationdata[3]);

makeXML(mainScene.getNamedNode(whichSensor).getNodeName(),whichSensor,x,y,z,rotation);
          send();
       }
   }
```

ΛΗΨΗ XML ΑΡΧΕΙΟΥ ΑΠΟ ΤΟΝ ΑΠΟΜΑΚΡΥΣΜΕΝΟ ΧΡΗΣΤΗ

Μετά την αποστολή του XML αρχείου, στον απομακρυσμένο χρήστη, σειρά έχει η διαδικασία, λήψης του αρχείου από τον απομακρυσμένο χρήστη. Η εξαγωγή των κατάλληλων τιμών που περιέχονται στις ετικέτες του αρχείου και τέλος η δημιουργία κίνησης στον κόμβου στον απομακρυσμένου χρήστη που αντιστοιχεί με τον κόμβο που μετακίνησε ο χρήστης.

- **ΛΗΨΗ XML ΑΡΧΕΙΟΥ**

Στο πρόγραμμα έχει δημιουργηθεί ένα νήμα στο οποίο έχει σαν σκοπό την λήψη των xml αρχείων που στέλνει ο απομακρυσμένος χρήστης και στη συνέχεια την αναπαράσταση της κίνησης που δημιούργησε ο απομακρυσμένος χρήστης .

Η μέθοδος run του νήματος χρησιμοποιώντας το socket sizeSocket αρχικά λαμβάνει το μέγεθος του αρχείου που πρόκειται να στείλει ο χρήστης

```
            try{
                bytesend=new byte[4];
                sizeRpacket=new DatagramPacket(bytesend,bytesend.length);
                sizeSocket.receive(sizeRpacket);
                String str=new String(sizeRpacket.getData());
                h= Integer.parseInt(str.trim());

                }
                catch(IOException exe){exe.printStackTrace();}
```

και στη συνέχεια, μέσω ενός άλλου socket (που ονομάζεται socket) λαμβάνει το xml αρχείο.

```
message = new byte[h];
rpacket = new DatagramPacket(message,message.length);
socket.receive(rpacket);
```

Το επόμενο βήμα της μεθόδου όπως θα δούμε και παρακάτω είναι η αποκωδικοποίηση των δεδομένων του αρχείου καλώντας τη συνάρτηση parseXML. Ακολουθεί η μέθοδος run.

```
public void run()
{
    int h=0;
    while(true)
    {
       try{
          buttonConnect.setText(socket.getInetAddress().getHostName());

          try{
             bytesend=new byte[4];
             sizeRpacket=new DatagramPacket(bytesend,bytesend.length);
             sizeSocket.receive(sizeRpacket);
             String str=new String(sizeRpacket.getData());
             h= Integer.parseInt(str.trim());

          }catch(IOException exe){exe.printStackTrace();
          }

          message = new byte[h];
          rpacket = new DatagramPacket(message,message.length);
          socket.receive(rpacket);

          statusLabel.setText("\n\n\nReceiving Data from the remote user...");

          byte pin[]=rpacket.getData();
          String stra=new String(rpacket.getData());

          // Open or create the output file

          FileOutputStream os = new FileOutputStream(xmlFileUrl+"\\Areceived.xml");
          FileDescriptor fd = os.getFD();

          // Write some data to the stream
          // Flush the data from the streams and writers into system buffers.
          // The data may or may not be written to disk.
          os.write(pin);

          // Block until the system buffers have been written to disk
          // After this method returns, the data is guaranteed to have been written to disk.
          os.flush();                    .

          fd.sync();
```

```
            parseXML();
            rpacket=null;

        }catch(IOException exe){exe.printStackTrace();}
    }

}
```

- **ΑΠΟΚΩΔΙΚΟΠΟΙΗΣΗ XML ΑΡΧΕΙΟΥ**

Για κάθε νέο XML αρχείο που λαμβάνει ο χρήστης, το πρόγραμμα πραγματοποιεί την ανάλυση των δεδομένων που περιέχει το αρχείο. Η ανάλυση αυτών των δεδομένων είναι απαραίτητη για την δημιουργία του αντιγράφου της κίνησης που προηγουμένως δημιούργησε ο απομακρυσμένος χρήστης. Οι παραπάνω διαδικασία, εκτελείται από τη συνάρτηση **parseXML**().

Σκοπός της συνάρτησης, είναι η εξαγωγή των τιμών από τις ετικέτες (tags) του αρχείου που θα χρησιμοποιηθούν αργότερα ώστε να δώσουμε τις νέες τιμές στους κόμβους που πρέπει να μετακινηθούν.

Όπως προαναφέρθηκε, η δομή του αρχείου είναι η παρακάτω:

