PROJECTE DE FINAL DE CARRERA

L'entorn virtual tridimensional MOVE: Multiuser Oriented Virtual Environments

Oriol Montalà Pinyol, <omontala@tinet.org>
Carles Pairot Gavaldà, <cpairot@tinet.org>
Enginyeria en Informàtica

Directors: Pedro García López, <pgarcia@etse.urv.es>
Robert Rallo Moya, <rrallo@etse.urv.es>

Departament d'Enginyeria Informàtica i Matemàtiques (DEIM)
Escola Tècnica Superior d'Enginyeria (ETSE)
Universitat Rovira i Virgili (URV), http://www.etse.urv.es

25 de Juny de 2002

Índex

1. Introducció i Objectius

El disseny d'un entorn virtual (VE) és un problema distribuït d'accés multi-usuari a uns determinats recursos compartits. Aquest problema requereix prendre unes acurades decisions de disseny per tal de proporcionar una infrastructura de sistema capacitada per a suportar interaccions flexibles en els escenaris compartits.

La complexitat d'aquest domini ha portat a la creació d'intrincats sistemes de software que proporcionen solucions *ad-hoc* a problemes específics. Conseqüentment, molts d'ells han arribat a un camí sense sortida degut al seu disseny no-extensible i la seva manca de reusabilitat de codi i de mòduls.

L'objectiu d'aquest projecte de fi de carrera ha estat el de desenvolupar un entorn virtual tridimensional que s'ha implementat per sobre d'un marc de treball (*framework*) de components *groupware*. El nostre principal objectiu ha estat que aquest marc de components ens proporcioni una infrastructura extensible que ofereixi un conjunt de serveis col.laboratius. A nivell conceptual, proporciona els serveis col.laboratius essencials: sessions compartides, suport per a components síncrons i asíncrons, seguretat, coordinació, i a la banda del servidor, una infrastructura de "consciència" (*awareness*). A nivell arquitectònic, el marc està construït per sobre d'una plataforma d'integració *middleware* que utilitza uns serveis de notificació altament eficients de publicació / subscripció.

MOVE és un entorn tridimensional col.laboratiu que s'ha implementat utilitzant els serveis que proporciona el marc de treball anteriorment esmentat. En el nostre entorn, els usuaris (Avatars) poden interaccionar amb d'altres usuaris o amb una sèrie d'artefactes compartits, com ara eines de votació, projectors de transparències, simulacions en 3D, documents o continguts multimèdia.

La nostra aplicació és una de les aplicacions clau que s'estan provant per al projecte de la Internet2 de Catalunya i està principalment orientat al camp de l'ensenyament en diferents escenaris, tals com ara simulacions de classes virtuals, aplicacions mèdiques o arqueològiques.

Així doncs, **MOVE** es beneficia del marc de treball *ANTS Computer Supported Cooperative Work (CSCW)[34]*. A més, **MOVE** s'ha desenvolupat utilitzant tecnologies obertes, com ara el llenguatge *Virtual Reality Modeling Language (VRML97)[2]*, o l'especificació estàndard d'animació d'humanoides *H-Anim v1.1[36]*. El llenguatge de desenvolupament triat ha estat *Java* i el pont utilitzat per comunicar aquest llenguatge amb *VRML97* ha estat la interfície *External Authoring Interface (EAI)[19]*.

Per tal de permetre l'accés a l'aplicació **MOVE** també hem desenvolupat una aplicació *web*, realitzada en *Java2 Enterprise Edition[46]*, que permet, a banda del que hem esmentat, la introducció de nous usuaris al sistema, la gestió i navegació pels diferents *worlds* i *places*, la gestió d'objectes, eines, rols, etc.

2. Estat de l'art

En aquest apartat analitzarem una mica la perspectiva històrica i estat dels diferents temes que hem tractat en desenvolupar del nostre sistema. Aquests són: *VRML* per al disseny de la capa de presentació de la nostra aplicació tridimensional i *J2EE*[1] per a la resta. Proporcionem també una petita visió global sobre els diferents sistemes multiusuari, així com dels de realitat virtual, que han anat apareixent al llarg d'aquests anys.

2.1 El llenguatge VRML

El llenguatge *VRML*[2] representa el format standard de descripció d'escenes tridimensionals per a Internet. A diferència de la simplicitat de les pàgines *HTML*[3] en 2D, en un món *VRML* és possible construir un entorn tridimensional en el qual l'usuari pugui moure's, canviant el seu punt de vista mitjançant les oportunes comandes de navegació.

Un fitxer *VRML* és un simple fitxer *ASCII*[4] que conté en el seu interior la definició dels diferents aspectes que caracteritzen el món *VRML*; en particular s'han de trobar definits aquells aspectes geomètrics que construeixen la component estàtica de l'escena 3D i la component dinàmica que consisteix majoritàriament en animacions. A més, és possible monitoritzar el punt de vista de l'usuari, que consisteix en manejar oportunament els objectes del món en funció del comportament del visitant.

VRML pot ser usat en una gran varietat d'àrees d'aplicacions tals com visualització d'enginyeria i científica, presentacions multimedia, programes d'entreteniment i educacionals, pàgines Web i móns virtuals compartits.

La semàntica de *VRML* descriu un comportament abstracte i funcional de la informació multimedia basada en el temps i interactiu en 3D. No es defineixen dispositius físics o qualsevol altre concepte independent de la implementació (com la resolució de la pantalla o dispositius d'entrada de dades). L'especificació de *VRML* està pensada per a una gran varietat de dispositius i aplicacions, i dóna una gran llibertat en la interpretació i implementació de la funcionalitat. Per exemple, no s'assumeix l'existència d'un dispositiu apuntador com el ratolí o d'una pantalla 2D.

VRML ha estat dissenyat per complir amb els següents requisits:

- **Capacitat de desenvolupament:** permetre el desenvolupament de programes capaços de crear, editar i mantenir fitxers *VRML*, així com utilitats de conversió.

[1] **J2EE: Java2 Enterprise Edition.**
[2] **VRML: Virtual Reality Modeling Language.**
[3] **HTML: HyperText Markup Language.**
[4] **ASCII: American Standard Coding for Information Interchange.**

- **Capacitat per a ser composat:** facilitar l'habilitat d'utilitzar i combinar objectes 3D dinàmics a dins d'un món *VRML* i per tant, permetre'n la reutilització.

- **Extensibilitat:** facilitar l'habilitat d'afegir nous tipus d'objectes no definits explícitament en *VRML*.

- **Capacitat per a ser implementat:** capacitat d'implementació en una extensa àrea de sistemes.

- **Funcionament:** èmfasi en el funcionament escalable i interactiu en una gran varietat de plataformes.

- **Escalabilitat:** permetre móns 3D dinàmics arbitràriament grans.

Una característica important dels fitxers *VRML* és que es pretén que siguin usats en entorns distribuits, com és ara el *World Wide Web*. Hi ha varis objectes i mecanismes dins del llenguatge que soporten múltiples fitxers distribuits, incloent:

- Inclusió d'altres fitxers *VRML* amb el mètode `Inline`.

- Hiperenllaços a altres fitxers.

- Ús de standards d'Internet i *ISO*[5] per a altres formats de fitxers (fitxers gràfics, de so, etc.)

- Definició d'una sintaxi compacta.

L'objectiu, a llarg plaç, de la incipient comunitat *VRML* que hi havia l'any 1994 era la implementació del ciberespai. Aquest objectiu es va dividir en tres àrees seqüencials:

- **Definició d'aparences:** Com es veuran les coses en el ciberespai. Això es va dur a terme mitjançant *VRML 1.0*.

- **Definició de comportaments:** Com fer que es moguin les coses. Aconseguit a *VRML 2.0.*

- **Definició de la distribució:** Com es compartiran els móns en moviment.

La complexitat s'incrementa des del primer objectiu fins a l'últim, essent més difícil l'adopció i la difusió de la tecnologia des de *VRML 1.0* a *VRML 2*.

[5] **ISO: International Standard Organization.**

2.1.1 VRML 1.0

Nascut a partir de l'interès generat sobre la creació d'un format de fitxer per a gràfics 3D que seguís un model similar al de *HTML*; és a dir, un llenguatge de descripció basat en text i independent de la plataforma on s'executés.

Silicon Graphics, gràcies a la insistència d'alguns dels seus empleats, va oferir el seu format de fitxer *Open Inventor* per a ser usat com la base de *VRML*. Aquest format fou acceptat i modificat ràpidament per assemblar-se més a la Web. També es va basar en l'ús d' *HTTP*[6] per accedir als fitxers a través d'Internet. Així doncs, *VRML 1.0* va aparèixer el 26 de Maig de 1995. De tota manera, de seguida es va veure que no era interpretat d'igual manera per tots els *browsers* (navegadors), així que el 29 de Gener de 1996 va veure la llum una versió clarificada: *VRML 1.0c*.

En aquesta primera versió, *VRML* era purament un llenguatge de descripció d'escenes 3D, al que li faltaven moltes coses. Per exemple, cap objecte de l'escena es podia moure excepte el punt de vista. En realitat no tenia massa de realitat virual. Més aviat era una imatge 2D reforçada.

Tot i que ja hi havia una base sòlida a considerar, *VRML 2.0* va ser més difícil d'aconseguir.

2.1.2 VRML 2.0 (o VRML 97)

Per als dissenyadors de *VRML 2.0*, hi va haver un conjunt de requeriments a considerar:

- Refinar *VRML 1.0* basant-se en els informes i recomanacions dels primers desenvolupadors de contingut.
- Permetre nous tipus de contingut; en particular so i altres fluxs de dades, així com animacions i altres interaccions.

VRML 2.0 es va presentar durant la fira **Siggraph '96**. Els dissenyadors són ara capaços de crear un enorme ventall d'aplicacions a través de l'ús de *VRML 2.0*. Aquest els permet crear aplicacions de model.latge químic o biomèdic per a visualitzar informació i per a disseny industrial. Ara s'ha convertit en la base d'un nou procés de disseny que es composa de:

- Els *desenvolupadors de continguts* ja han posat en marxa les noves capacitats d'animació i extensibilitat de *VRML 2.0*, comprovant si la seva semàntica és dequada per a definir móns virtuals que puguin satisfer als clients.
- Els arquitectes de *VRML* s'encarreguen d'analitzar els requeriments i articulat els inconvenients de disseny que suposa la coordinació de les interaccions entre múltiples participants humans i mecànics en escenes virtuals distribuides.

[6] **HTTP: HyperText Transfer Protocol.**

Aquest fenomen s'ha produit gràcies a algunes de les noves característiques de *VRML 2.0* respecte a l'anterior versió, com són events i enrutaments, sensors i interpoladors, prototipus i nodes *Script*. A part d'això *VRML 2.0* també és orientat a objectes. *"Pensem que el benefici de la versió 2.0 està en el que s'anomena PROTO, que són els objectes reutilitzables amb comportament"*, va dir **Anthony Parisi**, co-creador de *VRML*. *PROTO* és un mecanisme de prototipatge que inclou *VRML 2.0* i que permet als desenvolupadors crear nous nodes que estan definits en termes d'altres nodes. Aquests poden estar definits en fitxers externs i són reutilitzats en altres móns, reduint els costos de desenvolupament de *VRML*.

Les noves característiques de *VRML 2.0* respecte l'anterior versió són:

- **Events i enrutaments:** La majoria dels camps dels nodes d'una escena *VRML* poden canviar dinàmicament, però al disposar ja d'un mecanisme per permetre que aquests camps canviin, es necessita un mode d'enviar els events a aquests camps. El mecanisme *ROUTE* ha estat dissenyat per a tal, especificant un camp font i un camp destí per a un event. Usant un *ROUTE*, l'autor de l'escena pot especificar un camp *eventOut* com la font d'un event i enrutar aquest event a un camp *eventIn*. Alguns nodes *VRML* generen events en resposta als canvis de l'entorn o a la interacció amb l'usuari. L'enrutament proporciona un mecanisme a través del qual aquests events es poden propagar per efectuar canvis en altres nodes. Un cop generats, els events s'envient temporalment ordenats als seus destinataris i són processats pel node receptor. Aquest processament pot canviar l'estat del node, generar events addicionals o canviar l'estructura de l'escena gràfica.

- **Sensors i interpoladors:** Són dos tipus de nodes capaços de generar events de sortida (*eventOuts*). Els sensors segueixen la pista d'alguna entitat externa i generen events quan aquesta canvia de valor. Cada tipus de sensor defineix quan es genera un event. Després que una sèrie de sensord hagin generat events, l'estat de l'escena gràfica serà el mateix que resultaria si cava event es processa separadament, però en ordre. Si els sensors generen events al mateix temps i el resultat depèn de l'ordre dels events, l'estat de l'escena gràfica serà indefinit. És possible crear dependències entre varis tipus de sensors. Els interpoladors generen valors entre uns punts d'inici i finalització determinats, que es poden usar com a events.

- **Prototipus:** Els prototipus són un mecanisme usat per expandir el llenguatge *VRML*. Permeten l'encapsulament i parametrització d'obejctes *VRML*. Un prototipus defineix un nou tipus que es pot usar més endavant com si fos un node standard, aconseguint així una forma d'extensibilitat natural de *VRML*. Una vegada el prototipus ja està definit, es fa servir una instanciació per a crear un node d'aquell tipus, com si d'un node predefinit es tractés.

- **Nodes Script:** *VRML* té un node *Script* i Java pot ser usat com a llenguatge de *scripting* per a aquell node. El *script* Java interactua amb el món *VRML* a traves de *JSAI*[7], permetent a Java enviar events a altres nodes en el món *VRML*, crear nous components en l'escena i demanar informació sobre aquesta. El *script* Java rep events d'altres nodes i pot portar a terme algorismes o utilitzar altres

[7] **JSAI: Java Script Authoring Interface.**

packages Java. El *script* pot, llavors, enviar els resultats d'aquesta execució a altres nodes com a events. Això permet a Java proporcionar comportaments complexos als objectes d'una escena.

Un bon exemple d'això seria una escena en la que Java realitza operacions físiques complicades per controlar l'ondulació d'una bandera o el rebot d'una pilota seguint un model provat científicament. Junt amb el mecanisme de prototipatge de *VRML* es poden crear nodes amb comportaments complexos predefinits per composar altres escenes. La classe Java vindrà activada en el *JSAI* per mitjà del node oportú del món *VRML*, que indicarà la classe que s'ha d'activar en la fase de càrrega del món. A més, vindran indicats els camps que poden ser modificats des de dita classe, ja que la interacció estarà limitada a aquells camps que s'indiquin expressament en el node *Script*.

La classe Java enllaçada al món *VRML* no funciona com un applet, sinó que ve definida com a extensió de la classe *Script*. Per a poder obtenir referències als camps indicats en el node *Script* existeixen els packages que venen amb el *browser VRML*. Amb les classes en aquests paquets com a recurs és possible accedir a un camp del món *VRML*, llegir i escriure'n el valor desitjat. Per exemple, donada una referència al camp *translation* d'un node *Transform*, modificar tal valor té com a efecte imposar a tots els objectes continguts en el camp *children* la nova posició. Si això es fa a intervals regulars de temps s'obtindrà com a efecte una animació. Amb aquesta aproximació, la classe Java és activada en temps de càrrega del món *VRML*, pel que resulta impossible imposar valors a cap camp en temps d'execució en resposta a entrades de l'usuari.

2.1.2.1 External Authoring Interface (EAI)

Per a solucionar les deficiències dels nodes *Script*, neix l'*EAI*[8], una *API*[9] Java que permet controlar un món *VRML* des d'un applet extern. El *Virtual Reality Modeling Language* està format per dues parts. La primera d'elles (**ISO/IEC 14772-1**) defineix la funcionalitat base i la codificació per a *VRML*. La segona part (**ISO/IEC 14772-2**) defineix la base funcional i els límits per a la *VRML External Authoring Interface*, que és la interfície que les aplicacions externes al *browser VRML* poden utilitzar per a accedir i modificar les objectes definits en la **ISO/IEC 14772-1**.

Aquesta especificació pretén cobrir totes les formes d'accedir al *browser VRML* des de les aplicacions externes. Així doncs, és necessària una interfície perquè el món *VRML* es pugui comunicar amb l'entorn. *EAI* defineix un conjunt de funcions sobre el *browser VRML* que l'entorn pot utilitzar per actuar sobre el món *VRML*. Aquesta interfície fou proposada perquè es permeti a un programa extern (en el nostre cas, un applet) accedir als nodes d'una escena *VRML* usant el model d'events existent de *VRML*.

En aquest model, un *eventOut* d'un node donat pot ser enrutat a un *eventIn* d'un altre node. Quan l'*eventOut* genera l'event, aquest es notifica i el node el processa.

[8] **EAI: External Authoring Interface.**
[9] **API: Application Programmer's Interface.**

Addicionalment, si un *script* d'un node *Script* té un apuntador a un node donat, pot enviar events directament a qualsevol *eventIn* d'aquell node i pot també llegir l'últim valor enviat des de qualsevol dels seus *eventOuts.*

L'*External Authoring Interface* permet quatre tipus d'accessos a l'escena *VRML*:

1. Accedir a la funcionalitat del *browser script interface*.
2. Enviar events als *eventIns* de nodes de l'escena.
3. Llegir l'últim valor enviat des dels *eventOuts* dels nodes de l'escena.
4. Notificar quan s'enviin events des dels *eventOuts* de l'escena.

L'*External Authoring Interface* es va model.lar després de la *Script Authoring Interface* (interfície usada pels *scripts* dins d'un node *Script*). Els primers tres tipus d'accessos són conceptualment idèntics en amdues interfícies.

Per al primer tipus d'accés es disposa d'un objecte de tipus *Browser* a dins l'applet, que serveix per a accedir a la *Script Authoring Interface*. Per als tipus d'accés números 2 i 3 es pot obtenir un apuntador a un node a través del quales poden enviar els *eventIns* i es poden llegir els *eventOuts.*

Hi ha dues diferències conceptuals entre la *External Authoring Interface* i la *Script Authoring Interface*. La primera està relacionada amb la forma d'obtenir un apuntador a un node a través del qual s'accedeix als *eventIn* i als *eventOut*. La segona està relacionada amb el tipus d'accés número 4, que és conceptualment diferent. Això és degut a que no és possible crear un *ROUTE* entre l'escena *VRML* i l'applet. Així doncs, l'applet ha de crear un mètode que s'executi quan un *eventOut* d'un node donat sigui llençat.

Referent a la manera d'accedir als nodes, com ja sabem, aquests es poden definir en *VRML* utilitzant la clàusula *DEF*. L'applet pot accedir a qualsevol node que s'hagi creat utilitzant aquest mecanisme. Un cop creat l'apuntador a aquest node, es pot accedir també als seus *eventIns* i *eventOuts. EAI* permet als applets que estan vinculats en una pàgina *HTML* la comunicació amb l'escena *VRML* de la mateixa pàgina. Per a que un applet Java es comuniqui amb l'escena *VRML*, el primer pas és obtenir una instància de la classe *Browser*. Aquesta dasse és l'encapsulament Java de l'escena *VRML*. *EAI* connecta la màquina virtual Java que s'executa en una finestra d'un navegador Web amb el *plugin* que s'utilitza per a visualitzar el món *VRML*. A aquest últim s'hi accedeix a través d'un conjunt de classes Java definides en l'especificació d'*EAI.* El *plugin* té la funció d'actualitzar l'applet en quant a la posició i orientació dels usuaris en el món es refereix i capturar tots els events de l'escena *VRML.*

2.1.3 X3D

VRML 2.0 va entrar amb força i fou àmpliament acceptat. Van començar a aparèixer una gran quantitat de *plug-ins* per als navegadors més comuns, com ara *Cosmo Player*, que fou el més extensament utilitzat durant força temps, *ParallelGraphics Cortona*, o el que utilitzem nosaltres: *blaxxun Contact 5.0/5.1[30]*. Ara bé, el fet que *VRML* perdés el suport de *Silicon Graphics* va provocar que es perdés el suport per el standard i la innovació i a més es va ralentir molt. Això va fer que molts desenvolupadors el donessin com un llenguatge mort i sense futur.

Tot i així, l'Agost de 2001, el *Web3D Consortium* va treure l'estàndard obert ***X3D*** (*Extensible 3D*) i el va presentar com el successor de nova generació de *VRML*, per tal de "portar la riquesa dels gràfics en 3D a la web per una àmplia varietat d'aplicacions i dispositius", tal i com esmenta el seu *website* oficial.

X3D és un estàndard extensible que ha estat desenvolupat pel *Web3D Consortium* amb el suport de vàries companyies fortes en el sector, com ara **blaxxun, Nexternet, OpenWorlds** i **ParallelGraphics**.

X3D arregla les limitacions de *VRML*. Està completament especificat, de tal manera que els continguts seran compatibles. És extensible, cosa que significa que *X3D* es pot fer servir per a crear un petit i eficient reproductor d'animacions 3D, o es pot utilitzar per a suportar les últimes extensions en *streaming* o *rendering*. Soporta múltiples codificacions i *API*s, de manera que es pot integrar fàcilment amb els navegadors utilitzant *XML* o amb d'altres aplicacions.

Actualment, l'especificació *X3D* existeix però encara està sota desenvolupament. N'existeix un esborrany disponible i les diferents propostes que van realitzar les diferents companyies van ser desenvolupades i avaluades per l'*X3D Task Group*. Aquest mateix task group, en conjunció amb el *Browser Working Group* són els encarregats de la integració d'aquestes en un únic disseny coherent.

2.2 La plataforma *Java2 Enterprise Edition* (J2EE)

Cap a l'any 1996, les tecnologies *JDBC, Java Servlet* i *Java IDL* es van començar a expandir cap a una nova plataforma de servidors Java distribuïts. Els principals fabricants de software es van posar d'acord en una visió en comú: una plataforma i vàries tries. A partir d'aquest punt es va començar a desenvolupar el que avui es coneix com *J2EE*. La primera plataforma *J2EE* es va veure la llum el Desembre de 1999 i contenia l'especificació, una *suite* de tests de compatibilitat i una implementació de referència. Avui dia ja existeixen més de 25 productes compatibles amb aquesta tecnologia.

La plataforma *J2EE* soporta un model d'aplicació distribuïda multinivell basada en components escrits en Java:

- *Components client*: aplicacions de client i applets
- *Components web*: servlets i JavaServer Pages (JSP)
- *Components de negoci*: Enterprise JavaBeans (EJB)

Els components *J2EE* poden incloure components basats en JavaBeans, però aquests no son considerats part de la especificació *J2EE*.

Els objectius de la plataforma *J2EE* són definir una arquitectura de components estàndard per la construcció d'aplicacions distribuïdes basades en Java. Separar els aspectes de lògica de negoci d'altres suportats per la plataforma, com podrien ser; transaccions, aspectes de seguretat, execució de threads, pooling i d'altres elements de baix nivell. Per finalitzar, *J2EE* vol cobrir tots els aspectes de desenvolupament, desplegament i execució del cicle de vida d'una aplicació. A més manté la filosofia java d'escriure una vegada i executar en qualsevol lloc.