```
<?xml version="1.0" encoding="UTF-8"?>
<X3d>
   <MODIFICATION>
      <SENSOR>
               PositionInterpolatorDEF="WindowTimeSensor"Coordinates=0.0_1.0_0.0Rotat006
               9on=0
          </SENSOR>
   </MODIFICATION>
</X3d>
```

Οι τιμές που χρειάζεται να εξαχθούν από το αρχείο για δημιουργία της κίνησης αφορούν το είδος του αισθητήρα που προκάλεσε το γεγονός, το όνομα του αισθητήρα και τις συντεταγμένες.

Το πρόγραμμα ξεκινά την ανάλυση του αρχείου από την ετικέτα <X3d>. Παίρνει τα στοιχεία της ετικέτας και στην συνέχεια προχωρά στην ετικέτα <MODIFICATION> για να καταλήξει στην ετικέτα <SENSOR> που είναι και η κύρια ετικέτα του αρχείου. Από τη συγκεκριμένη ετικέτα λαμβάνονται τα δεδομένα που απαιτούνται.

```
        NodeList root = doc.getElementsByTagName("X3d");
        Element root1 = (Element) root.item(0);
        NodeList nodeList = root1.getElementsByTagName("MODIFICATION");
        Element name = (Element) nodeList.item(0);
        NodeList nameList = name.getElementsByTagName("SENSOR");
        Node sensorNode = nameList.item(0);
        Node SENSORcontents = sensorNode.getFirstChild();
        String sensorINFO = new String(SENSORcontents.getNodeValue());

        //Get the Coordinates…
```

```
parsedSensor = sensorINFO.split("Coordinates=");
parsedKind=parsedSensor[0].split("DEF");
parsedRotation=parsedSensor[1].split("Rotation=");
parsedKindOSensor=parsedKind[0];
parsedSensorN = sensorINFO.split("=");
parsedSensorName=parsedSensorN[1].substring(1,parsedSensorN[1].length()-12);
//whichSensor is moved
parsedStringCoords=parsedRotation[0].split("_");

//Parse the coordinates from String to Float
for (int l=0; l<parsedFloatCoords.length-1; l++)
{
   parsedFloatCoords[l] = Float.parseFloat(parsedStringCoords[l]);
}
```

Όπως είναι φανερό στον παραπάνω κώδικα

- Το <u>είδος</u> του αισθητήρα που ενεργοποιήθηκε καταχωρήθηκε στη μεταβλητή **parsedKindOSensor**
- Το <u>όνομα</u> του αισθητήρα που ενεργοποιήθηκε καταχωρήθηκε στη μεταβλητή **parsedSensorName**
- Οι τιμές των συντεταγμένων καταχωρήθηκαν στον Float πίνακα **parsedFloatCoords** και τέλος
- Η τιμή της περιστροφής καταχωρήθηκε στην μεταβλητή **parsedRotation**

Ο αισθητήρας (κόμβος) του οποίου το όνομα που πήραμε από το xml αρχείο σύμφωνα με τον τρόπο δημιουργίας των Routes του X3d αποτελεί τον κόμβο πηγής του Route. (sourceNode). Το πρόγραμμα χρησιμοποιώντας το όνομα του αισθητήρα βρίσκει τον κόμβο, και από εδώ και στο εξής θα τον θεωρεί sourceNode του Route. Στόχος είναι να καταλήξει στον κόμβο προορισμού του Route και να του αποδώσει τις τιμές του αρχείου στο κατάλληλο πεδίο του κόμβου που πρέπει να μετακινηθεί.

Το πρόγραμμα αρχικά αναζητεί το όνομα του αισθητήρα (parsedSensorName) που έλαβε από το αρχείο ανάμεσα σε όλους τους ονοματισμένους κόμβους της σκηνής. Στη συνέχεια με τη βοήθεια της συνάρτησης getNamedNode καταχωρείται ο κόμβος του προγράμματος στη μεταβλητή sourceNode

```
X3DNode sourceNode= mainScene.getNamedNode(parsedSensorName);
```

Το πρόγραμμα αναζητεί τον κόμβο που είναι ίσος με τον κόμβο sourceNode που βρήκαμε πιο πάνω σε όλα τα routes της σκηνής. Το route του οποίου ο κόμβος πηγής είναι ίσος με τον κόμβο που βρέθηκε παραπάνω είναι αυτό που θα μας βοηθήσει να φτάσουμε στο στόχο μας.

Στη συνέχεια μέσω του Route στο οποίο καταλήξαμε μέσω της αναζήτησης, βρίσκουμε τον κόμβο προορισμού και καταλήγουμε στο στόχο μας.

```
X3DNode destNode = routs[l].getDestinationNode();
```

Το μόνο που απομένει είναι να αποδοθούν οι κατάλληλες τιμές που αναλύθηκαν από το xml αρχείο, στο κατάλληλο πεδίο του κόμβου, κάτι που εξαρτάται από το είδος του αισθητήρα που δημιούργησε το γεγονός στον απομακρυσμένο χρήστη. Ακολουθεί η διαδικασία για καθέναν από τους αισθητήρες.

- **Touchsensor**

Στην περίπτωση που από το αρχείο ληφθεί η πληροφορία ότι ενεργοποιήθηκε αισθητήρας τύπου Touchsensor το xml για τους λόγους που αναλύθηκαν παραπάνω, σαν είδος αισθητήρα θα επιστρέψει την τιμή "PositionInterpolator".

Με την επιστροφή της παραπάνω πληροφορίας, το πρόγραμμα ανανεώνει τις τιμές του πεδίου "translation_changed" του κόμβου που ανιχνεύθηκε παραπάνω, μέσω του ιδεατού αγωγού μεταφοράς συμβάντων (route) με τις νέες τιμές που αναλύθηκαν προηγουμένως από το αρχείο xml.

```
if(parsedKindOSensor.equals("PositionInterpolator"))
{
        try
        {
                SFVec3f coor = (SFVec3f)destNode.getField("translation_changed");
                coor.setValue(parsedFloatCoords);
        }catch(Exception e){e.printStackTrace();
}
```

- **PlaneSensor**

Στην περίπτωση που από το αρχείο ληφθεί η πληροφορία ότι ενεργοποιήθηκε αισθητήρας τύπου PlaneSensor ακολουθείται η ίδια διαδικασία που περιγράφηκε για τον προηγούμενο αισθητήρα.

```
if(parsedKindOSensor.equals("PlaneSensor"))
 try
 {
        SFVec3f coor = (SFVec3f)destNode.getField("translation");
        coor.setValue(parsedFloatCoords);
}catch(Exception e){e.printStackTrace();}
```

- **CylinderSensor**

Στην περίπτωση που η μεταβλητή parsedKindOSensor λάβει την τιμή CylinderSensor οι τιμές του πεδίου " rotation " του κόμβου destNode ανανεώνονται με τις τιμές που λήφθηκαν από το xml.

```
try
   {
        if(parsedKindOSensor.equals("CylinderSensor"))
        {
           SFRotation coor = (SFRotation)destNode.getField("rotation");
           coor.setValue(new float[]
           {
```

```
                parsedFloatCoords[0],1f,parsedFloatCoords[2],Float.parseFloat(parsedRotation[1])
            });
        }
    }catch(Exception e){e.printStackTrace();}
```

- **SphereSensor**

Στην περίπτωση που η μεταβλητή parsedKindOSensor λάβει την τιμή SphereSensor οι τιμές του πεδίου " rotation " του κόμβου destNode ανανεώνονται με τις τιμές που λήφθηκαν από το xml.

Ο συγκεκριμένος αισθητήρας προκαλεί μόνο την περιστροφή ενός κόμβου και όχι την μετατόπιση της θέσης του. Συνεπώς η μόνη τιμή που απαιτείται να μεταβληθεί, είναι η τιμή της περιστροφής του πεδίου rotation_changed του κόμβου destNode.