Una aplicació Web basada en *J2EE* té aquesta arquitectura:

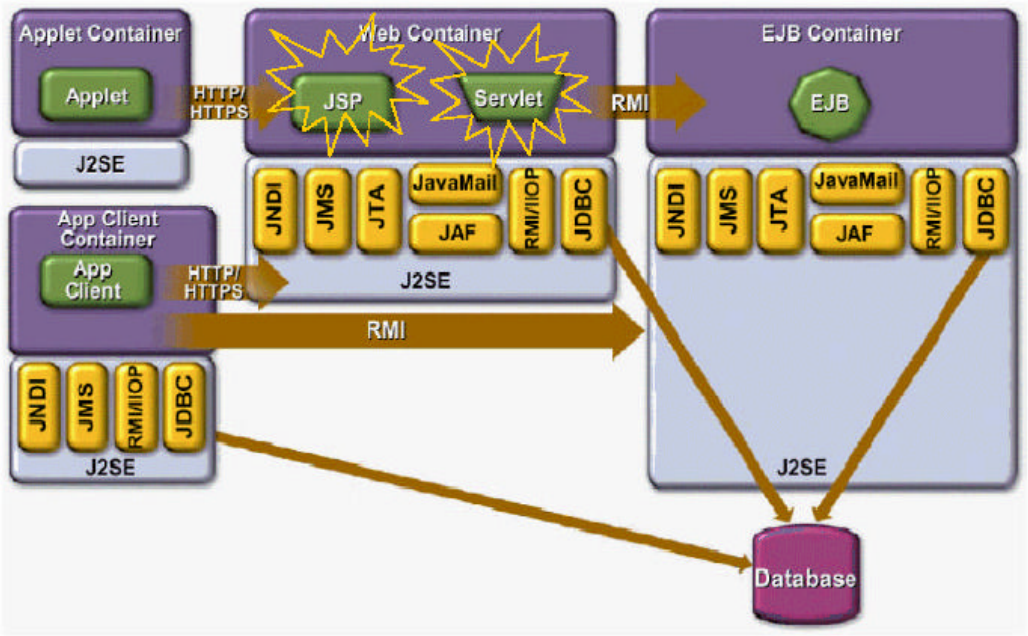


Figura 1. Arquitectura J2EE.

I a un nivell més general, aquesta:

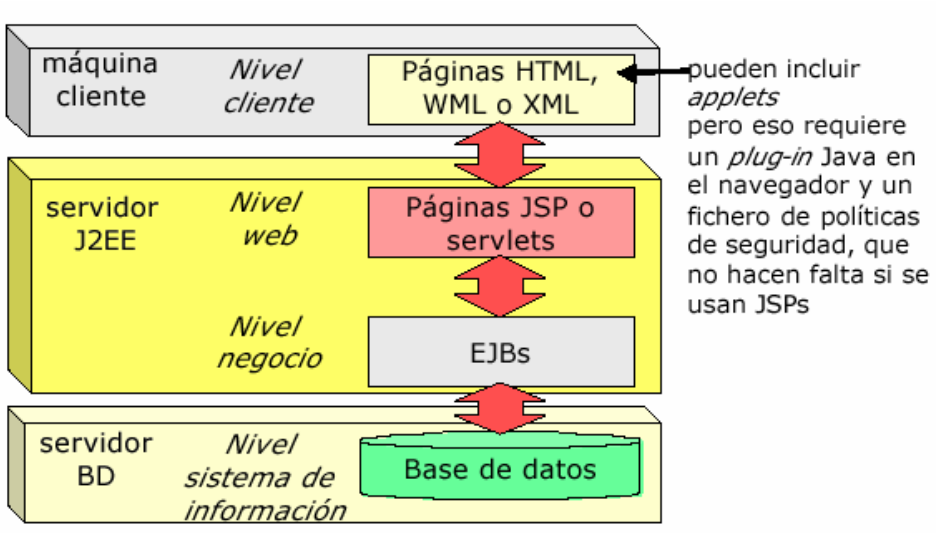


Figura 2. Generalització de l'arquitectura J2EE.

Components Web

- *Servlets*: Classes escrites en Java que processen peticions i construeixen respostes
- *Pàgines JSP*: Documents basats en text que contenen dos tipus de text: una part de dades estàtiques que pot estar escrita en format HTML, WML o XML i elements JSP (escrits en java) que determinen la construcció del contingut dinàmic

Aplicació no Web basada en J2EE

Podria donar-se el cas que vulguessim construïr una aplicació J2EE però que no estès orientada a la Web. En aquest cas l'arquitectura seria la següent:

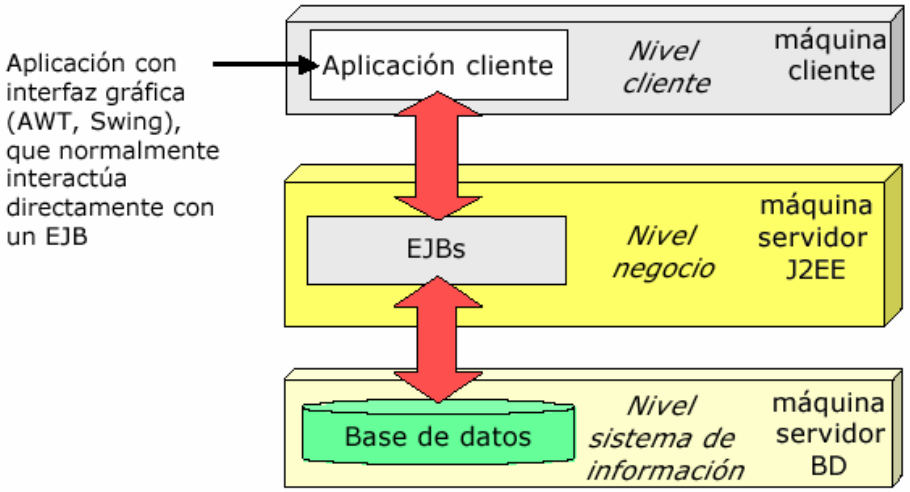


Figura 3. Estructura d'un *application client*.

Components de negoci

Els components de negoci són aquells que ens resolen les necessitats d'un determinat domini de l'aplicació.

Enterprise JavaBeans (EJBs)

- Poden processar dades rebudes del costat client i enviar-les al nivell de la base de dades per al seu emmagatzematge.
- També poden llegir dades de la base de dades, processar-les i enviar-les al client.

Existeixen tres tipus d'EJBs:

- *Bean de sessió*: conversen amb el client i amb els beans entitat.
- *Bean d'entitat*: representen les dades persistents
- *Bean dirigit per missatges*: combina las característiques d'un bean de sessió amb el Java Message Service (JMS).

Tot seguit es mostra un petit esquema del seu funcionament:

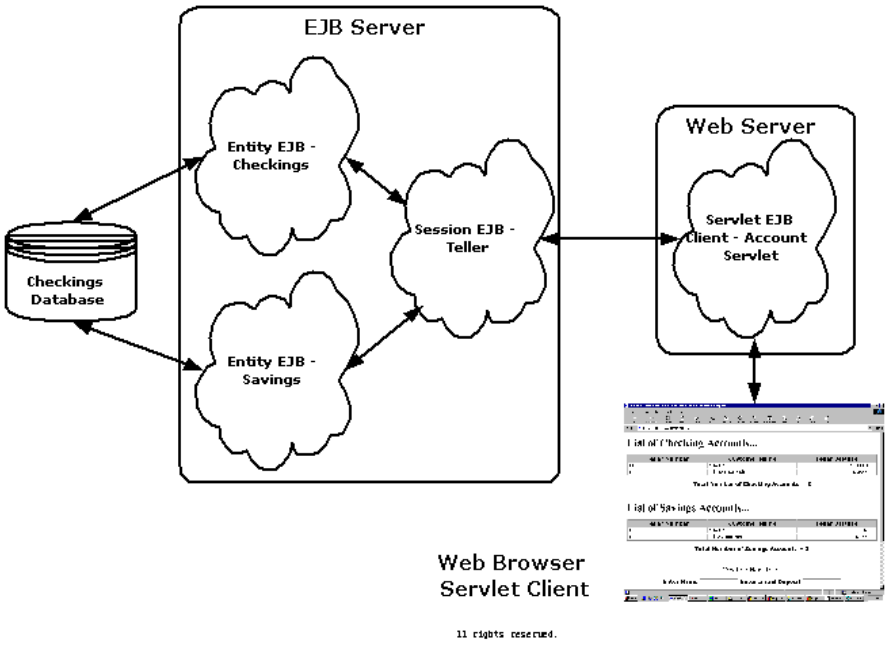
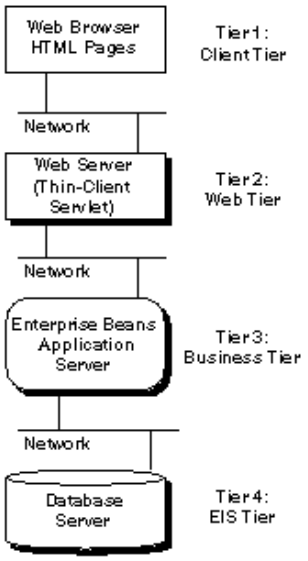


Figura 4. Esquema de funcionament dels EJBs.

Components utilitzats

Amb el que hem vist dins ara podem concloure que per a realitzar la implementació utilitzarem una arquitectura de 4 capes, els components de les quals passem a detallar a continuació:



Capa 1: Client Tier

Aquesta capa està situada en el costat del client. És el navegador web que utilitzem. Ja sigui Netscape Navigator, Internet Explorer, Opera o altres.

Capa 2: Web Tier

Aquesta capa està situada en el servidor i la composen els diferents JSP que hem realitzat. L'encarregat de compilar-los i d'executar-los serà el servidor d'aplicacions *Orion*.

Capa 3: Bussiness Tier

Aquesta capa està situada al servidor i la composen els diferents *EJB*s. Aquests estaran desplegats en el servidor d'aplicacions Orion.

Capa 4: EIS Tier

Aquesta capa la composa la base de dades. En el nostre cas hem utilitzat la base de dades *HypersonicSQL* degut a que és la que proporciona gratuïtament el servidor d'aplicacions *Orion*.

2.3 Sistemes Multiusuari

En una organització cooperativa virtual, grups d'usuaris poden ser creats per compartir i interactuar amb objectes abstractes i comunicar-se entre si a través d'aquests objectes. En un món d'aquesta classe, els usuaris són conscients de les activitats dels altres, interactuen amb objectes compartits i tenen la seva pròpia representació en 3D.

La major distinció entre els sistemes *CSCW*[10] i les aplicacions orientades a usuaris individuals és que els sistemes *CSCW* permeten que els usuaris coordinin les seves activitats entorn a un grup de tasques. La forma de coordinar-se no està clarament especificada. Els usuaris poden utilitzar protocols socials o un model, essent el primer dels dos la manera més popular de suportar la coordinació entre els usuaris, ja que permet a aquests desenvolupar mètodes de cooperació i coordinació que poden ser fets a mida per a tasques específiques.

Al llarg d'aquests anys s'han anat desenvolupant diversos sistemes multiusuari. A continuació anirem fer un cop d'ull a alguns dels més importants.

2.3.1 Distributed Interactive Virtual Environment (DIVE)

DIVE és un sistema multiusuari basat en Internet en el qual els participants naveguen en un espai 3D i es veuen, es troben i interactuen amb altres usuaris i aplicacions. El *software DIVE* és un prototipus de recerca amb un sistema de llicències. Els fitxers binaris per a un ús no comercials, no obstant, estan disponibles gratuitament per a un gran nombre de plataformes. La primera versió de *DIVE* va aparèixer l'any 1991.

DIVE està especialment optimitzat per a aplicacions multiusuari, en les quals, diversos participants remots interactuen a través d'Internet.

DIVE està basat en una aproximació punt-a-punt sense disposar d'un servidor centralitzat, on els nodes es comuniquen utilitzant la tècnica de *multicast*. La consistència i el control de la concurrència de les dades comuns (objectes) es duu a terme per replicació activa i protocols *multicast*. Això vol dir que els objectes són replicats a tots els nodes, on la rèplica manté la consistència essent contínuament actualitzada. Els missatges d'actualització són enviats mitjançant *multicast* de tal manera que tots els nodes efectuen la mateixa seqüència d'actualitzacions.

Aquesta aproximació punt-a-punt sense un servidor centralitzat significa que mentre hi hagi algun node actiu en el món, aquest, amb els seus objectes, continua existint. Com que els objectes són completament replicats a tots els nodes, són independents de qualsevol altre procés i poden existir independentment del seu creador.

Un usuari que participi en un món *DIVE* s'anomena *actor* i pot ser o bé un usuari real humà o una aplicació automatitzada (*bot*). Un *actor* és representat per una icona del seu cos (o *avatar*) per a facilitar-ne el seu reconeixement per part dels altres usuaris.

[10] **CSCW: Computer Supported Cooperative Work.**

L'usuari *veu* el món a través d'una aplicació de *rendering* anomenada *visualitzador* (el que porta per defecte s'anomena *Vishnu*). El *visualitzador* genera l'escena des del punt de vista de l'ull de l'*actor*. Canviant la posició de l'ull o canviant l'ull cap a un altre objecte farà canviar el punt de vista. El *visualitzador* es pot configurar per a suportar múltiples dispositius d'entrada/sortida, com ara un *HMD*[11], guants de dades (*datagloves*), etc. A més, llegeix les diferents entrades de l'usuari i transforma les seves accions físiques en accions lògiques per al sistema *DIVE*. Això inclou navegació en un espai 3D, fer *click* en els objectes i agafar-los.

Al ser un prototipus de recerca no hi ha previsions de que hi hagin ampliacions futures i evidentment, no es dóna suport tècnic.

2.3.2 ActiveWorlds

ActiveWorlds és una companyia establerta a Massachusetts, USA, que proporciona tant una plataforma (espai en el seu servidor) com el *software* necessari per interactuar amb altres usuaris en un entorn 3D a Internet. Qualsevol pot descarregar-se aquest *software* de la seva Web (http://www.activeworlds.com/) per navegar pels diferents móns prèviament creats. ActiveWorlds també dóna la possibilitat de llogar un espai al seu servidor perquè l'usuari crei el seu propi món virtual i pugui administrar-lo. El Setembre de 1999 es va crear un nou univers (paral.lel a l'univers principal de ActiveWorlds), anomenat *Eduverse*, l'objectiu del qual és proporcionar un espai gratuit a les institucions educatives per experimentar amb les noves possibilitats de les comunitats virtuals.

La pantalla d'ActiveWorlds es composa de quatre finestres, amb un menú principal al capdamunt, una finestra *Worlds* a l'esquerra, que mostra tots els móns existents i el nombre d'usuaris en cadascun, una finestra de *chat* al capdavall de la pantalla i una finestra *Internet* a la dreta, des d'on els *links* de dins el món virtual poden portar-nos cap a pàgines Web, i la finestra principal al mig, que mostra el món virtual.

Quan entrem per primer cop a ActiveWorlds ens hem de descarregar el *software* necessari (aproximadament 1.5MB.). Llavors ja podem entrar als diferents móns en mode *tourist*. Es pot navegar, comunicar-se i construir com a turista, però els nostres edificis no estan protegits contra altres persones. Per tant, qualsevol pot alterar o destruir el nostre treball. Per tal de rebre "drets de construcció" hem de registrar-nos com a ciutadans (*citizens*). En el cas de l'univers educatiu (*Eduverse*) això és gratuit. Una altra característica interessant de l'estat de ciutadà és que es pot usar una gran varietat d'*avatars*, mentre que en classe turista no es pot. A més, es pot crear una llista de contactes, enviar fitxers i telegrames, localitzar altres ciutadans i es pot reservar un nom que només podrà ser utlitzat per un mateix.

Els móns virtuals d'ActiveWorlds han tingut molt d'èxit i actualment n'existeixen més de 1000. La última versió de la seva aplicació és la 3.3 i fins i tot hi ha la possibilitat de descarregar-se un software (*World Server*) que permet a qui estigui interessat a hostatjar els seus propis móns en poder fer-ho.

[11] **HMD: Head Mounted Display:** Unes ulleres o un casc amb uns petits monitors al davant de cada ull que generen imatges, que donen la impressió, a qui ho porta, que està immers en un món 3D.

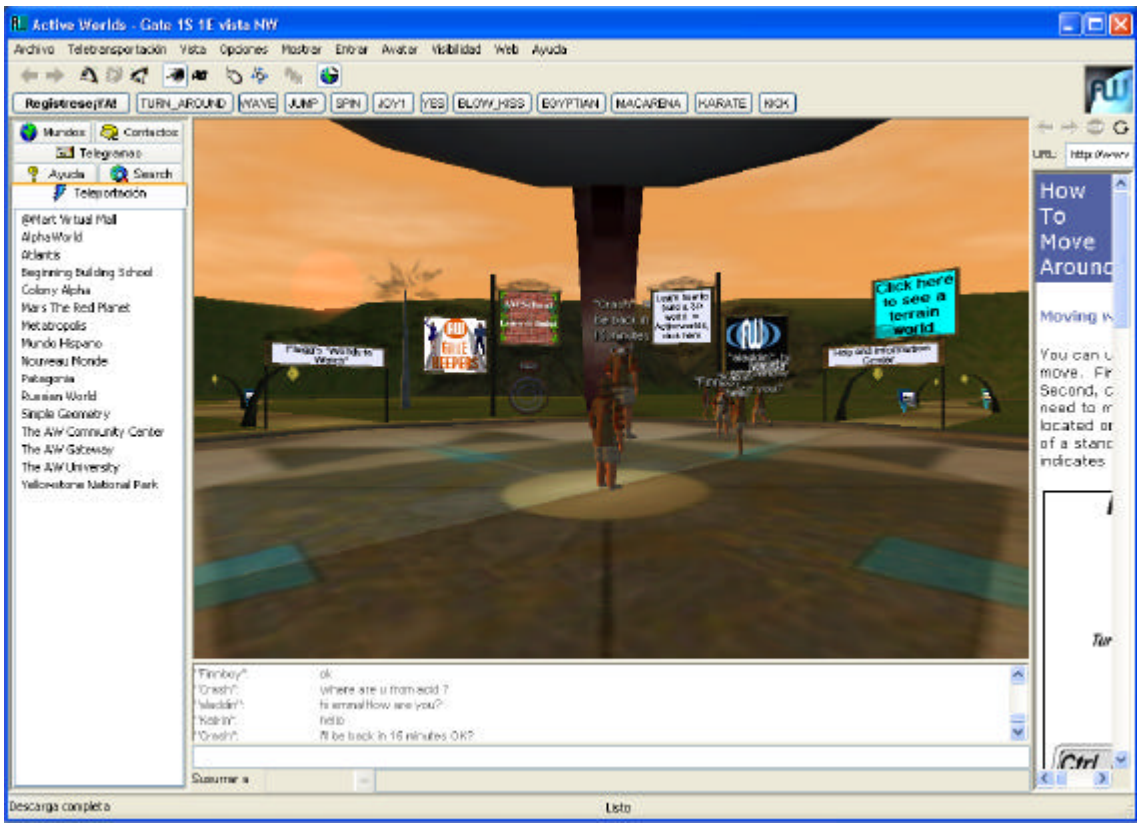


Figura 5. Aspecte de l'aplicació d'ActiveWorlds.

2.3.3 blaxxun3D

L'empresa alemanya *blaxxun Interactive* també ha desenvolupat una eina per crear i gestionar móns multiusuari en 3D. Concretament, el seu producte s'anomena *blaxxun3D* i segons ells, es pot utilitzar tant per *E-commerce*, entreteniment, animacions Web, *chat* entre diferents *avatars* i altres aplicacions. *blaxxun3D* s'integra en una pàgina Web standard a l'igual que qualsevol altre applet Java. És completament transparent a tots els visitants de la pàgina Web i es visualitza conjuntament amb els altres continguts de la pàgina.

No requereix cap descàrrega o instal.lació i es carrega automàticament quan un visitant obre la pàgina Web que conté l'applet (de 55KB.). A més, és compatible amb *VRML* i *X3D*, suportant objectes 3D i interaccions amb ells. Tampoc necessita de cap *plug-in VRML*.

Es disposa d'una versió d'avaluació gratuita completament funcional. Aquesta companyia també ha implementat un *plug-in* de *VRML* per tal de poder visualitzar els móns 3D en el navegador. El nom d'aquest *plug-in* és *blaxxun Contact* i és el que hem utilitzat nosaltres per tal de poder visualitzar correctament la part 3D de MOVE. Per a més informació es pot visitar la seva pàgina Web: http://www.blaxxun.com

2.4 Realitat Virtual

Es presenta una nova forma de realitat virtual que es basa en el PC com una aplicació normal i que es coneix com Entorn Visual, el qual pretén explotar la tridimensionalitat i la interactivitat dels móns virtuals més que no pas el fet de ser immersius. És llavors quan es parla de Realitat Virtual d'"escriptori" o *Desktop VR*. Un cas concret d'aquest nou tipus de Realitat Virtual és el nostre sistema, en el qual es dóna cabuda a totes les tecnologies necessàries per a aquest tipus de desenvolupaments.

El dispositiu de visualització més senzill és la pantalla normal. Sense ser massa exigent es poden reconèixer com a Realitat Virtual aplicacions que despleguen informació a l'usuari a través de la pantalla de l'ordinador. En aquests casos es parla de *Realitat Virtual de finestra*, ja que la pantalla es converteix en una finestra a través de la qual es veu el món virtual. La sensació de realisme s'incrementa notablement quan no sols es veu el món, sinó que se'n poden escoltar sorolls adequats al que està passant. Aparentment, el número d'estímuls visuals que arriben al nostre cervell en cada unitat de temps és més gran que el nombre d'estímuls originats en els altres sentits combinats. L'home és un ser doncs, eminentment visual.

La Realitat Virtual, doncs, s'imposa com a una nova interfície home-màquina que permet una forma més natural d'interacció.

En quant a dispositius de percepció sensorial, els *HMD* (*Head Mounted Display*) foren els primers dispositius en proporcionar una experiència immersiva a l'usuari. Un dispositiu *HMD* típic conté dues pantalles en miniatura i un sistema òptic que canalitza les imatges des de les pantalles fins als ulls, presentant una visió estereoscòpica d'un món virtual. Un sensor de moviment mesura contínuament la posició i orientació del cap de l'usuari i permet a l'ordinador que genera la imatge ajustar la representació de l'escena a la vista actual. Com a resultat, l'usuari pot mirar a tot arreu i "caminar" a través de l'entorn virtual que l'envolta.

Una gran varietat de dispositius d'entrada, com ara *datagloves*, *joysticks*, etc. permeten a l'usuari navegar a través d'un entorn virtual i navegar amb objectes virtuals a través d'un entorn virtual i interactuar amb objectes virtuals. So direccional, dispositius *force-feedback*[12], reconeixement de veu i altres tecnologies estan essent utilitzades per enriquir l'experiència immersiva i per crear unes interfícies més "humanes".

[12] Un dispositiu *force-feedback* és un dispositiu que reacciona segons la força que li aplica l'usuari, oferint més o menys resistència segons convingui.

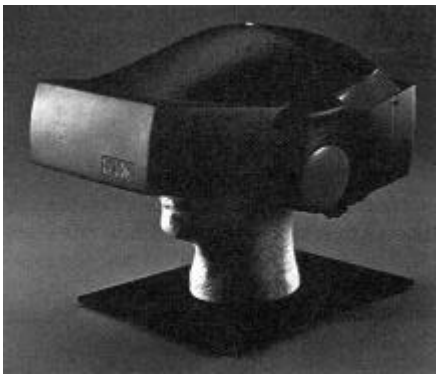


Figura 6. Detall d'un *Head Mounted Display* (esquerra) i un *dataglove* (dreta).

De fet, les característiques d'una realitat virtual immersiva poden resumir-se a continuació:

- La visualització utilitzant el cap de l'usuari com a referència proporciona una interfície natural per a la navegació en un espai tridimensional i proporciona les capacitats de mirar, caminar i fins i tot poder volar pels voltants de l'entorn virtual.

- La visió estereoscòpica amplia la percepció de profunditat i el sentit de l'espai.

- El món virtual és presentat a escala completa i es comporta adequant-se a l'alçada d'un ésser humà real.

- La interacció realista amb els objectes virtuals a través dels *datagloves* i dispositius similars permet la manipulació, operació i control dels móns virtuals.

- La il.lusió d'estar completament immersos en un món artificial pot ser potenciada mitjançant l'ús de so direccional, espacial, o altres tecnologies no-visuals.

2.4.1 Cave Automatic Virtual Environment (CAVE)

CAVE, mostrat en la figura següent, va ser desenvolupat a la Universitat de Illinois a Chicago, i proporciona la il.lusió d'immersió projectant diverses imatges estereoscòpiques en les parets i el terra d'una habitació en forma de cub. Diverses persones que porten ulleres estereoscòpiques lleugeres poden entrar i caminar lliurement per dins de *CAVE*. Un sistema de sensors al cap ajusta contínuament la projecció estereoscòpica a la posició actual de l'usuari visualitzador principal.

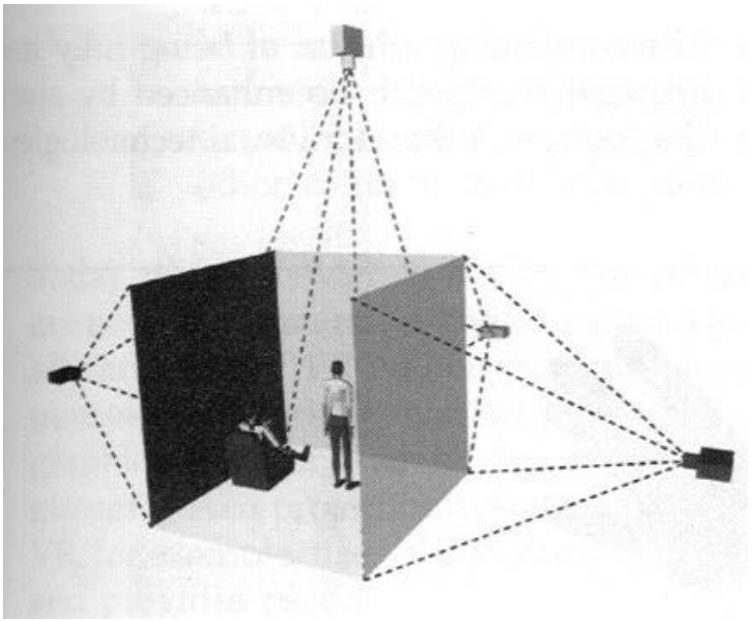


Figura 7. El sistema *CAVE*.

3. Especificació i requisits

Tal i com hem comentat anteriorment, l'arquitectura *J2EE* ens proporciona tota la infrastructura necessària per a la creació de la nostra aplicació web. Per tal de no perdre de vista tots els components de la nostra aplicació, a continuació presentem un esquema de les seves principals parts. S'ha de tenir en compte que en tractar-se d'una aplicació multiusuari, poden existir molts clients connectats alhora:

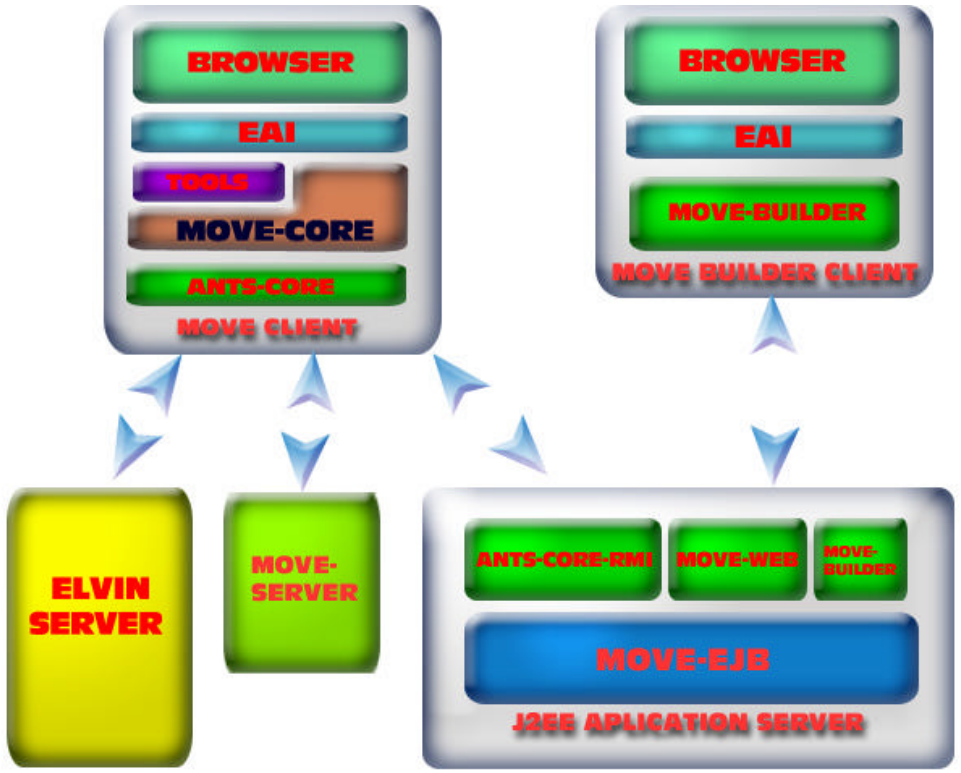


Figura 8. Arquitectura general de MOVE.

Per tal de mantenir la consistència entre els estats dels diferents clients s'utilitza el *framework* ANTS CSCW que ens proporciona un tractament senzill de les propietats de les diferents eines que proporciona el sistema. *Elvin server* és l'encarregat de retransmetre els events que es produeixen al sistema a tots els clients que hi estiguin interessats. *Elvin* és un servei de notificació desenvolupat que podem descarregar i veure'n les seves característiques a http://elvin.dstc.edu.au. El component *move-server* és un *proxy* encarregat de filtrar els events de moviments dels avatars per tal de millorar el rendiment del sistema. En la part del client ens trobem tots els components necessaris per interactuar amb l'usuari. Degut a que l'entorn sera 3D utilitzarem un *plug-in* VRML que serà, com ja hem comentat anteriorment, *blaxxun Contact 5.0*.

4. Fase d'anàlisi

Els diagrames de casos d'ús mostren les diferents operacions que s'esperen d'una aplicació. Per representar-los s'ha utilitzat el *Unified Modeling Language* (UML).

4.1 Diagrames de casos d'ús de l'aplicació web

A continuació es presenta el diagrama de casos d'ús de l'aplicació web, que contempla totes les accions que poden realitzar els diferents rols d'usuaris del sistema. Els rols existents són *administrador*, *teacher* i *student*. Hem agrupat en un de sol els de *teacher* i *student* per claredat del diagrama.

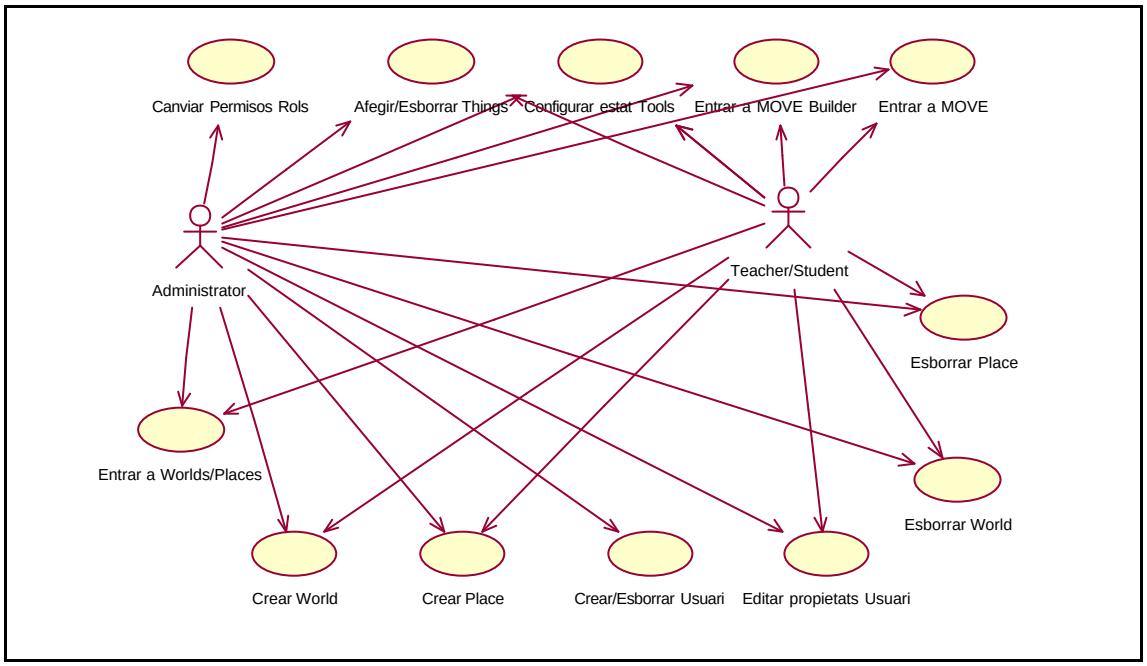


Figura 9. Diagrama de casos d'ús de l'aplicació web.

Passem a veure quines són doncs, les possibles operacions que pot realitzar cadascun d'aquests rols:

- **Administrador:** L'administrador podrà entrar a *worlds* i *places*, crear i esborrar *worlds*, crear i esborrar *places*, editar les propietats dels usuaris, afegir i esborrar *things*, configurar l'estat de les *tools*, entrar tant a MOVE Builder com MOVE i les seves opcions específiques, que són canviar els permisos dels rols (pot decidir quines *tools* estaran activades i desactivades per als rols *teacher* i *student*) i crear i esborrar usuaris.

- **Teachers / Students:** En realitat, a la part web de l'aplicació, els *students* i *teachers* poden realitzar les mateixes funcions. Menció apart mereix l'esborrat de *worlds* i *places*: només podran esborrar-los si n'han sigut els creadors; és a dir, si un usuari d'aquests rols crea un *world*/*place*, només podrà esborrar-lo ell mateix i ningú més. Per tant, d'acord amb el diagrama de casos d'ús anterior,

podran entrar a *worlds/places*, crear *worlds/places*, esborrar *worlds/places* que hagin creat ells, editar les seves propietats d'usuari, afegir i esborrar *things*, que estiguin activades (si ho estan o no ho decideix l'*administrador*), configurar l'estat de les *tools* i entrar tant a MOVE Builder com MOVE.

4.2 Diagrames de casos d'ús de MOVE BUILDER

Un cop s'és dins de MOVE BUILDER, el rol de l'usuari ja no importa, de manera que qualsevol usuari de qualsevol rol sempre pot efectuar les mateixes accions, que es poden veure a continuació:

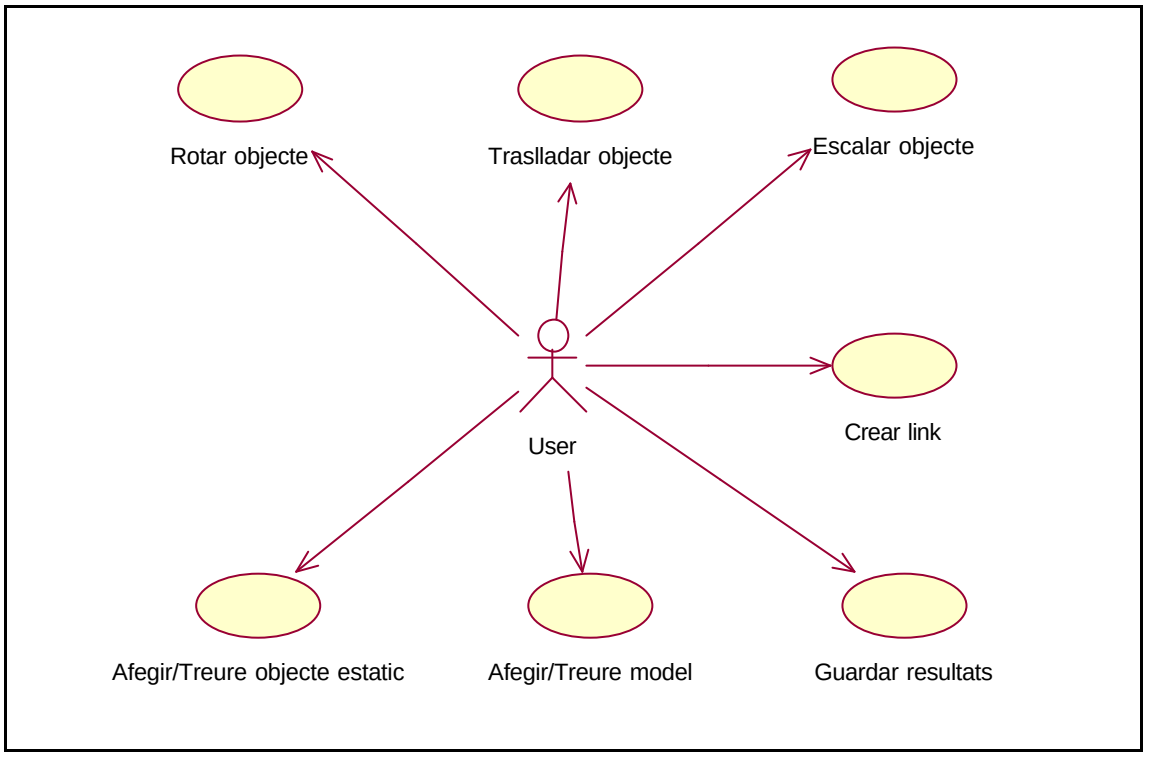


Figura 10. Diagrama de casos d'ús de MOVE BUILDER.

Anem a comentar en què consisteix cadascuna d'aquestes accions:

- **Rotar, traslladar i escalar objecte:** Permeten rotar, traslladar o bé escalar l'objecte actualment seleccionat.
- **Crear link:** Permete crear un enllaç des del *place* actual cap a un altre *place*, del mateix *world*, o de diferent.
- **Afegir/Treure objecte estàtic:** Permet afegir o treure un objecte decoratiu de la representació tridimensional del *place*.
- **Afegir/Treure model:** Permet afegir o treure un model o artefacte compartit de la representació tridimensional del *place.*

- **Guardar resultats:** Emmagatzema els resultats al fitxer **.wrl* corresponent i actualitza la base de dades amb els canvis realitzats.

4.3 Diagrames de casos d'ús de MOVE

El diagrama de casos d'ús de l'entorn multiusuari tridimensional MOVE és el que es mostra a continuació. S'ha de tenir en compte que dins d'aquest entorn, només tenen valor els rols de *student* i *teacher*. Si un usuari de tipus *administrador* entra a dins l'entorn, hereda el rol de *teacher*.

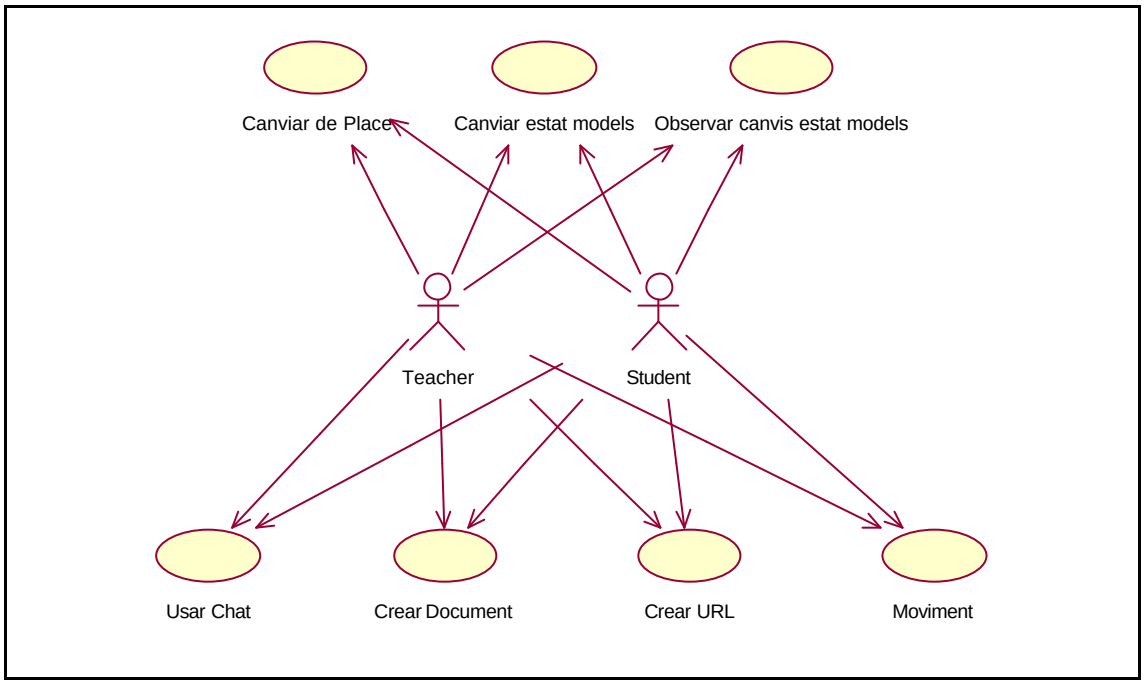


Figura 11. Diagrama de casos d'ús de MOVE.

Tal i com es pot observar el rol *teacher* pot realitzar tots els tipus d'operacions, mentre que el rol *student* també; això sí, mentre el rol posseeixi el permís per fer-ho. Passem a comentar breument les funcionalitats de les operacions esmentades:

- **Usar Chat:** Tant *students* com *teachers* poden fer ús de l'eina de chat que surt per defecte al panell de control de MOVE.

- **Crear Document i URL:** Si aquestes *tools* estan actives, es permet la creació d'instàncies d'aquestes a dins del *place*. Ens sortirà una finestreta que ens demanarà la localització del document/URL i una descripció. A continuació podrem deixar el document o enllaç al lloc que preferim del *place*.

- **Moviment:** Els usuaris es poden moure per tota la representació tridimensional del *place*.

- **Canviar de place:** En el cas que existeixin objectes de tipus *link*, aquests es poden utilitzar per canviar de *place*.

- **Canviar estat models:** Tant els *teachers* com els *students* podran canviar l'estat dels models, de manera que podran, per exemple, avançar transparències, reproduir videos, etc. Sempre i quan el seu rol tingui permís per fer-ho.

- **Visualitzar canvis d'estat dels models:** Els canvis d'estat es poden visualitzar observant la representació 3D del model.

5. Fase de disseny i implementació

En aquest apartat veurem la fase de disseny dels mòduls que composen la nostra aplicació.

5.1 El mòdul *move-ejb*

El component encarregat de la persistència de l'entorn serà el component *EJB*. Aquest component està format per diferents *session beans* i *entity beans*. Els *entity beans* seran BMPs (*Bean Managed Persistence*) i guardaran les seves propietats en una base de dades relacional. Aquesta base de dades serà la que porta per defecte el servidor d'aplicacions *Orion 1.5.2* i s'anomena *HypersonicSQL*. A més, disposarem de diferents *session beans* encarregats de la lògica del servidor. Seran *stateless session beans* per tal de millorar el rendiment ja que no han de guardar cap informació d'estat. A continuació tenim el diagrama de classes en UML del package *ants.move.ejb*:

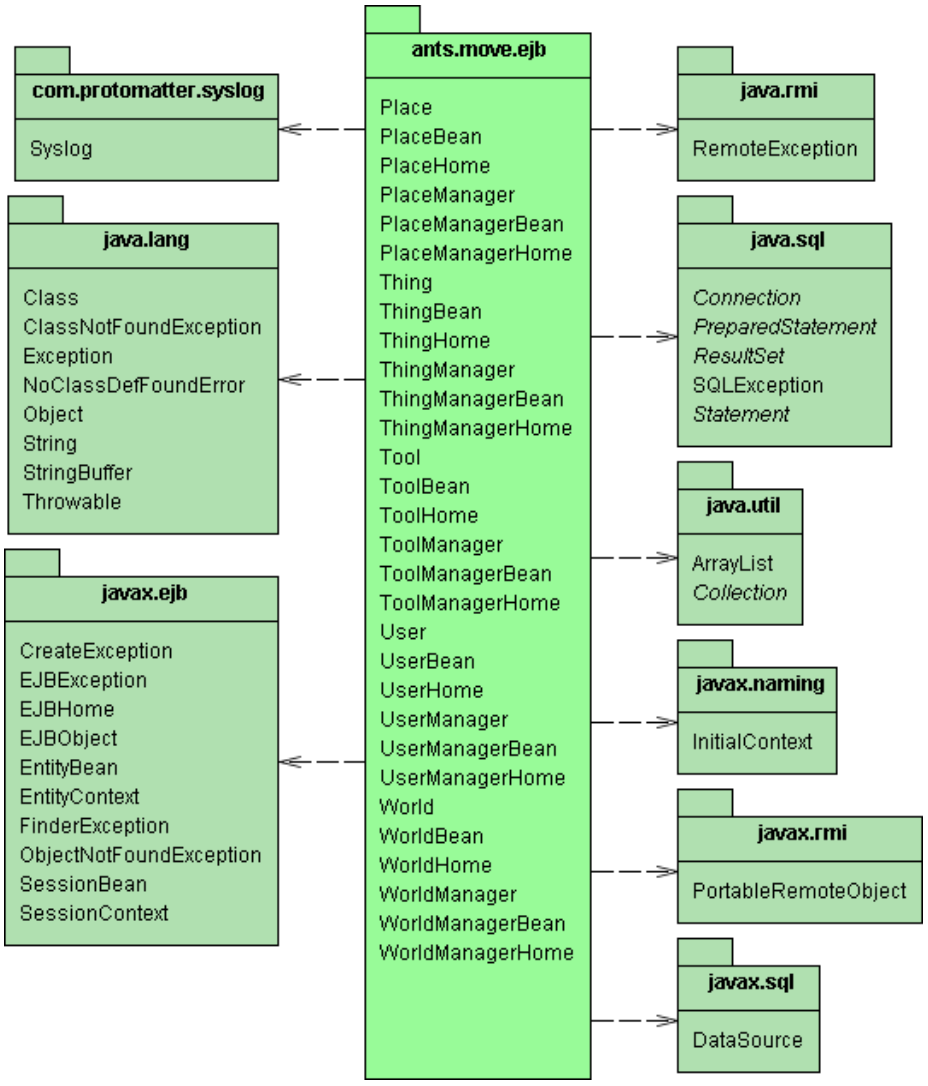


Figura 12. Diagrama de classes del package *ants.move.ejb*.