```
try
  {
      if(parsedKindOSensor.equals("SphereSensor"))
      {
          SFRotation coord = (SFRotation)destNode.getField("rotation_changed");
          parsedFloatCoords[3]=Float.parseFloat(parsedRotation[1]);

          float pin[]=new float[4];
          pin[1] =0.0f;
          pin[2]=0.0f;
          pin[3]=0.0f;
          pin[3]=Float.parseFloat(parsedRotation[1]);


          coord.setValue(parsedFloatCoords);
          float m[]=new float[4];
          coord.getValue(m);

      }
   }catch(Exception e){e.printStackTrace();}
```

Ακολουθεί η συνάρτηση parseXML

```
 public void parseXML()
 {
    X3DRoute routs[]=mainScene.getRoutes();
    float touchFloat[]={0,0,0};
    String parsedSensor[], parsedSensorN[],parsedSensorName, parsedStringCoords[],
    parsedKindOSensor, parsedKind[],parsedRotation[];
    float parsedFloatCoords[] ={0,0,0,0};
    int i=0;
    String spint="";
    DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();
    try{
        DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
        Document doc = docBuilder.parse(xmlFileUrl+"\\Areceived.xml");
```

```
doc.getDocumentElement().normalize();

NodeList root = doc.getElementsByTagName("X3d");
Element root1 = (Element) root.item(0);
NodeList nodeList = root1.getElementsByTagName("MODIFICATION");
Element name = (Element) nodeList.item(0);
NodeList nameList = name.getElementsByTagName("SENSOR");
Node sensorNode = nameList.item(0);
Node SENSORcontents = sensorNode.getFirstChild();
String sensorINFO = new String(SENSORcontents.getNodeValue());

//xwrismos twn syntetagmenwn apo to sensor
parsedSensor = sensorINFO.split("Coordinates=");
parsedKind=parsedSensor[0].split("DEF");
parsedRotation=parsedSensor[1].split("Rotation=");
parsedKindOSensor=parsedKind[0];
parsedSensorN = sensorINFO.split("=");

//whichSensor is moved
parsedSensorName=parsedSensorN[1].substring(1,parsedSensorN[1].length()-12);
parsedStringCoords=parsedRotation[0].split("_");


for (int l=0; l<parsedFloatCoords.length-1; l++)
{
   parsedFloatCoords[l] = Float.parseFloat(parsedStringCoords[l]);
   if (l==2)
      System.out.println("");
}
String sensorArrayToCompare[]=new String[nlength];

//Search all the nodes and compare them with the sensor name we parsed from the x3d
for (int k=0; k<nlength; k++)
{
   if (namedNodes[k].equals(parsedSensorName))
   {
     X3DNode sourceNode= mainScene.getNamedNode(parsedSensorName);
     for (int l=0; l<mainScene.getRoutes().length; l++)
     {
        if (routs[l].getSourceNode().equals(sourceNode))
        {
              X3DNode destNode = routs[l].getDestinationNode();
           if(parsedKindOSensor.equals("PositionInterpolator"))
           {
              try
              {
                 for (int o=0; o<parsedFloatCoords.length-1; o++)
                 {
                    touchFloat[o]=parsedFloatCoords[o];
                 }

                 SFVec3f coor = (SFVec3f)destNode.getField("translation_changed");
```

```
                        coor.setValue(parsedFloatCoords);
                    }catch(Exception e){e.printStackTrace();
                    }

                }
                if(parsedKindOSensor.equals("PlaneSensor"))
                    try
                    {
                        SFVec3f coor = (SFVec3f)destNode.getField("translation");
                        coor.setValue(parsedFloatCoords);
                    }catch(Exception e){e.printStackTrace();}

                try
                {
                  if(parsedKindOSensor.equals("CylinderSensor"))
                    {
                        SFRotation coor = (SFRotation)destNode.getField("rotation");
                        coor.setValue(new float[]
{parsedFloatCoords[0],1f,parsedFloatCoords[2],Float.parseFloat(parsedRotation[1])});
                    }
                }catch(Exception e){e.printStackTrace();}

                try
                {
                    if(parsedKindOSensor.equals("SphereSensor"))
                    {
                        SFRotation coord = (SFRotation)destNode.getField("rotation_changed");
                        parsedFloatCoords[3]=Float.parseFloat(parsedRotation[1]);

                        float pin[]=new float[4];
                        pin[1] =0.0f;
                        pin[2]=0.0f;
                        pin[3]=0.0f;
                        pin[3]=Float.parseFloat(parsedRotation[1]);


                        coord.setValue(parsedFloatCoords);
                        float m[]=new float[4];
                        coord.getValue(m);

                    }
                }catch(Exception e){e.printStackTrace();}
            }

          }
        }
      }
     for (int k=0; k<parsedSensor.length; k++)
        parsedSensor[i]=null;
      for (int m=0; m<parsedSensorN.length; m++)
        parsedSensor[i]=null;
     parsedSensorName=null;
  }
```

```
        catch(Exception e){e.printStackTrace();}
    }
```

Ακολουθεί ο constructor του προγράμματος

```
public user_with_dialog_usera()
{
        super("X3D client A");
        String ftype, nn,ftypes[];
        int k=0,h=0,o=0;
        X3DNode Nod;

        interfacex3d();

        //The number of the named Nodes
        nlength=mainScene.getNamedNodes().length;

        //The number of the scene routes
        X3DNode MapArray[] = new X3DNode[mainScene.getRoutes().length];

        //Create an array for each type of sensor (PlaneSensor, CylinderSensor, TouchSensor, CylinderSensor)
        // Each array contains a list of the DEF names from each type of sensors
        String planeSensorList[]=new String[nlength];
        String touchSensorList[]=new String[nlength];
        String cylSensorList[]=new String[nlength];
        String sphereSensorList[]=new String[nlength];

        // Get the scene's named Nodes
        namedNodes = mainScene.getNamedNodes();

        // Get the Routes of the scene
        X3DRoute r[]=mainScene.getRoutes();

       // Get the Route Nodes
        X3DNode rNodes[] = mainScene.getRootNodes();

        for (int i=0; i<nlength; i++)
        {
           nn=namedNodes[i];
           ftype = mainScene.getNamedNode(nn).getNodeName();

           if (ftype=="PlaneSensor")
           {
              planeSensorList[k]=nn;

             // Find the Plane Sensor Node
              plnSensor=mainScene.getNamedNode(planeSensorList[k]);
              SFVec3f planeTranslation = (SFVec3f) plnSensor.getField("translation_changed");

            // Accosiate user data with the translation_changed field
```

```
    planeTranslation.setUserData(planeSensorList[k]);
    planeTranslation.addX3DEventListener(this);                    // Add listener to plane
    Sensor
    k++;
}

if (ftype=="TouchSensor")
{

    touchSensorList[h]=nn;

    //Find the Touch Sensor
    tchSensor=mainScene.getNamedNode(touchSensorList[h]);
    SFTime touchTranslation = (SFTime) tchSensor.getField("touchTime");

    //Add listener to the sensor
    touchTranslation.addX3DEventListener(this);
    touchTranslation.setUserData(touchSensorList[h]);
    h++;
}

if (ftype=="SphereSensor")
{
    sphereSensorList[o]=nn;

    //Find the Sphere Sensor
    sphereSensor=mainScene.getNamedNode(sphereSensorList[o]);
    SFRotation sphereTranslation = (SFRotation)
    sphereSensor.getField("rotation_changed");
    X3DFieldDefinition [] m=sphereSensor.getFieldDefinitions();
    sphereTranslation.setUserData(sphereSensorList[o]);

    //add listener to sphereSensor
    sphereTranslation.addX3DEventListener(this);
    o++;
}


if (ftype=="CylinderSensor")
{
    cylSensorList[o]=nn;

    //Find the Cylinder Sensor
    cylSensor=mainScene.getNamedNode(cylSensorList[o]);
    X3DFieldDefinition d[]=cylSensor.getFieldDefinitions();
    SFRotation cylTranslation = (SFRotation) cylSensor.getField("rotation_changed");
    cylTranslation.setUserData(cylSensorList[o]);