5.1.1 Diagrama Entitat-Relació

S'han hagut de crear una sèrie de taules a la base de dades per tal de poder emmagatzemar tota la informació. A continuació podem veure el diagrama entitat-relació:

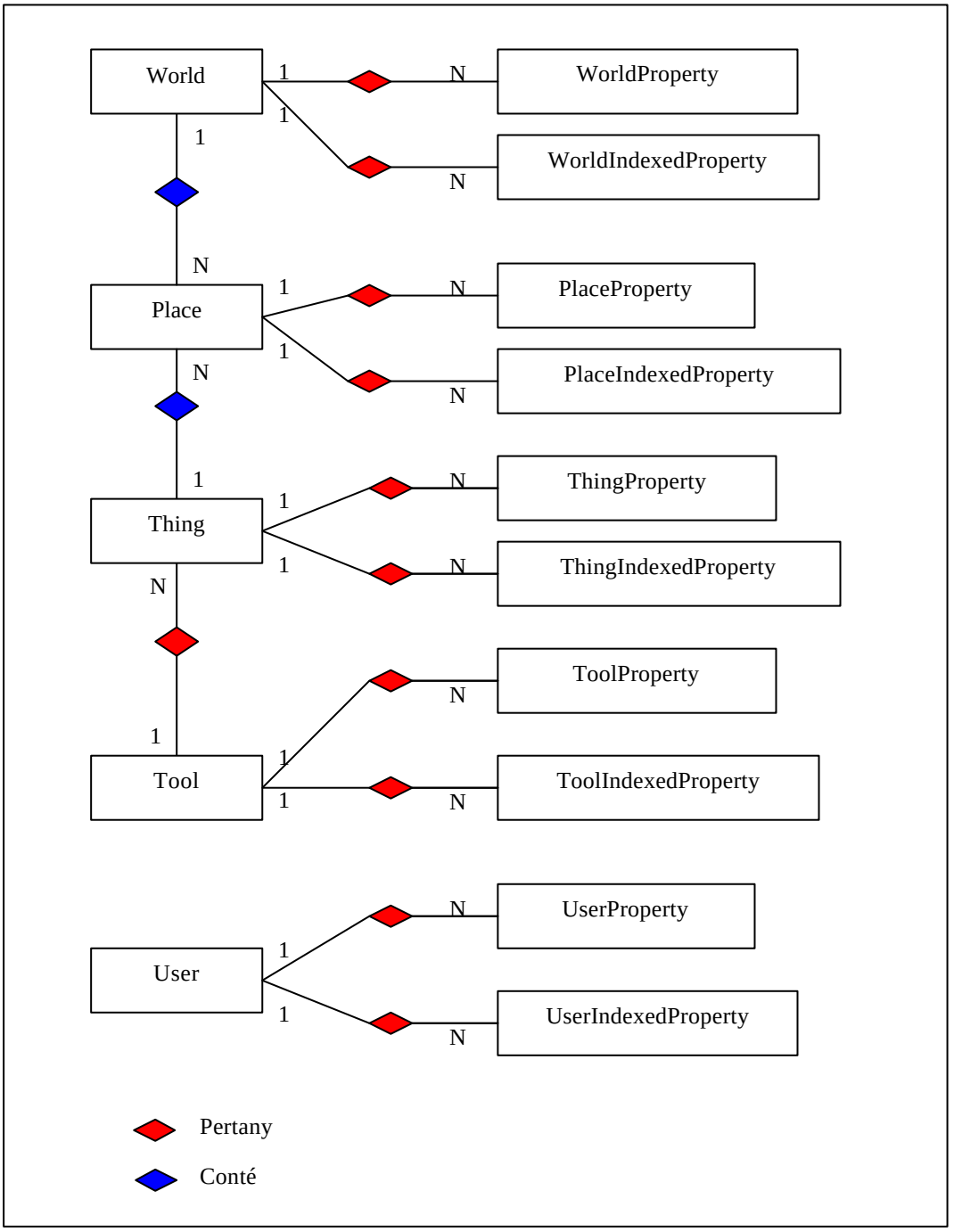


Figura 13. Diagrama entitat-relació.

Seguidament, observarem les diferents taules i els atributs que s'han guardat en totes elles:

World	WorldProperty	WorldIndexedProperty
Id	SubId	SubId
Name	Name	Name
Type	Value	Index
Description		Value

Place	PlaceProperty	PlaceIndexedProperty
Id	SubId	SubId
Name	Name	Name
Type	Value	Index
Description		Value
WorldId		

Thing	ThingProperty	ThingIndexedProperty
Id	SubId	SubId
Name	Name	Name
ToolId	Value	Index
Description		Value
PlaceId		

Tool	ToolProperty	ToolIndexedProperty
Id	SubId	SubId
Name	Name	Name
Type	Value	Index
Description		Value

User	UserProperty	UserIndexedProperty
Login	SubId	SubId
Password	Name	Name
Role	Value	Index
		Value

Disposem d'un *EJB* de tipus *Entity* per a cada taula de la base de dades. Aquests *EJB*s seràn *BMP*s (Bean Managed Persistence), per tal de facilitar la interacció amb la base de dades i millorar-ne el rendiment. L'entorn de treball serà el següent: tenim una realitat superior que anomenem World, aquests Worlds tenen un paràmetre associat anomenat *Type* que ens indica de quin tipus de World es tracta. Aquest paràmetre només és informatiu i no ens condueix a cap restricció. Dins d'un World poden existir diferents Places, aquests també disposen del paràmetre informatiu *Type* . Per exemple el tipus d'un World podria ser "Universitat" i el d'un Place "Facultat". Dins d'un place no pot haver-hi cap altre place. Aquesta restricció s'ha posat per tal de no complicar l'arquitectura del sistema.

Les Tools són eines instal·lades al sistema que interactuen amb les Things. És per això que una Tool està instal·lada a tot el sistema i no pertany a cap World ni Place en concret. Les Tools són visibles a tot el sistema. Disposen d'un camp anomenat *Type* que

ens indica amb quin tipus de Thing interactua. Només interactua amb un tipus de Thing determinat.

Disposem d'entitats Thing. Aquestes pertanyen a un Place determinat per la *PlaceId* i a més interactuen amb un tipus de Tool determinat per *ToolId*.

Un exemple il·lustratiu d'aquest entorn podria ser: en un World anomenat *Empresa*, disposem d'un Place anomenat *Taller* i dins d'aquest Taller disposem de Things de tipus *clau* i de tipus *cargol*. A més en el sistema tenim instal·lada la Tool *martell* que interactua amb Things de tipus *clau* i la Tool *tornavís* que interactua amb Things de tipus *cargol*.

Per tal de mantenir els usuaris disposem d'una taula a la base de dades (User) que conté el *login*, *password* i el *rol* de l'usuari en el sistema.

Totes les entitats anteriors per tal de millorar-ne la extensibilitat podran tenir diferents propietats (Property) o propietats indexades (IndexedProperty).

5.1.2 Diagrames de classes de l'entity bean *World*

Disposem d'un *entity bean* per l'entitat world. Aquest *entity bean* serà l'encarregat de guardar les dades corresponents als diferents *Worlds* dels que disposem, així com de mapejar aquestes dades a la nostra base de dades relacional.

La interfície *home* és *ants.move.ejb.WorldHome* i disposa dels mètodes bàsics dels que consta una interfície *home*: el mètode de creació, un mètode per obtenir tots els elements i un mètode per localitzar un objecte mitjançant la seva clau primària.

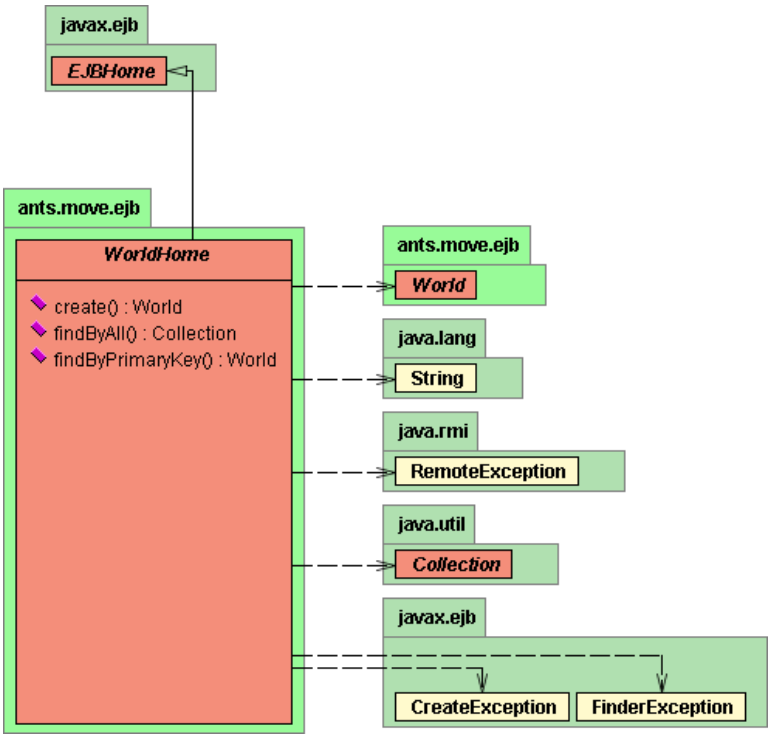


Figura 14. Diagrama de classes de la classe *ants.move.ejb.WorldHome* .

La classe de la *interface* és *ants.move.ejb.World* i ens proporciona mètodes per obtenir dades (*getter methods*) i mètodes per fixar dades (*setter methods*).

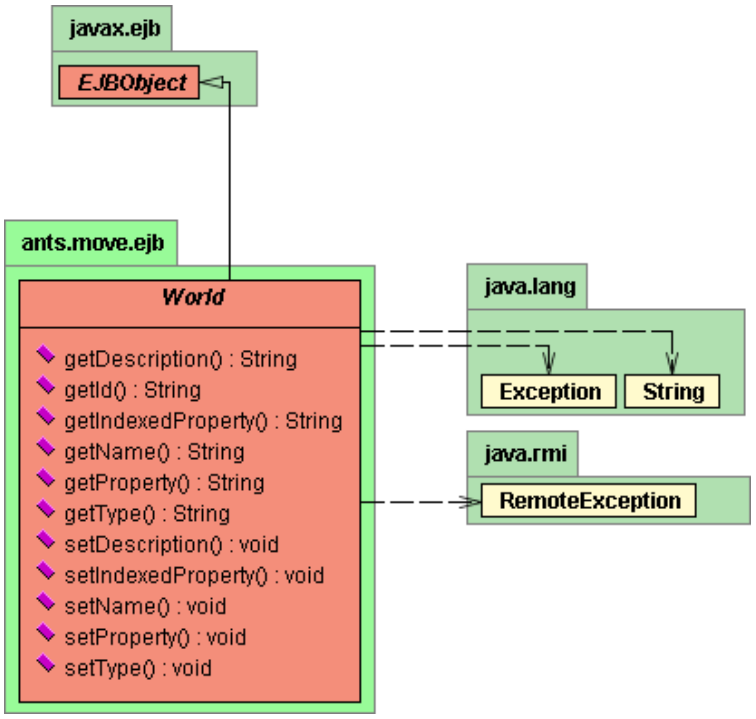


Figura 15. Diagrama de classes de la classe *ants.move.ejb.World*.

La classe que implementa l'*entity bean* World és *ants.move.ejb.WorldBean* i implementa totes les operacions descrites en la interfície anterior, així com s'encarrega del mapeig de les dades a la base de dades.

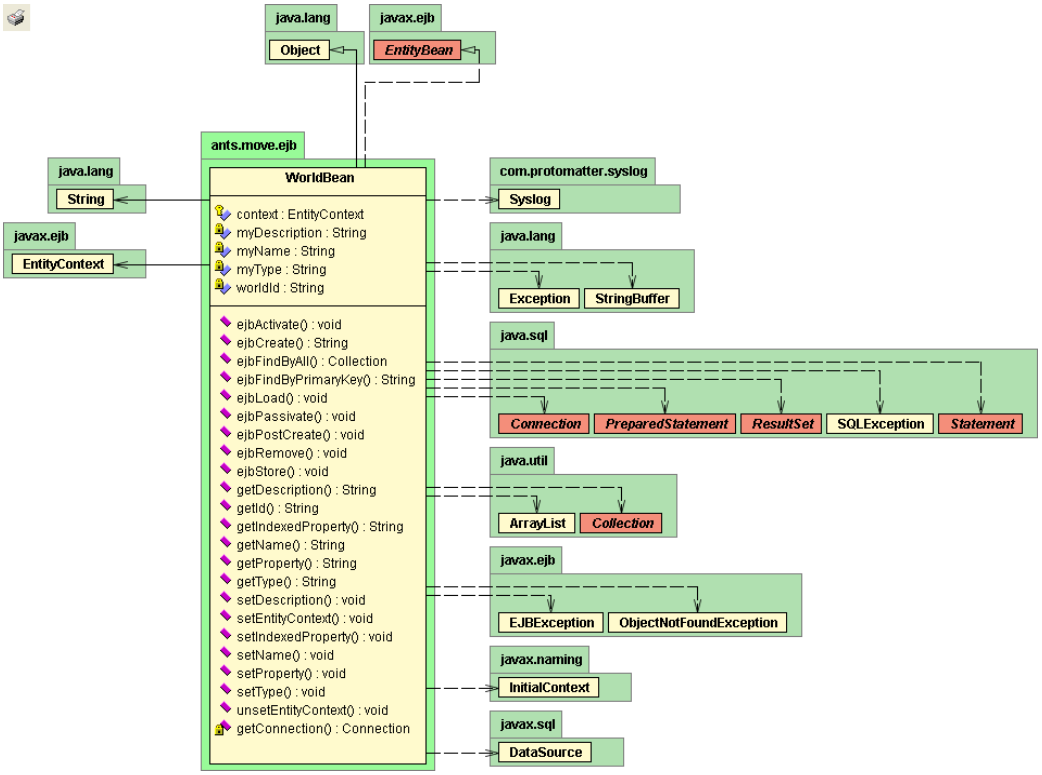


Figura 16. Diagrama de classes de la classe *ants.move.ejb.WorldBean*.

5.1.3 Diagrames de classes de l'entity bean *Place*

Disposem d'un *entity bean* per l'entitat place.

La interfície *home* és *ants.move.ejb.PlaceHome.* Proporciona, a més, un mètode per trobar el places que hi ha en un world: *findByWorld()*.

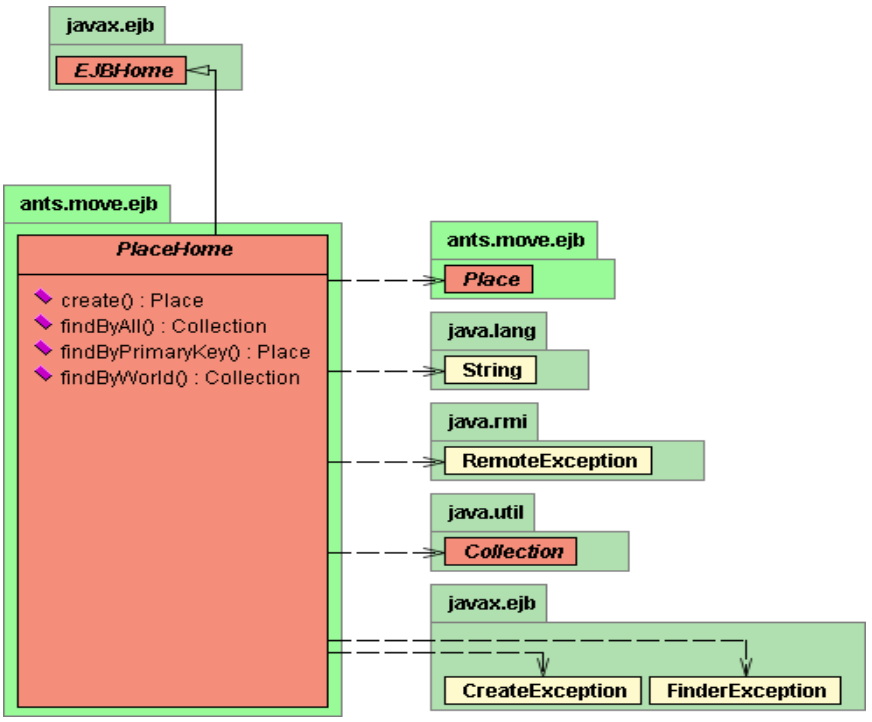


Figura 17. Diagrama de classes de la classe *ants.move.ejb.PlaceHome*.

La classe de la *interface* és *ants.move.ejb.Place*:

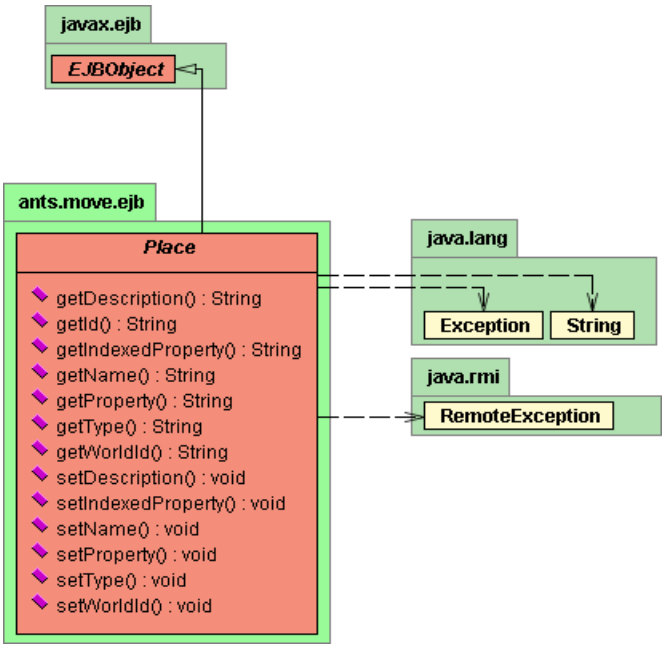


Figura 18. Diagrama de classes de la classe *ants.move.ejb.Place*.

La classe que implementa l'*entity bean* Place és *ants.move.ejb.PlaceBean*:

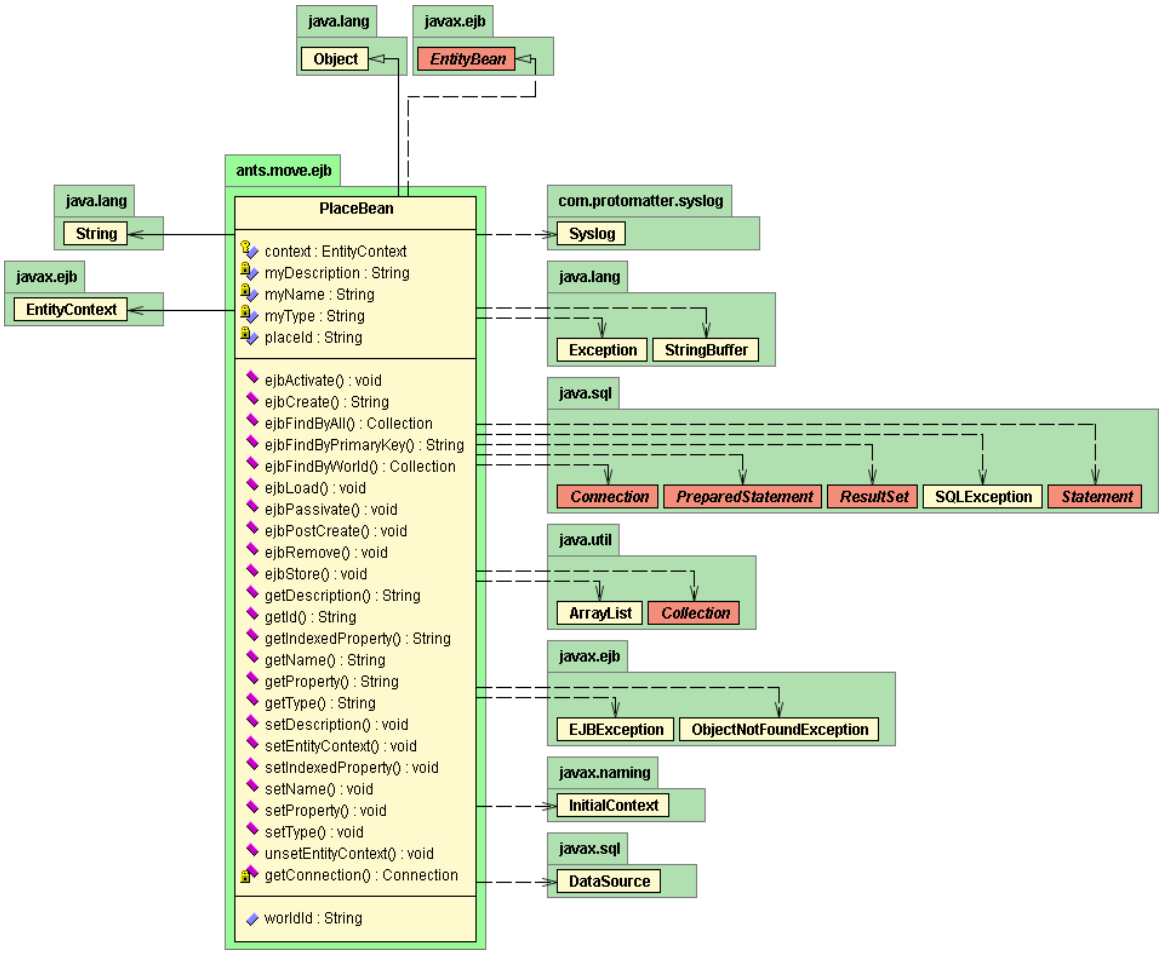


Figura 19. Diagrama de classes de la classe *ants.move.ejb.PlaceBean*.

5.1.4 Diagrames de classes de l'entity bean *Thing*

Disposem d'un *entity bean* per l'entitat thing.

La interfície *home* és *ants.move.ejb.ThingHome*. Proporciona un mètode per trobar les things que hi hagi en un place i les que són d'un tipus determinat de tool

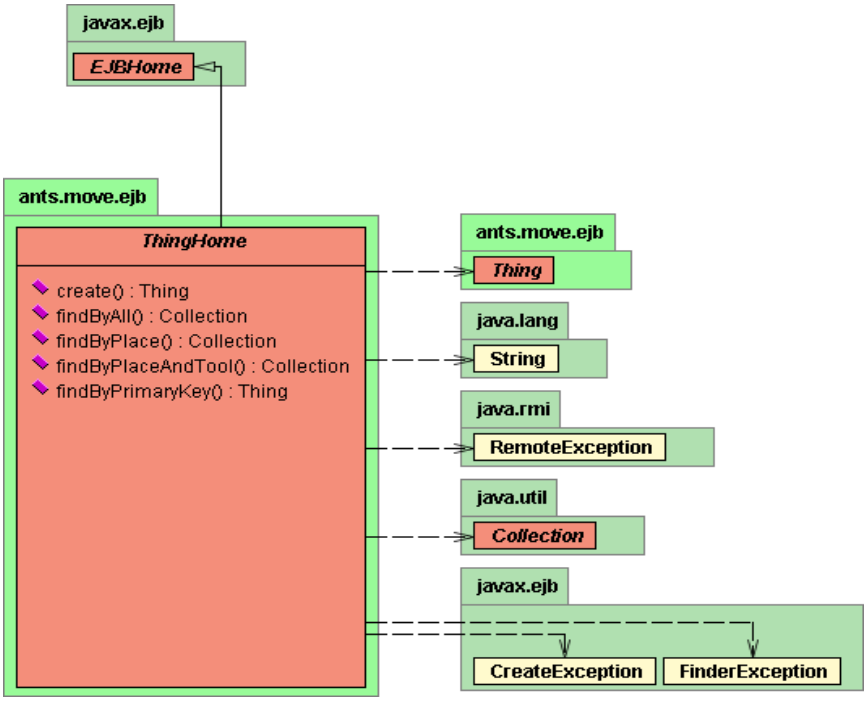


Figura 20. Diagrama de classes de la classe *ants.move.ejb.ThingHome*.

La classe de la *interface* és *ants.move.ejb.Thing*:

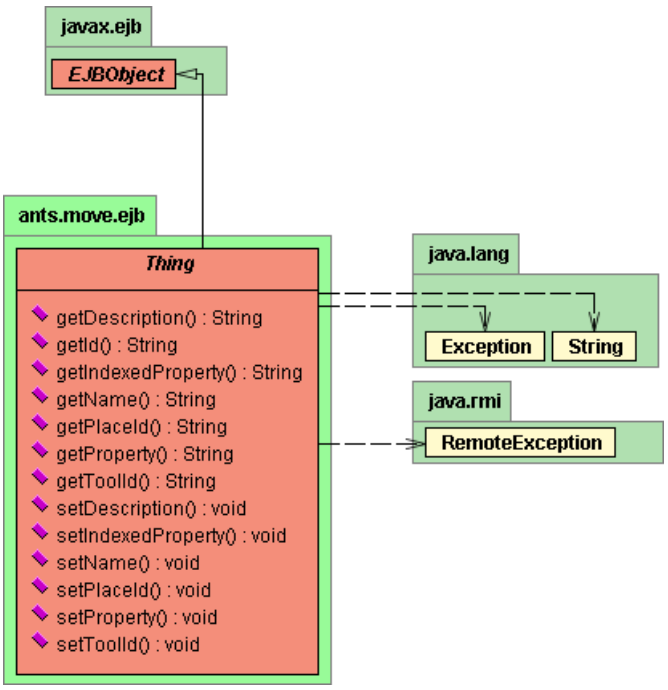


Figura 21. Diagrama de classes de la classe *ants.move.ejb.ThingHome*.