    //add listener to CylinderSensor
    cylTranslation.addX3DEventListener(this);
    o++;
}
```

```
    }

    //Activate run
    clThread=new Thread(this);

   //Start run
    clThread.start();
    clThread.setPriority(1);


    setSize(800,600);
    setVisible(true);
    setDefaultCloseOperation(EXIT_ON_CLOSE);

   // Parses the current x3d file
    parseX3D();

}
```

Απαιτήσεις εργασίας
Για να επιτευχθεί η ζωντανή απεικόνιση των τριασδιάστατων αλλαγών της X3D σκηνής του χρήστη απαραίτητη είναι η ανίχνευση, η επεξεργασία και η αποστολή κάθε παραγόμενου γεγονότός στον απομακρυσμένο χρήστη, ο οποίος με την κατάλληλη επεξεργασία από τη δική του μεριά θα μπορεί να έχει την παρουσίαση των γεγονότων όπως ακριβώς δημιουργήθηκαν από τον αποστολέα.

Η ανίχνευση των γεγονότων γίνεται μέσω του SAI (**S**cene **A**ccess **I**nterface) και ειδικών Listeners.

Απαιτήσεις Συστήματος
Java: jsdk 1.4.2_04
Editor: Netbeans 3.6
X3D: Xj3D-1-0-windows-full.exe
XML Parser: Xerces-J-bin.2.9.0.zip (http://archive.apache.org/dist/xml/xerces-j/)

ΕΠΙΚΟΙΝΩΝΙΑ JAVA-Xj3D
Για να επικοινωνήσει η Java με το Xj3D χρειάζονται οι βιβλιοθήκες και τα jars βρίσκονται στο φάκελο εγκατάστασης του Xj3D, στους υποκαταλόγους jars, bin. Για να γίνει εφικτή η επικοινωνία της Java με το Xj3D τα αρχεία αυτά θα πρέπει να μεταφερθούν στον υποφάκελο ext του φακέλου στον οποίο είναι εγκατεστημένη η Java

ΕΠΙΚΟΙΝΩΝΙΑ JAVA-XML parser
Στο σύστημά μας για την ανταλλαγή δεδομένων ανάμεσα στους χρήστες, απαραίτητο ήταν να χρησιμοποιηθούν XML αρχεία. Για την δημιουργία και την προσπέλαση των αρχείων αυτών χρησιμοποιήθηκε ο XML Parser **:** Xerces. Για την επίτευξη της

επικοινωνίας της Java με τον Parser, είναι απαραίτητο να τοποθετήσουμε τα jar αρχεία του Parser στον υποκατάλογο ext της Java. Στην συνέχεια στον κώδικά μας, θα πρέπει να κάνουμε import τα συγκεκριμένα αρχεία.

6. Βιβλιογραφία

- http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification_Revision1_to_Part1/Part01/concepts.html#Scenegraphhierarchy
- http://66.102.9.104/search?q=cache:mcUER4sTsh4J:www.comp.lancs.ac.uk/~rukzio/publications/X3D2003_rukzio.pdf+externprotodeclare+explain+x3d&hl=el&ct=clnk&cd=5&gl=gr&client=firefox-a
- http://www.web3d.org/x3d/learn/web3d_2006/153-ying.pdf
- http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/Part01/components/core.html
- http://www.xml.com/pub/a/2003/08/06/x3d.html?page=4
- http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/Part01/components/pointingsensor.html#Concepts
- http://www.rcs.manchester.ac.uk/aboutus/people/students/universidaddezaragoza/2006-2007/raquelblasco/report/report.pdf
- http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/Part01/components/time.html#TimeSensor
- http://216.239.59.104/search?q=cache:wiYh8qdFa-AJ:www.leavcom.com/pdf/3Dweb.pdf+external+prototypes+x3d&hl=el&ct=clnk&cd=23&gl=gr&client=firefox-a (αναφέρεται σε x3d browser)
- http://en.wikipedia.org/wiki/X3D
- http://en.wikipedia.org/wiki/UDP
- http://en.wikipedia.org/wiki/TCP