La classe que implementa l'*entity bean* Thing és *ants.move.ejb.ThingBean*:

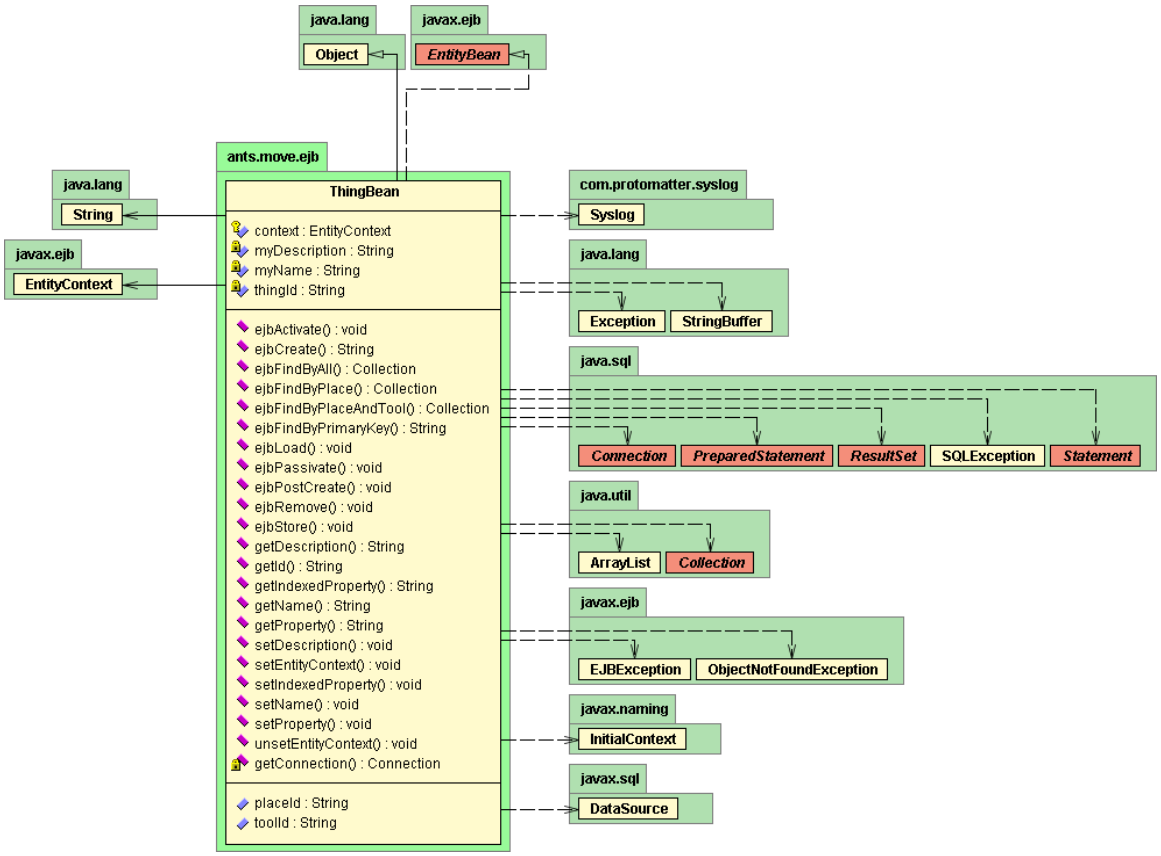


Figura 22. Diagrama de classes de la classe *ants.move.ejb.ThingBean*.

5.1.5 Diagrames de classes de l'entity bean *Tool*

Disposem d'un *entity bean* per l'entitat tool.

La interfície *home* és *ants.move.ejb.ToolHome*:

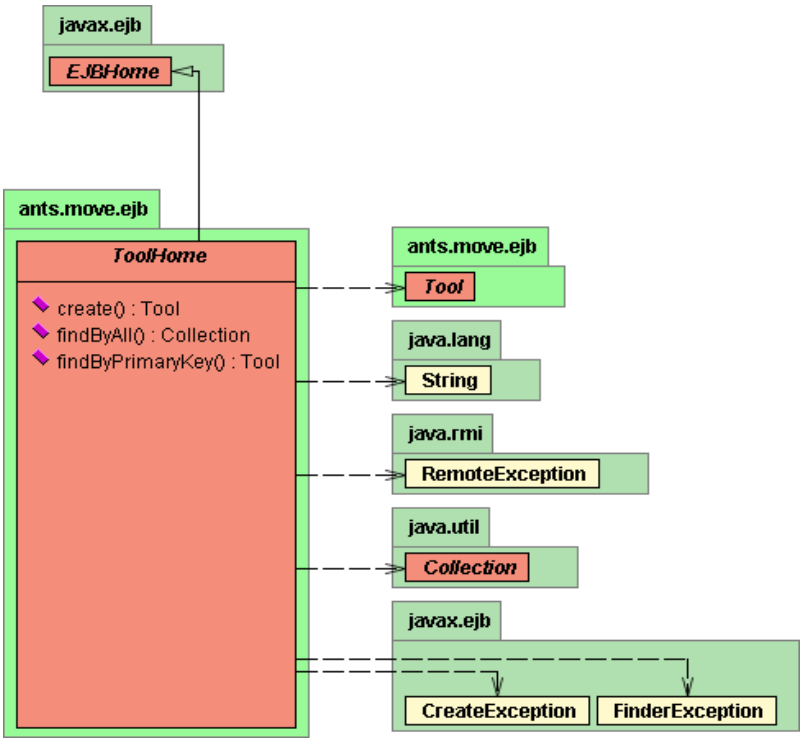


Figura 23. Diagrama de classes de la classe *ants.move.ejb.ToolHome*.

La classe de la *interface* és *ants.move.ejb.Tool*:

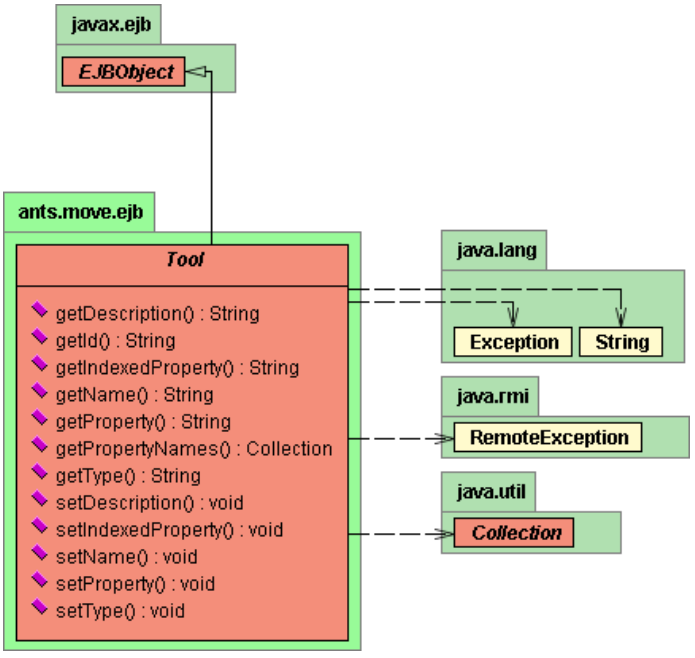


Figura 24. Diagrama de classes de la classe *ants.move.ejb.ToolHome*.

La classe que implementa l'*entity bean* Tool és *ants.move.ejb.ToolBean*:

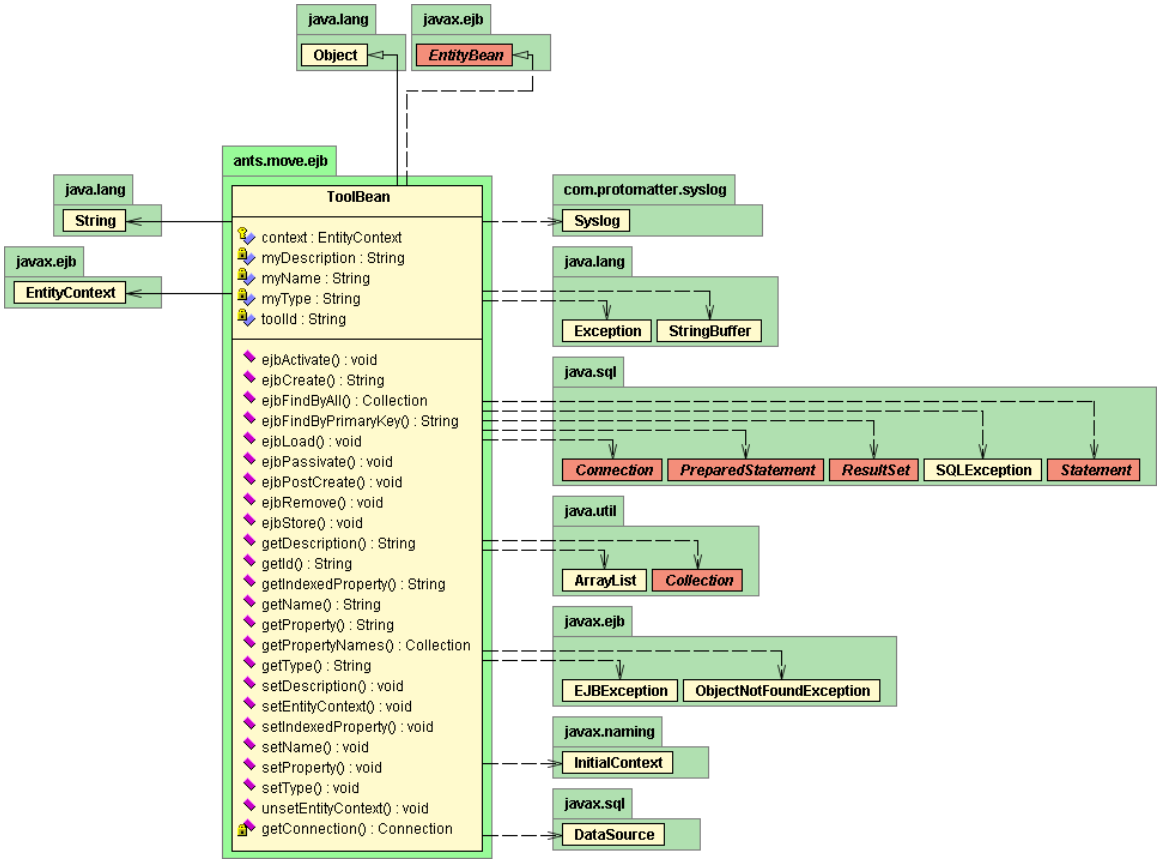


Figura 25. Diagrama de classes de la classe *ants.move.ejb.ToolBean*.

5.1.6 Diagrames de classes de l'entity bean *User*

Disposem d'un *entity bean* per l'entitat user.

La interfície *home* és *ants.move.ejb.UserHome*:

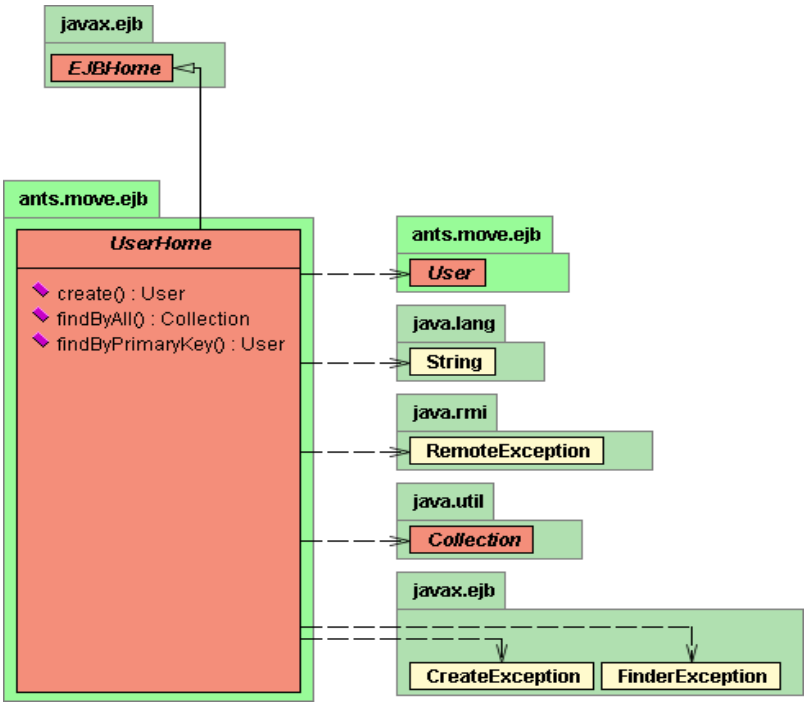


Figura 26. Diagrama de classes de la classe *ants.move.ejb.UserHome*.

La classe de la *interface* és *ants.move.ejb.User*:

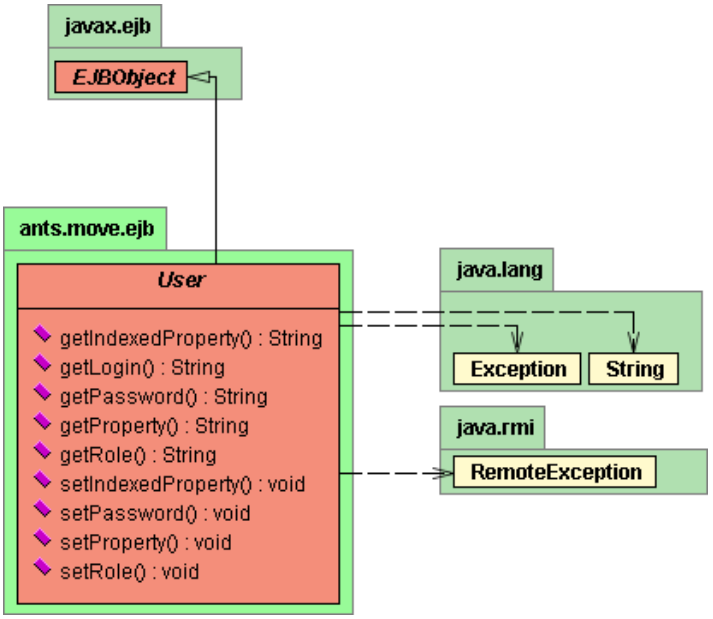


Figura 27. Diagrama de classes de la classe *ants.move.ejb.User*.

La classe que implementa l'*entity bean* User és *ants.move.ejb.UserBean*:

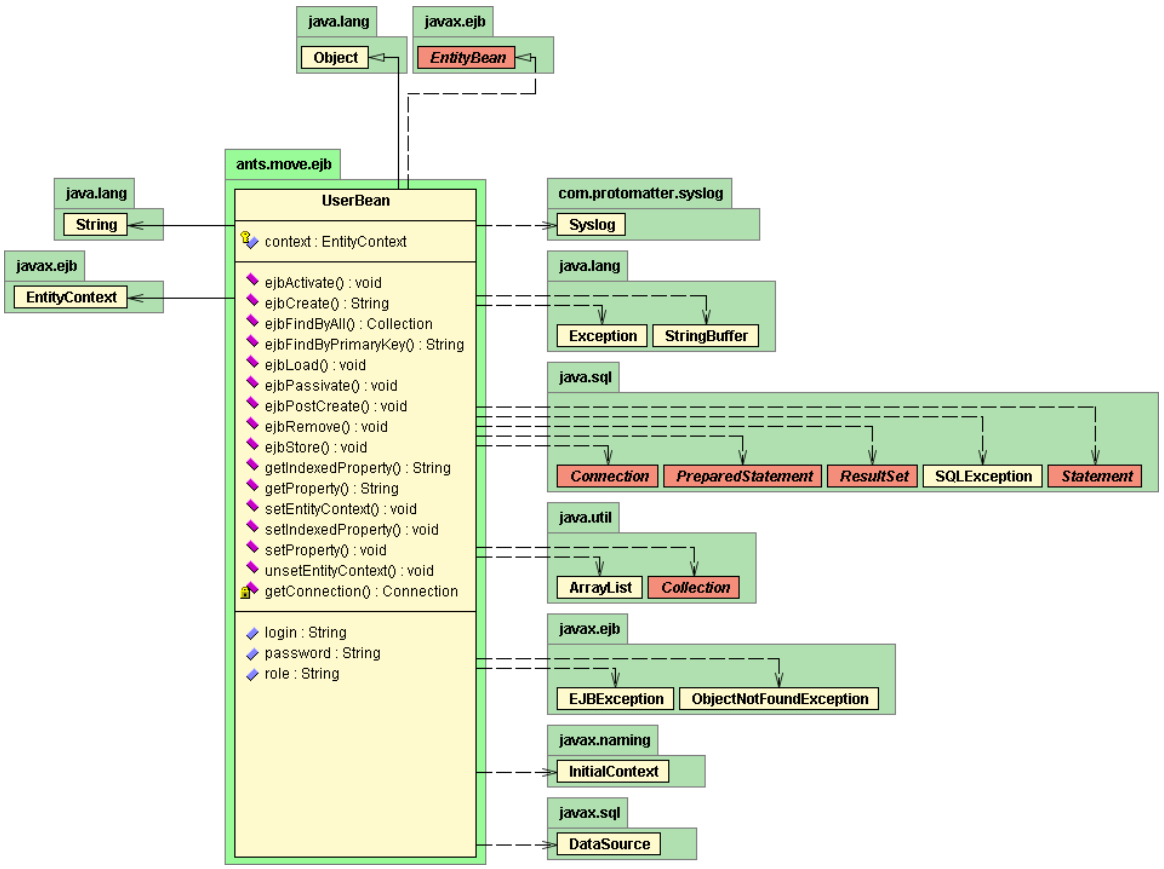


Figura 28. Diagrama de classes de la classe *ants.move.ejb.UserBean*.

5.1.7 Diagrames de classes del session bean *WorldManager*

WorldManager és un *session bean* que ens proporciona els mètodes necessaris per tal de poder treballar amb entitats de tipus *World*. Així doncs, ens proporciona la façana amb els mètodes per poder crear *Worlds*, obtenir-ne un o varis, o bé eliminar-los.

La interfície *home* és *ants.move.ejb.WorldManagerHome*:

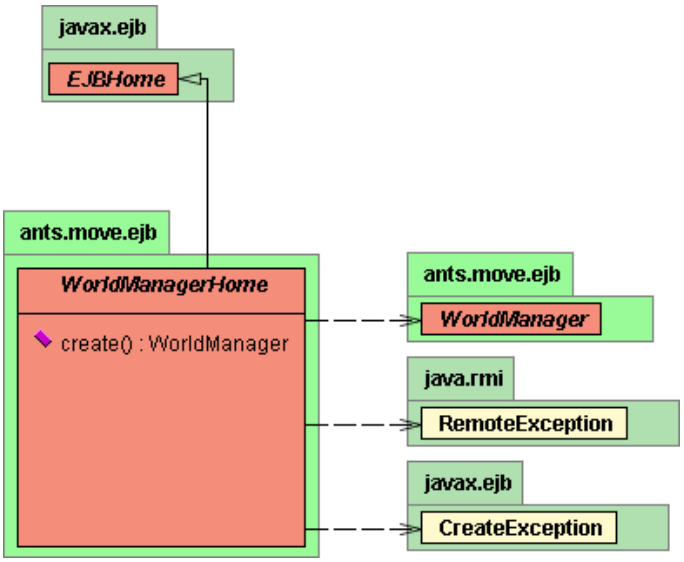


Figura 29. Diagrama de classes de la classe *ants.move.ejb.WorldManagerHome*.

La classe de la *interface* és *ants.move.ejb.WorldManager* i a l'igual que en el cas dels *entity beans*, també proporciona la interfície accessible del *session bean*.

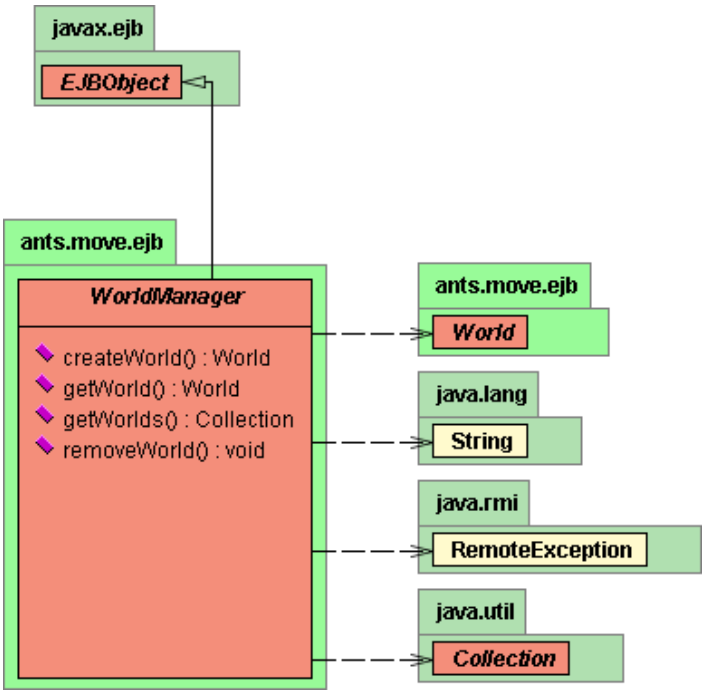


Figura 30. Diagrama de classes de la classe *ants.move.ejb.WorldManager*.

La classe que implementa el *session bean* WorldManager és *ants.move.ejb.WorldManagerBean* i, a l'igual que en el cas dels *entity beans*, també conté la implementació dels mètodes d'aquest.

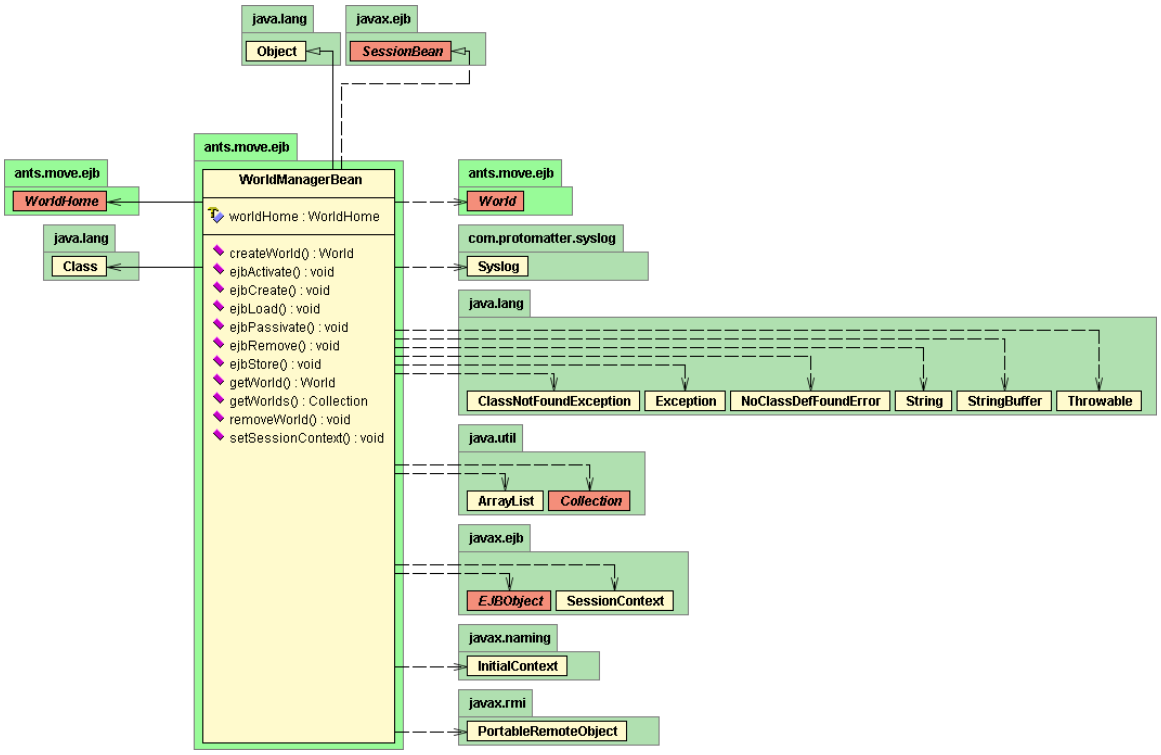


Figura 31. Diagrama de classes de la classe *ants.move.ejb.WorldManagerBean*.

5.1.8 Diagrames de classes del session bean *PlaceManager*

Aquest *session bean* s'encarrega de proporcionar els mètodes necessaris per poder treballar amb entitats de tipus *Place*. Les operacions que inclou són creació, eliminació i obtenció de *Places*.

La interfície *home* és *ants.move.ejb.PlaceManagerHome*:

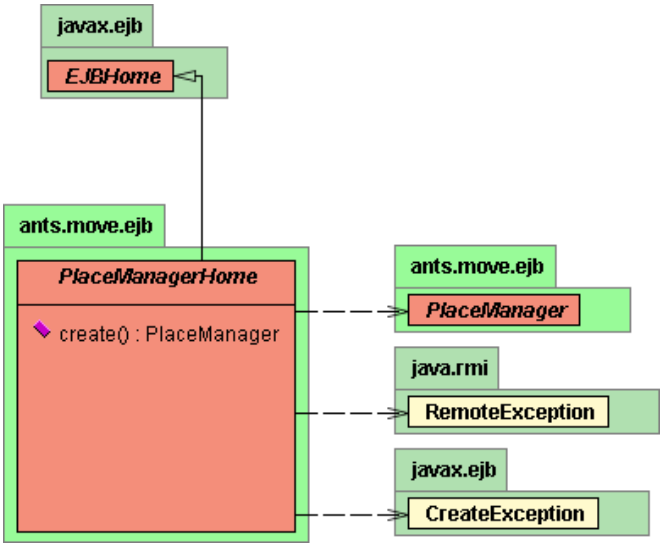


Figura 32. Diagrama de classes de la classe *ants.move.ejb.PlaceManagerHome*.

La classe de la *interface* és *ants.move.ejb.PlaceManager*:

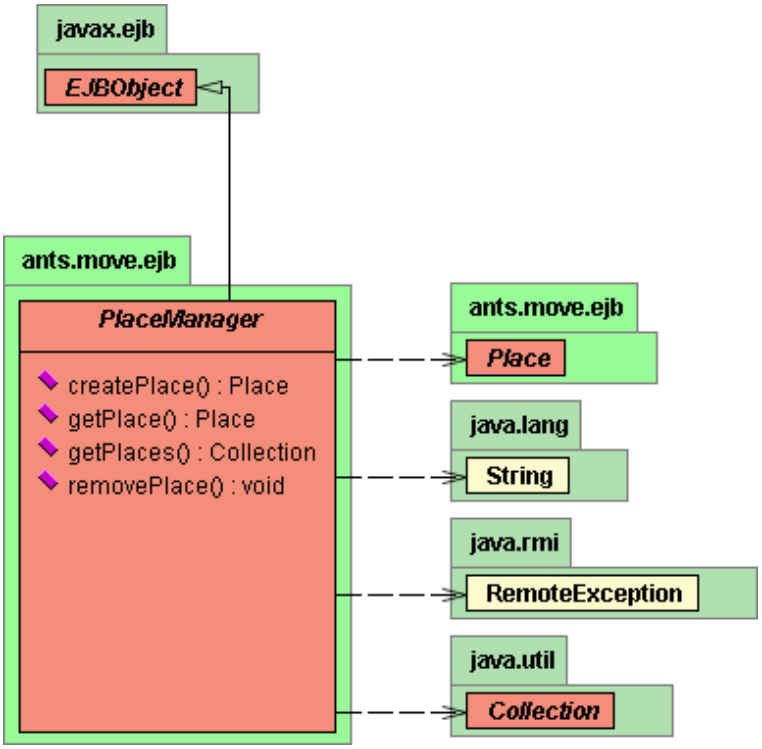


Figura 33. Diagrama de classes de la classe *ants.move.ejb.PlaceManager*.

La classe que implementa el *session bean* PlaceManager és, com és d'esperar, *ants.move.ejb.PlaceManagerBean*.

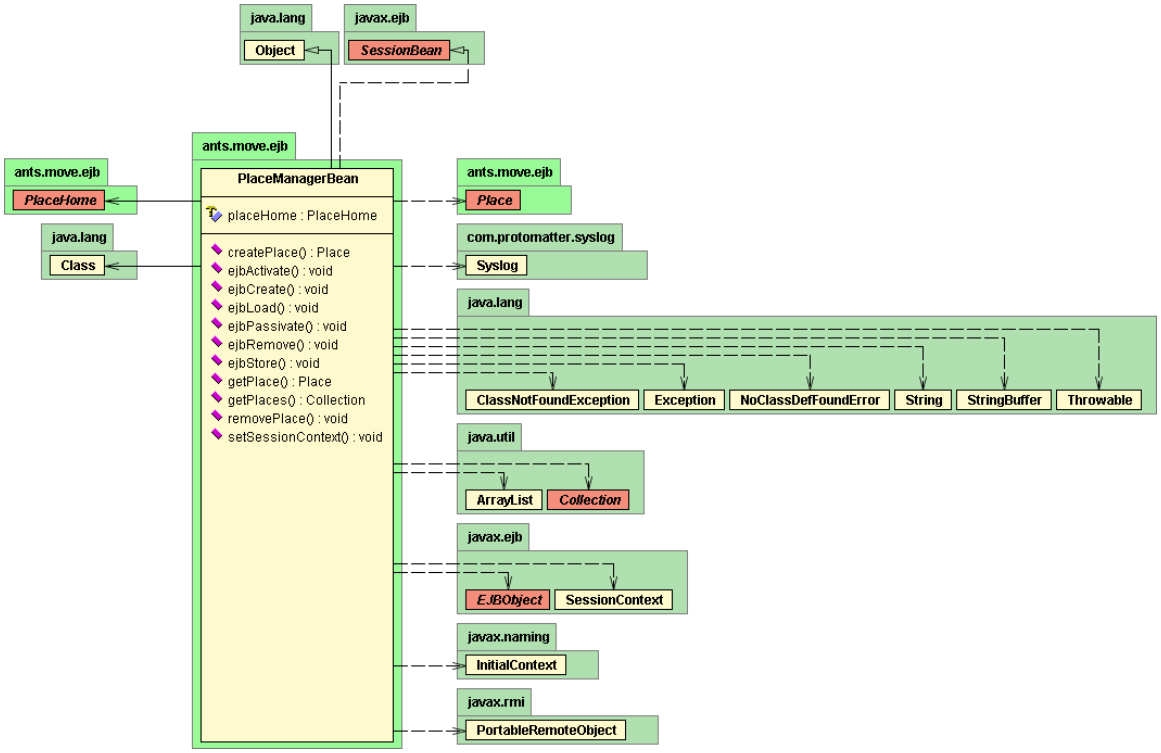


Figura 34. Diagrama de classes de la classe *ants.move.ejb.PlaceManagerBean*.

5.1.9 Diagrames de classes del session bean *ThingManager*

Aquest *session bean* s'encarrega de proporcionar els mètodes necessaris per poder treballar amb entitats de tipus *Thing*. Les operacions que inclou són creació, eliminació i obtenció de *Things*, ja sigui una de sola, de manera genèrica, les things existents en un place o bé les things d'un place i d'una determinada tool.

La interfície *home* és *ants.move.ejb.ThingManagerHome*:

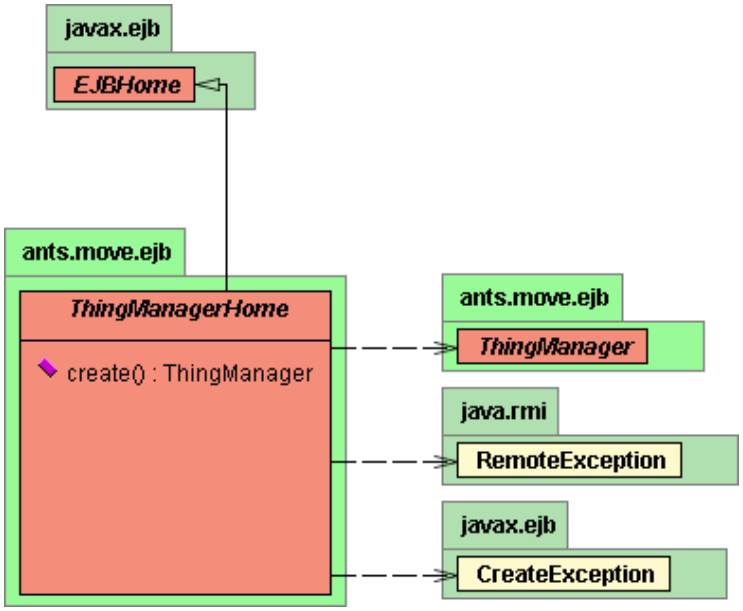


Figura 35. Diagrama de classes de la classe *ants.move.ejb.ThingManagerHome*.

La classe de la *interface* és *ants.move.ejb.ThingManager*:

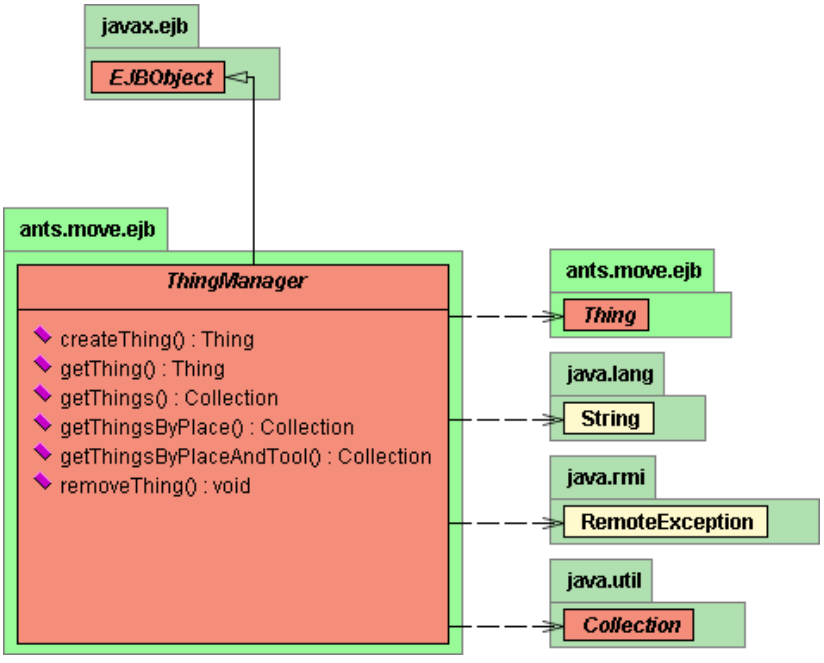


Figura 36. Diagrama de classes de la classe *ants.move.ejb.ThingManager*.

La classe que implementa el *session bean* ThingManager és *ants.move.ejb.ThingManagerBean*.

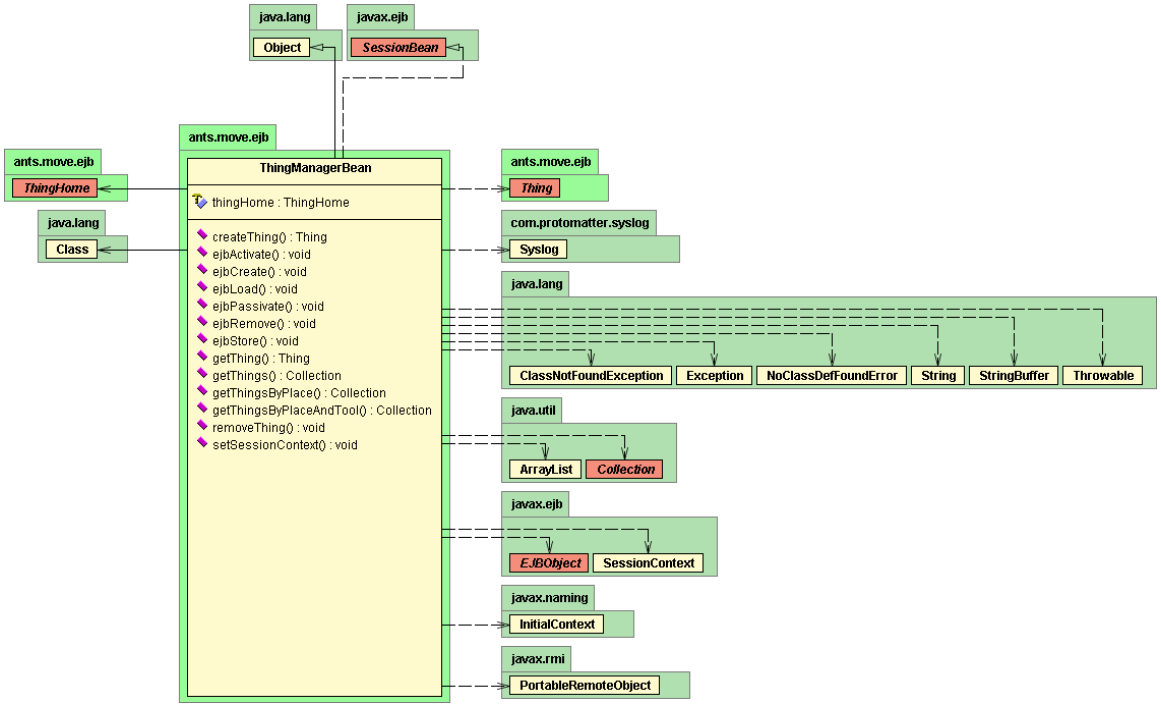


Figura 37. Diagrama de classes de la classe *ants.move.ejb.ThingManagerBean*.

5.1.10 Diagrames de classes del session bean *ToolManager*

Aquest *session bean* s'encarrega de proporcionar els mètodes necessaris per poder treballar amb entitats de tipus *Tool*. Les operacions que inclou són creació, eliminació i obtenció de *Tools*.

La interfície *home* és *ants.move.ejb.ToolManagerHome*:

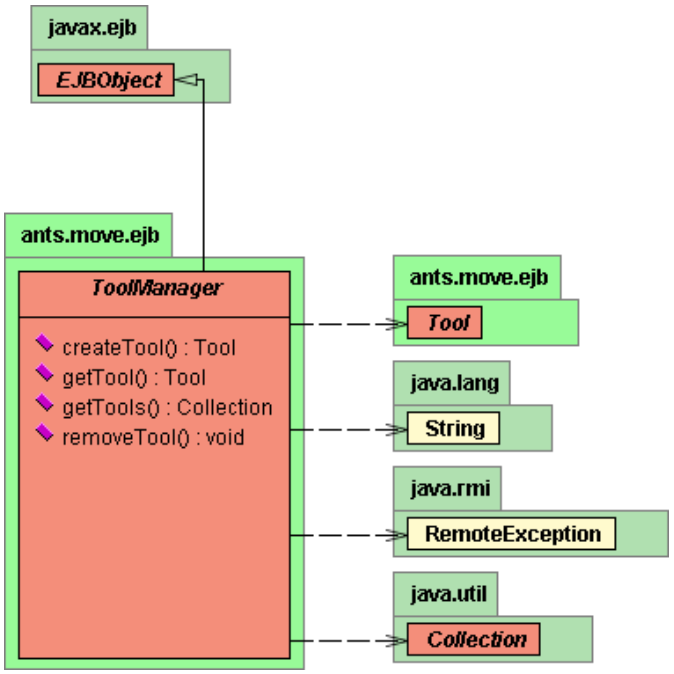


Figura 38. Diagrama de classes de la classe *ants.move.ejb.ToolManagerHome*.

La classe de la *interface* és *ants.move.ejb.ToolManager*:

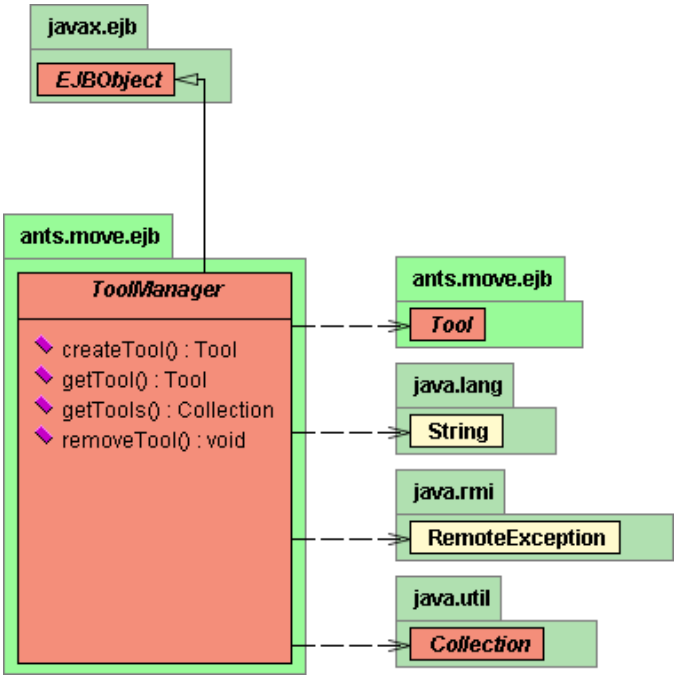


Figura 39. Diagrama de classes de la classe *ants.move.ejb.ToolManager*.

La classe que implementa el *session bean* ToolManager és *ants.move.ejb.ToolManagerBean*.

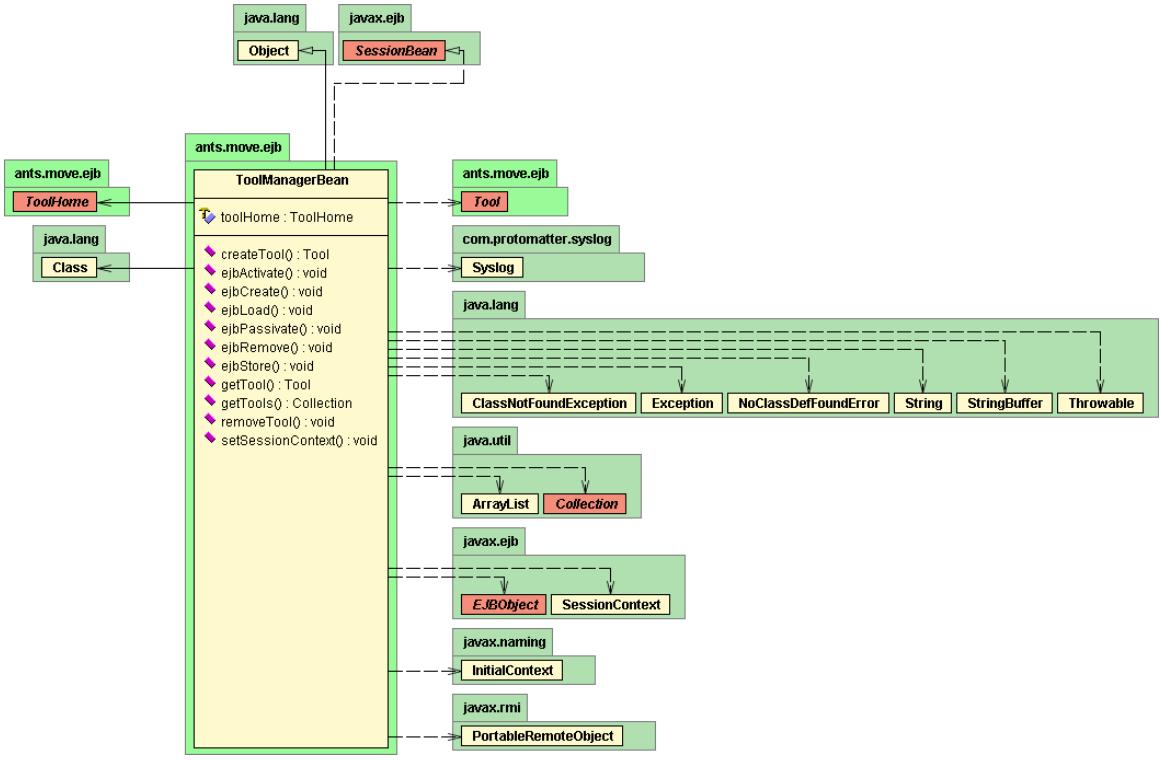


Figura 40. Diagrama de classes de la classe *ants.move.ejb.ToolManagerBean*.

5.1.11 Diagrames de classes del session bean *UserManager*

Aquest *session bean* s'encarrega de proporcionar els mètodes necessaris per poder treballar amb entitats de tipus *User*. Les operacions que inclou són creació, eliminació i obtenció d'usuaris.

La interfície *home* és *ants.move.ejb.UserManagerHome*:

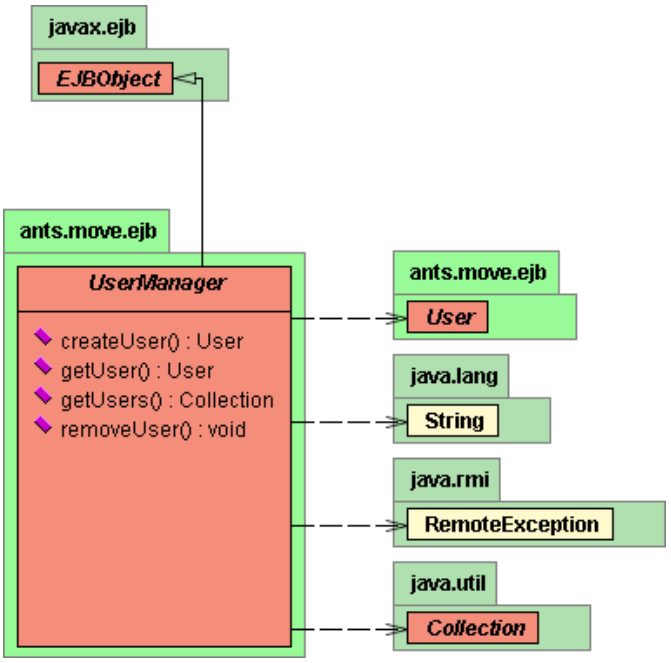


Figura 41. Diagrama de classes de la classe *ants.move.ejb.UserManagerHome*.

La classe de la *interface* és *ants.move.ejb.UserManager*:

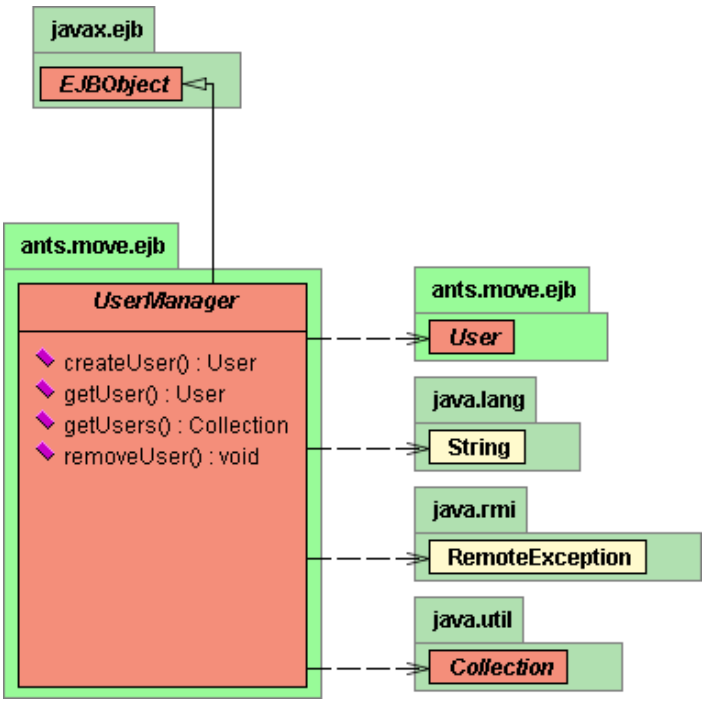


Figura 42. Diagrama de classes de la classe *ants.move.ejb.UserManager*.

La classe que implementa el *session bean* UserManager és *ants.move.ejb.UserManagerBean*.

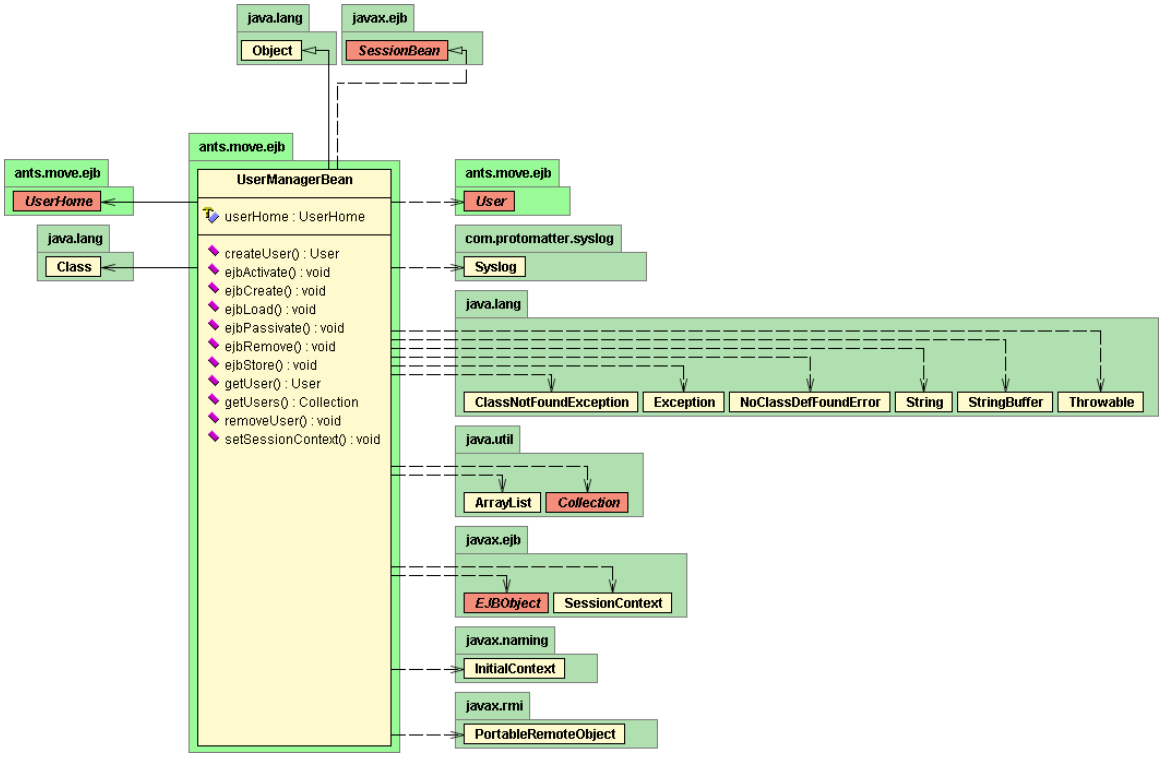


Figura 43. Diagrama de classes de la classe *ants.move.ejb.UserManagerBean*.

5.2 El mòdul *move-web*

Aquest mòdul és l'encarregat de la presentació 2D en pantalla. És per tant, la capa de presentació que permet als usuaris gestionar *worlds*, *places*, *things*, *tools* i *users*. Per tal de realitzar aquesta interfície web amb l'usuari s'han implementat una sèrie de JSPs (JavaServer Pages) que s'encarreguen de gestionar-ne les diferents peticions. A continuació enumerarem i en descriurem el funcionament dels més interessants.

5.2.1 JSPs relatius a Worlds, Places, Things i Users

Enumerarem, per començar, els JSPs que s'encarreguen de la gestió de Worlds, Places, Things i Users i que faciliten operacions com creació, esborrat i modificació.

- **createPlace.jsp, createUser.jsp, createWorld.jsp, newPlace.jsp, newUser.jsp:** Aquests JSPs s'encarreguen de la creació d'un *Place*, d'un *User* i d'un *World* respectivament. Bàsicament consisteixen d'un formulari en format HTML, del qual es demana a l'usuari que introdueixi les dades demanades i a continuació, quan es fa l'acció de *submit*, s'obté la referència al *session bean Manager* corresponent i es crea l'objecte persistent a la base de dades. Cal dir que quan s'autentifica l'usuari dins la nostra aplicació web es carreguen totes les referències als diferents *session beans* (*WorldManager*, *PlaceManager*, *ThingManager*, ...) que es faran servir des dels diferents JSPs i s'emmagatzemen dins la variable ***session*** que representa la sessió de l'usuari. Això es fa així per tal de guanyar eficiència ja que no caldrà fer un *lookup* cada cop que es vulgui obtenir la referència al *session bean*. L'encarregat de realitzar aquesta càrrega de referències dels *session beans* és **loadContext.jsp** i tal com acabem de dir, només es fa un cop durant la vida d'una sessió.

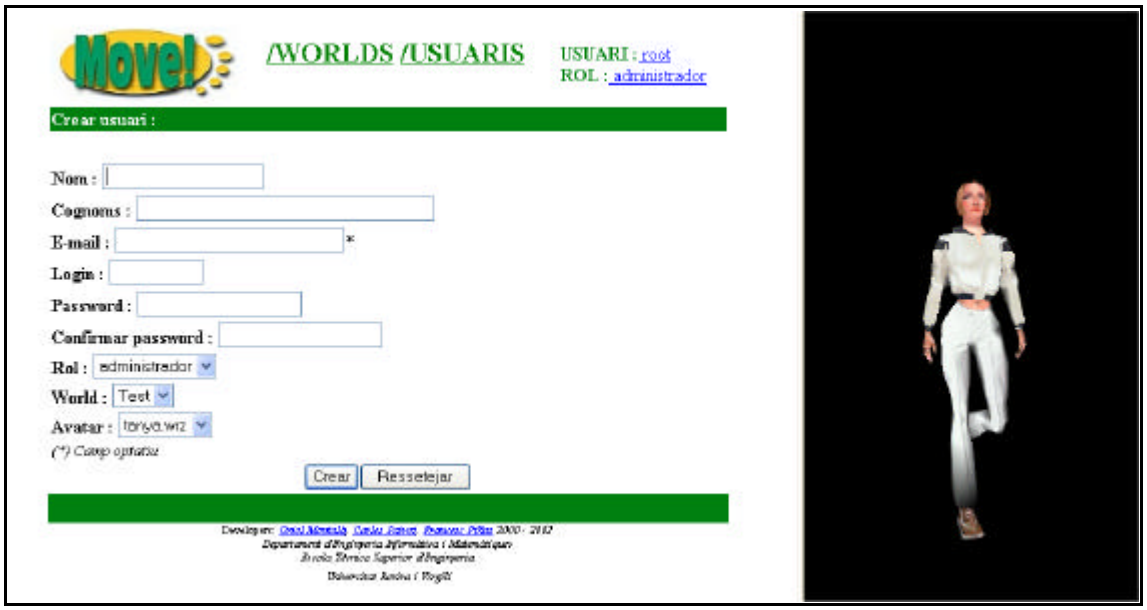


Figura 44. Aspecte del JSP d'entrada de nous usuaris *createUser.jsp*.

- **configurePermissions.jsp:** Aquest JSP és l'encarregat de configurar els permisos d'un tipus d'usuari respecte les diferents *tools* que hi ha instal.lades al

sistema; és a dir, permet determinar si un usuari amb rol *estudiant*, per exemple, quines *tools* tindrà activades i quines no.

- **configureTools.jsp:** S'encarrega de permetre a l'usuari que canviï l'estat d'una tool dins d'un *place*; és a dir, si estarà activada o no.

- **editUser.jsp, editUserProps.jsp:** Permeten canviar les propietats i dades d'un usuari.

- **listPlaces.jsp, listThings.jsp, listUsers.jsp, listWorlds.jsp:** Aquests JSPs mostren un llistat d'objectes, ja siguin *worlds*, *places*, *things*, ..., a l'usuari.

Figura 45. Detall del JSP de llistat d'usuaris *listUsers.jsp*.

- **removePlace.jsp, removeUser.jsp, removeWorld.jsp:** Serveixen per esborrar els diferents objectes. Cal dir que si esborrem, per exemple un *place*, hem d'esborrar-ne també totes les seves *things*. Igual passa si esborrem un *world*: primer n'esborrarem les *things* que conté cada *place* i després les *places* que contenia el *world* a esborrar.

5.2.2 JSPs relatius a les diferents Tools del sistema

Cada *tool* del sistema disposa dels seus propis JSPs, que serviran per crear, previsualitzar i esborrar *things* d'aquell tipus de *tool*.

- **create3DSlides.jsp, createAudio.jsp, createBanner.jsp, ...:** Per cada *tool*, doncs, tindrem un JSP que s'encarregarà de crear instàncies d'aquella *tool*, o el que és el mateix, *things*. Per exemple, per a un projector de transparències (*tool*), un joc de transparències en seria una *thing*, i el JSP encarregat de crear nous jocs de transparències seria **createSlides.jsp**.

Figura 46. Detall del JSP encarregat de crear un nou joc de transparències *createSlides.jsp*.

- **delete3DSlides.jsp, deleteAudio.jsp, deleteBanner.jsp, ...:** Per a cada *tool* també tindrem un JSP que s'executarà quan s'esborri una *thing* del tipus d'aquella *tool*. Aquest JSP esborrarà la *thing* de la base de dades (usant el *session bean ThingManager*) i esborrarà els fitxers de disc (si n'hi havia) corresponents a les dades d'aquella *thing* (imatges, videos, etc.).

- **listenAudio.jsp, previewBanner.jsp, previewSlides.jsp, ...:** Algunes *tools* disposaran de funcions de previsualització per tal que l'usuari pugui veure quin tipus de *thing* s'ha carregat a l'entorn. D'aquesta manera no serà necessari entrar a l'entorn 3D per tal de veure com és realment la *thing* que s'ha pujat.

5.2.3 JSPs de propòsit general

Els JSPs que es descriuen a continuació realitzen funcions de propòsit general o realitzen funcions auxiliars de l'entorn.

- **header.jsp, footer.jsp:** inclouen la capçalera i la part de cua comuns a totes les pàgines.

- **index.jsp:** pantalla d'autentificació.

- **loadContext.jsp:** ja l'hem comentat anteriorment. Carrega totes les referències als *session beans* en la variable de sessió.

- **goToBuilder.jsp:** es crida abans d'entrar a MOVE Builder. S'encarrega de registrar el motor RMI remot de MOVE Builder i mostra la pantalla d'aquest, incrustant l'applet a la pàgina HTML que genera.

- **goToMove.jsp:** es crida abans d'entra a MOVE. Registra la passarel.la d'usuaris (*UserGateway*), la passarel.la d'accés als places (*PlaceGateway*) i la passarel.la d'accés a les things (*ThingGateway*). Aquests objectes formen part del mòdul ANTS CORE, que descriurem més endavant. Finalment, genera el codi HTML

necessari per a obrir la pàgina que contindrà l'*applet* de visualització del place tridimensional.

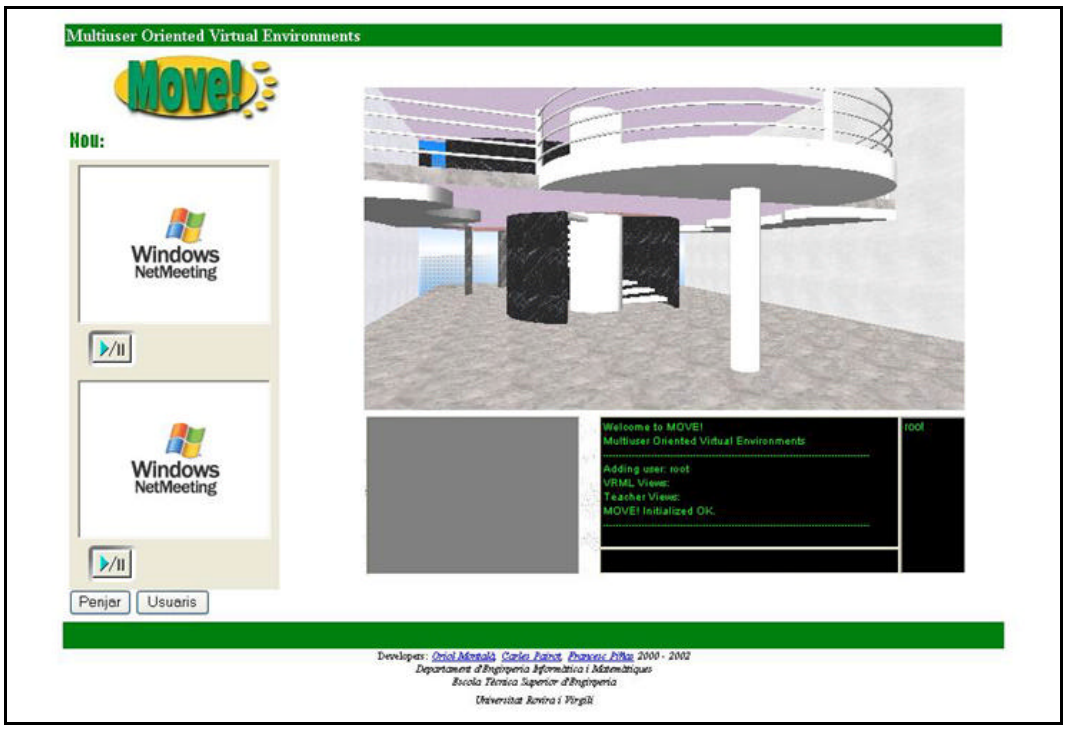


Figura 47. Detall de l'aspecte de MOVE generat amb *goToMove.jsp*.

5.3 El marc de treball col.laboratiu ANTS CSCW

Tal i com ja hem anat comentant al llarg d'aquest document, MOVE fa ús del marc de treball col.laboratiu anomenat ANTS CSCW. Com que el problema general és un repte complex distribuït, es necessiten tecnologies distribuides sòlides com a base així com també proporcionar uns serveis col.laboratius genèrics a les comunitats de desenvolupadors i usuaris.

En la següent figura es pot veure com el bus col.laboratiu és una part clau de l'arquitectura:

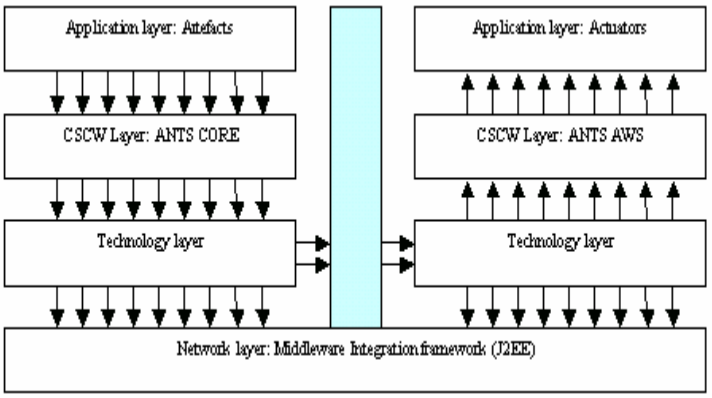


Figura 48. Arquitectura d'ANTS.

Per una banda, tots els components llencen esdeveniments cap al bus distribuït d'informació, i per altra banda, els components de "consciència" (*awareness*) escolten el bus per trobar informació sobre què és el que està passant en aquell moment en el sistema.

Es poden distingir tres capes principals en aquesta arquitectura: una capa de tecnologia (*Technology layer*), una capa de serveis CSCW (*CSCW layer*) i una capa d'aplicació (*Application layer*). Les capes superiors representen abstraccions de més alt nivell que oculten la complexitat als usuaris de les capes inferiors.

En aquesta línia, la capa d'aplicació està orientada a usuaris finals que extenguin el marc de treball per crear components col.laboratius i aplicacions. En el nostre cas, a la capa d'aplicació hi tindrem els *applets* de MOVE, així com les diferents *tools* del sistema. La capa CSCW comprèn el conjunt de serveis col.laboratius que el marc de treball ofereix als components del nivell d'aplicació. A més, aquests serveis CSCW estan construits per sobre dels serveis distribuits que són oferts per la capa de tecnologia. Anem a veure'n les principals funcionalitats:

- Al nivell d'aplicació, s'ha creat un model de components que abstreu l'accés a un component persistent remot basat en propietats, i als esdeveniments distribuits produits en l'actual context de sessió.

- La capa CSCW comprèn el conjunt de serveis col.laboratius que el marc de treball ANTS ofereix a la capa d'aplicació d'una manera coherent. Proporciona,

per tant, suport per a sessions, cicle de vida del component, abstracció de la comunicació, suport per coordinació i seguretat.

- A la capa de tecnologia s'han creat una sèrie de components software que proporcionen una façana transparent cap a serveis de *middleware* avançats i components de xarxa. Aquesta aproximació permet que puguem seleccionar dins d'una gran varietat de manufacturadors de *middleware* en sistemes de bases de dades, servidors d'aplicacions i serveis de notificacions.

5.3.1 La capa d'aplicació

A la capa d'aplicació, el marc de treball proporciona dues extensions: desenvolupament de nous components col.laboratius i actuadors de consciència que reaccionen als esdeveniments d'informació produits a dins del marc de treball.

El principal objectiu és facilitar el desenvolupament de nous components col.laboratius, i per tant, aconseguir una transició suau en el pas d'aplicacions locals a distribuides. Òbviament, aquesta aproximació s'ha de basar en especificacions ja existents i ja adoptades en la comunitat de desenvolupadors.

Com d'altres *groupware toolkits[38]*, s'ha decidit utilitzar el llenguatge *Java* i l'especificació de components *JavaBean*, com les nostres tecnologies de desenvolupament. En aquesta línia, aquesta aproximació ens porta a sostenir components *JavaBean* en un contenidor que oculta la complexitat en la programació distribuida. La idea general és proporcionar un sistema de persistència, i construir un servei d'esdeveniments *Java* distribuïts.

5.3.2 La capa CSCW

La capa CSCW comprèn el conjunt de serveis col.laboratius que el contenidor d'ANTS ofereix a la capa d'aplicació. Es distingeixes dos mòduls principals, que són el contenidor en temps d'execució (ANTS.CORE) i els serveis de consciència (ANTS.AWS), connectats entre sí pel bus col.laboratiu. Només comentarem el mòdul ANTS.CORE, comentant de manera més específica el mòdul ANTS.AWS més endavant.

5.3.2.1 El mòdul ANTS.CORE

ANTS.CORE és un mòdul essencial del marc de treball ANTS, que inclou suport explícit per sessions, artefactes compartits, control de coordinació, seguretat i un model de seguretat.

Hem estat fortament influenciats per les arquitectures multiusuari orientades a objectes (MOO[13]) degut a la seva arquitectura extensible i genèrica. En un sistema MOO, els objectes de tipus *place*, que representen localitzacions discretes interconnectades per objectes portals, defineixen la topologia de l'espai. Aquests

[13] **MOO:** Multi-user Object Oriented.

objectes portals poden establir qualsevol relació dirigida per grafs entre diferents *places*. En una arquitectura MOO, qualsevol entitat estàtica o dinàmica és un objecte (*thing*), qualsevol objecte té un conjunt de propietats i un conjunt de verbs, i aquests conjunts de propietats i verbs es poden afegir dinàmicament en temps d'execució.

Com en un sistema MOO, la nostra sessió compartida es representa per l'objecte *Place*, interconnectat per objectes portals. Els artefactes compartits es representen per components *JavaBean* que es carreguen dinàmicament pel contenidor d'ANTS. Les propietats dels objectes es representen en el nostre model pel component persistent remot, i els verbs dels objectes són classes *JavaBean*.

La *Place* representa la sessió compartida que conté usuaris, components i enllaços a altres *places*. Proporciona mètodes per tal d'enviar o subscriure's a esdeveniments en el context actual. També proporciona mètodes per obtenir els usuaris connectats i els enllaços disponibles cap a d'altres *places*. A més també permet la càrrega dinàmica de components *JavaBean* al context compartit. El cicle de vida d'un component seria, de manera genèrica, el següent:

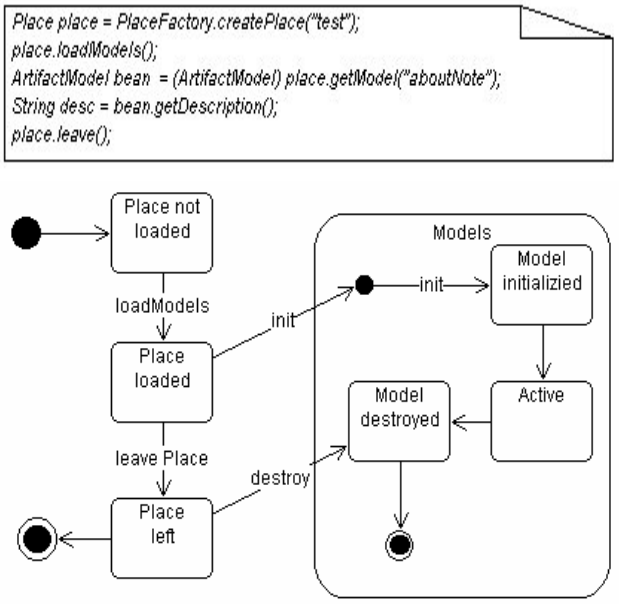


Figura 49. Cicle de vida d'un component.

Quan s'invoca el mètode *loadModels()* en un *place*, s'activen tots els components *JavaBean* (cridant-ne el seu mètode *init()*) i les classes es carreguen dinàmicament. Finalment, quan l'usuari abandona la sessió compartida, s'invoca el mètode *destroy()* en tots els components *JavaBean* que estan actius en el contenidor.

5.3.2.1.1 Diagrames de classes

El package ANTS.CORE es compòn de les següents classes:

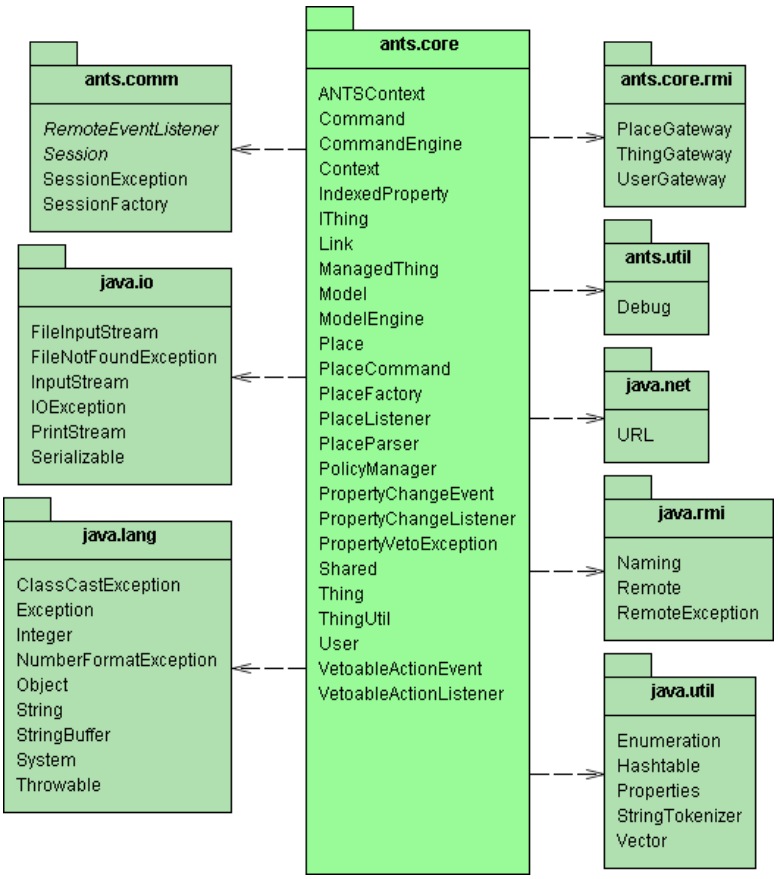


Figura 50. Diagrama de classes del package ANTS.CORE.

Veiem com existeixen les classes mencionades anteriorment: *Place* i *Thing*, a part d'una altra que encapsula els usuaris (*User*). També es pot veure com aquest package fa ús de les classes del paquet ANTS.CORE.RMI. La funció de les classes *PlaceGateway*, *ThingGateway* i *UserGateway* és la de fer de passarel.la entre les classes *ants.core.Place*, *ants.core.Thing* i *ants.core.User* i les seves corresponents classes en *J2EE*. És a dir, a través de les classes *Gateway* es pot accedir als mètodes que proporcionen les dades de les classes persistents *ants.move.ejb.Place*, *ants.move.ejb.Thing* i *ants.move.ejb.User*. Dins les classes *Gateway*, doncs, s'obtindrà les referències als diferents *managers* (*PlaceManager*, *ThingManager* i *UserManager*) i d'aquesta manera es podrà accedir a les dades emmagatzemades als EJBs (i el que és el mateix, les dades emmagatzemades a la base de dades).

Aquest esquema s'ha ha gut d'adoptar ja que l'accés als *session beans* directament des de l'*applet* del client és impossible, degut a que la versió de la màquina virtual de Java inclosa en la majoria dels navegadors és la 1.1.5 i les classes de *J2EE* no estan incloses.

D'aquesta manera, si un client vol obtenir el valor d'una propietat d'una *Thing*, el que farà és fer una crida al mètode *getProperty()* de la classe *ants.core.Thing*, que el que farà serà fer una crida remota al mètode *getProperty()* de la classe *ants.core.rmi.ThingGateway*, que al seu temps, i al tractar-se d'un objecte que es troba en el servidor, ja podrà accedir als *session beans* i per tant, demanar la obtenció de la propietat, que retornarà primerament a *ThingGateway* i finalment a la classe *Thing*.

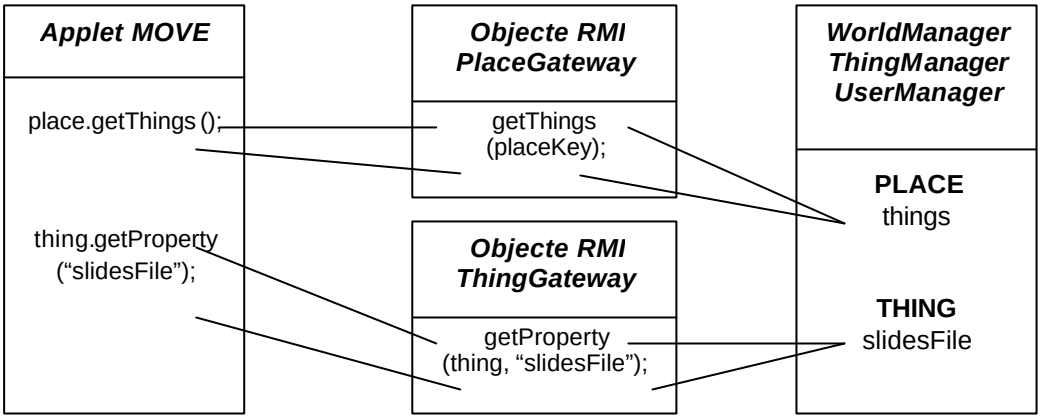


Figura 51. Esquema de dues crides i com s'executen usant ANTS.CORE.RMI.

Ara podem veure el diagrama de classes del package ANTS.CORE.RMI:

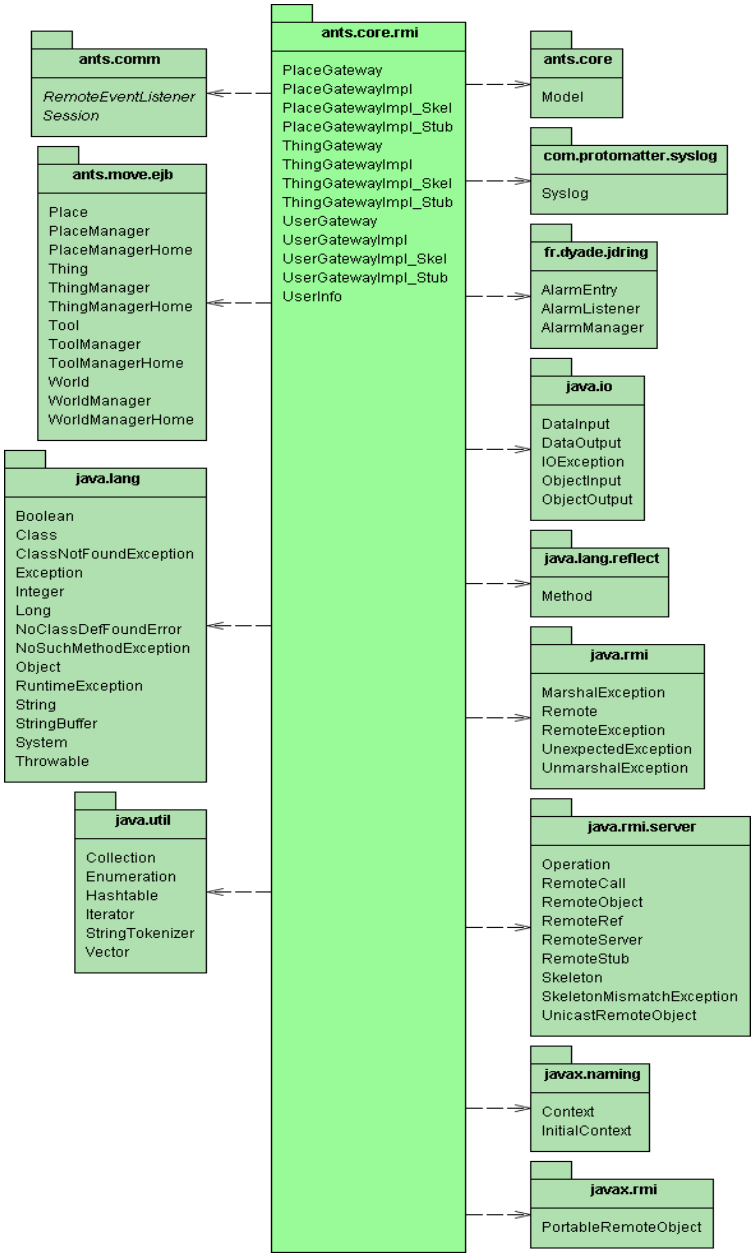


Figura 52. Diagrama de classes del package ANTS.CORE.RMI.

5.3.2.1.2 Descripció de les classes d'ANTS.CORE.RMI

Primer de tot comentarem la classe *ants.core.rmi.PlaceGateway*, el diagrama de classes de la qual és aquest:

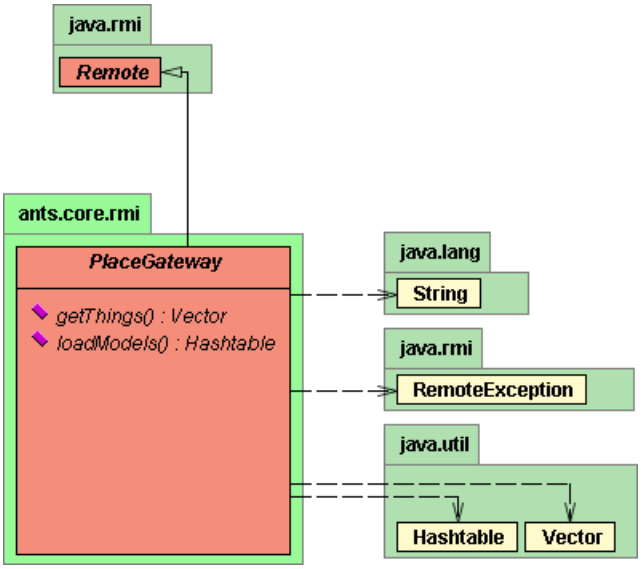


Figura 53. Diagrama de classes de la classe *ants.core.rmi.PlaceGateway*.

Al tractar-se d'una classe RMI remota, doncs només comentarem la classe d'interfície (que proporciona els mètodes que veurà la classe *ants.core.Place*, en aquest cas) i no la part d'implementació.

Els mètodes, doncs que proporciona, són *getThings()*, que ens retornarà totes les *things* existents en un place determinat (del qual se'n passarà la clau primària com a paràmetre) i el mètode *loadModels()*, que serà l'encarregat de llegir tots els artefactes compartits (altrament anomenats *models*) i guardar-los (si no estan desactivats) en una taula de hash que es retornarà al mètode *loadModels()* de la classe *ants.core.Place*, que és la que crida *ants.core.PlaceGateway.loadModels()*.

Passem a veure, tot seguit, el diagrama de classes d'*ants.core.rmi.ThingGateway*:

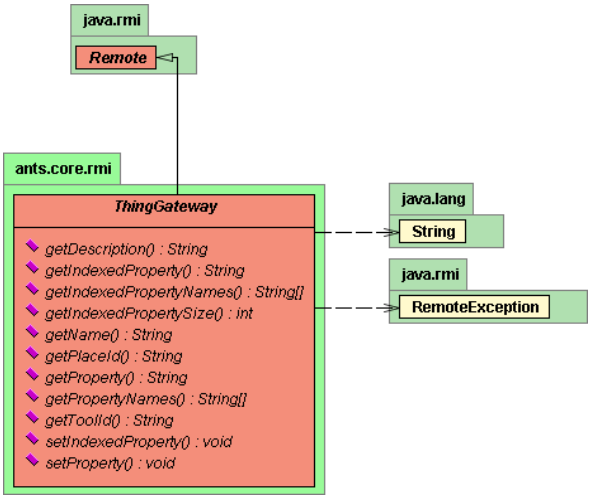


Figura 54. Diagrama de classes de la classe *ants.core.rmi.ThingGateway*.

Veiem que aquesta classe incorpora els mètodes necessaris per obtenir propietats, fixar-les, obtenir el nom de la *thing*, l'identificador del *place* al que pertany, etc. El funcionament és el mateix que en el cas anterior: la classe *ants.core.Thing*, en aquest cas és qui crida aquests mètodes, que obtindran les dades necessàries accedint al *session bean ThingManager* i les retornaran a *ants.core.Thing*. En el cas que es fixin valors de propietats, el valor s'emmagatzema a l'*entity bean Thing*, a través, com sempre del *session bean ThingManager*.

Per últim disposem de la classe *ants.core.rmi.UserGateway*:

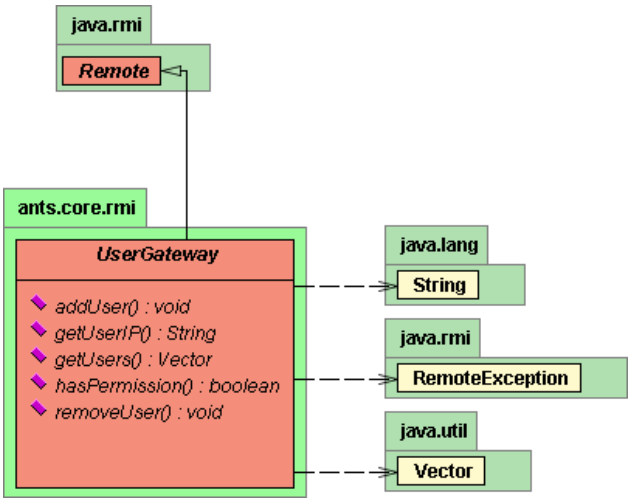


Figura 55. Diagrama de classes de la classe *ants.core.rmi.UserGateway*.

Aquesta classe ens permetrà afegir i eliminar usuaris a un *place*, obtenir la seva IP, obtenir els usuaris que hi ha en un moment determinat en un *place* i comprovació de permisos. A més, també realitza les funcions de comprovació de si un usuari és "viu" o no, és a dir, que durant l'execució de MOVE, l'objecte *UserGateway* s'encarrega d'enviar un esdeveniment de **PING** cada cert temps a tots els usuaris de totes les *places* 3D actives en aquell moment. Si algun usuari no respon aquest esdeveniment, voldrà dir que el seu client s'ha penjat, i per tant, a la següent ronda de **ping**, aquest serà eliminat del *place* en què es troba. Així doncs, es tracta d'un mecanisme de seguretat que permet detectar si a un usuari se li ha penjat el navegador o no.

5.3.2.1.3 Descripció de les principals classes d'ANTS.CORE

Començarem comentant la classe *ants.core.Place*, el diagrama de classes de la qual és el següent:

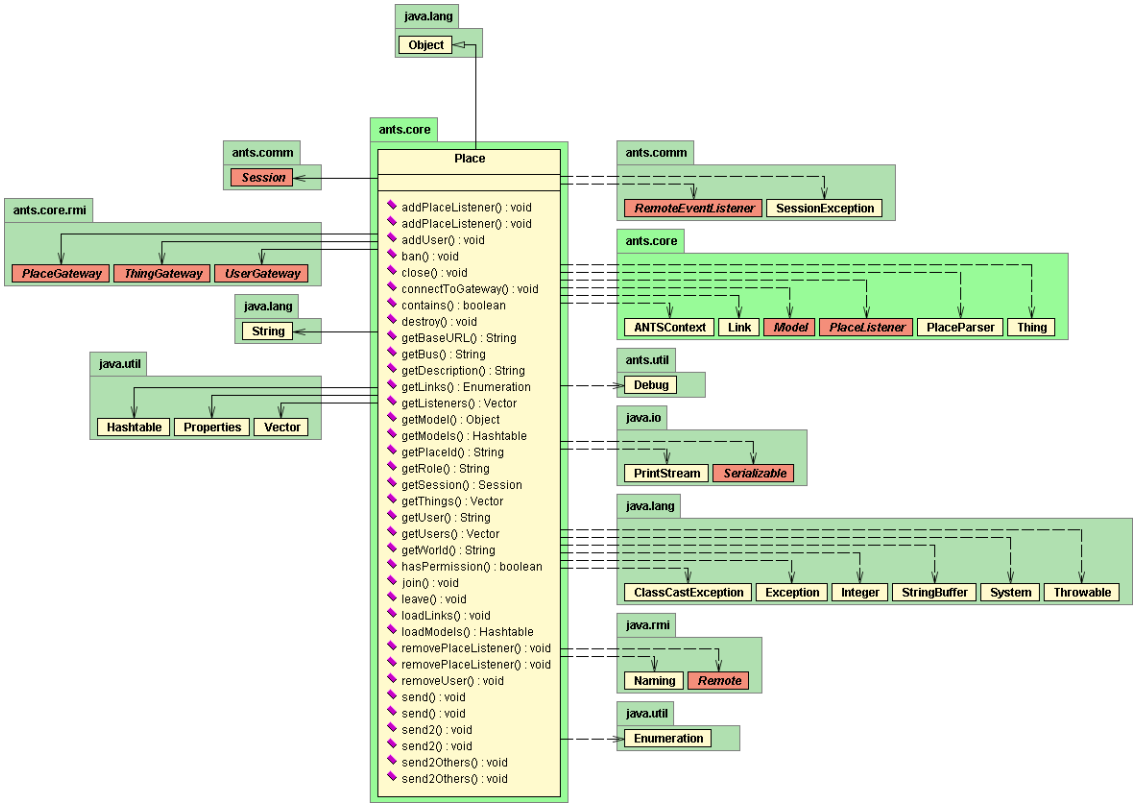


Figura 56. Diagrama de classes de la classe *ants.core.Place*.

Aquesta classe proporciona tots els mètodes necessaris per a encapsular un espai que conté una sèrie d'usuaris i una sèrie d'artefactes compartits. Proporciona mètodes per a l'afegiment d'usuaris, càrrega de models, obtenció d'aquests, enviament d'esdeveniments dins d'aquell *place*, entrar al *place*, abandonar-lo, etc.

Com a mètode més interessant, comentarem el mètode *loadModels()*. Aquest mètode és l'encarregat de carregar tots els artefactes compartits existents dins la *place* i inicialitzar-los. Primerament, obté tots els models del *place* mitjançant una crida al mètode *loadModels()* de la classe *ants.core.rmi.PlaceGateway*. Cal dir que tots els models implementen la classe *ants.core.Model*, amb la qual implementen varis mètodes, com ara el mètode *init()*, per a inicialitzar-los i el mètode *destroy()*, que es crida quan l'usuari abandona el *place* on es troba.

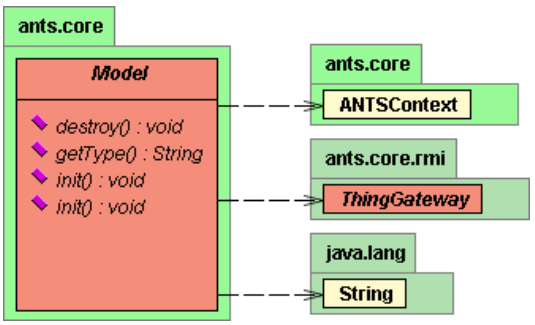


Figura 57. Diagrama de classes de la classe *ants.core.Model*.

Un cop obtinguts tots els models que ens ha retornat la crida a *ants.core.rmi.PlaceGateway.loadModels()*, els anem recorrent un a un i els anem cridant el mètode *init()*, de manera que queden inicialitzats (les inicialitzacions comprendran la subscripció al tipus d'esdeveniments que ha d'escoltar la *thing*, entre d'altres accions) i finalment, passem a retornar la llista de models a quin ens ha cridat.

Passem a veure, tot seguit, la classe que encapsula a un artefacte o objecte compartit dins el nostre entorn 3D: la classe *ants.core.Thing*. Tots els objectes compartits que tinguem heredaran de la classe *ants.core.Thing*, mentre que aquesta última veiem que implementa la classe *ants.core.Model*, ja que necessita tenir els mètodes *init()* i *destroy()* esmentats anteriorment.

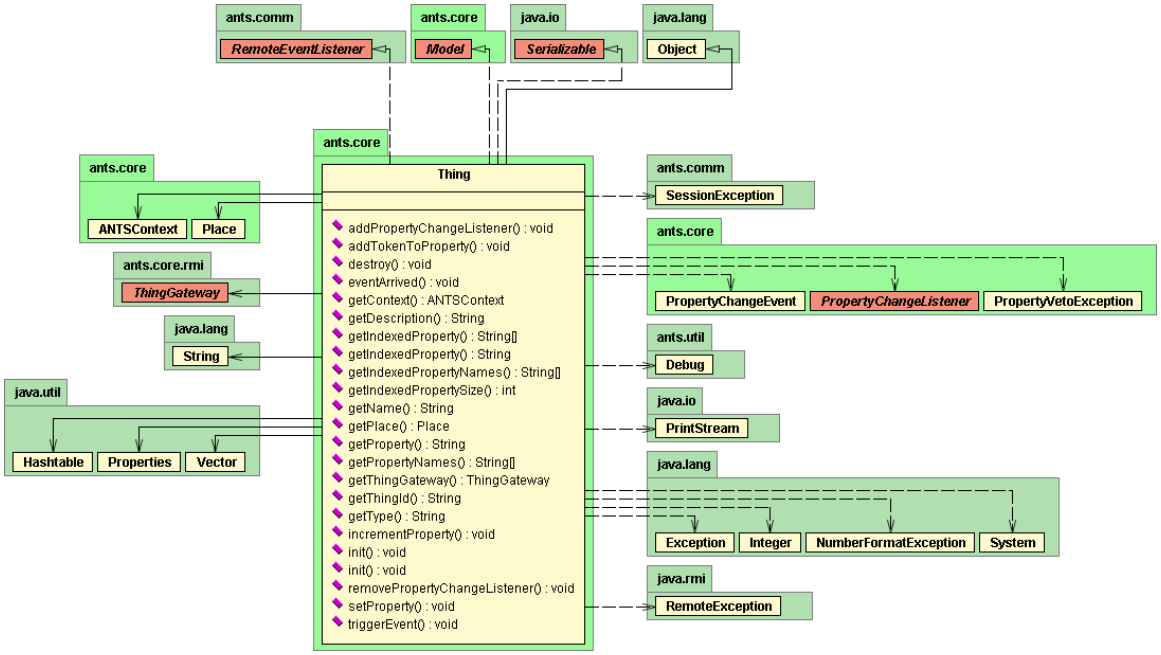


Figura 58. Diagrama de classes de la classe *ants.core.Thing*.

Aquesta classe proporciona mètodes per fixar i obtenir propietats de la *thing* en qüestió. A més, cada cop que es canvia una propietat es dispara un esdeveniment informant als altres clients que hi ha hagut un canvi en l'estat d'aquella *thing*. Com a exemple podem veure el codi del mètode *setProperty()*:

```
public synchronized void setProperty (String propName, String propValue)
       throws PropertyVetoException {
   try {
      tgate.setProperty (getThingId(), propName, propValue);
   } catch (java.rmi.RemoteException rex) {
      Debug.error (rex);
   }

   Hashtable evt = new Hashtable();
   evt.put (Context.METHOD,"SETPROPERTY");
   evt.put ("PROPNAME",propName);
   evt.put ("PROPVALUE",propValue);
   evt.put ("INDEX","-1");

   triggerEvent(evt);
}
```

Veiem com primerament es fixa el valor de la propietat cridant el mètode *setProperty()* de l'objecte *tgate* que és de la classe *ThingGateway*. Un cop fixat, formem el missatge de la notificació, especificant que fixem el valor d'una propietat (*METHOD = SETPROPERTY*) i posem el nom de la propietat i el seu valor. Finalment, llencem l'esdeveniment cridant el mètode *triggerEvent()*.

Quan una *thing* rep un esdeveniment al qual està subscrita, se li crida el mètode *eventArrived()* de manera automàtica i així pot actuar en conseqüència.

5.3.3 La capa de tecnologia

Primer de tot, un problema com aquest requereix una infrastructura que proporcioni seguretat, escalabilitat, persistència, transaccions i eficiència. Bastants sistemes existents confien en entorns no distribuits i monothreaded que limiten la seva aplicabilitat a problemes més complexos.

Per aquesta raó hem escollit l'estàndard *J2EE* com la nostra infrastructura tecnològica. Ens estalvia implementar la infrastructura i codi específic del sistema i permet basar-nos en especificacions i components oberts. Aquesta tecnologia fa que el nostre sistema sigui independent del manufacturador: podem triar qualsevol servidor d'aplicacions compatible amb *J2EE* (nosaltres hem triat *Orion Application Server 1.5.2*, però també es podria haver triat *JBoss 3.0*, o bé qualsevol altra solució comercial) o base de dades relacional (en el nostre cas *HypersonicSQL*, però es podria haver triat *Oracle*, *mySQL*, etc.).

Respecte al *middleware* orientat a missatges, necessitàvem un servei de notificacions de tipus publicació/subscripció per al marc de treball. Essent *JMS*[14] una alternativa estàndard, també estàvem força interessats en el servei de notificacions *Elvin*[15]. Aquest servei proporciona un molt bon rendiment i ha estat utilitzat per a propòsits de CSCW. Com a resultat, es va implementar una API de façana que ens permetés triar entre qualsevol solució de missatgeria compatible amb JMS i *Elvin*.

[14] **JMS:** Java Message Service.
[15] Per a més informació, visiteu la següent adreça: http://elvin.dstc.edu.au

5.3.4 De quina manera MOVE fa ús d'ANTS CSCW

En aquesta secció veurem l'arquitectura col.laborativa de MOVE i com fa ús dels serveis que proporciona el marc de treball ANTS CSCW.

5.3.4.1 Manegament de sessió i zona

Per tal de determinar l'estructura de l'entorn, es fa servir, tal i com hem vist anteriorment, el concepte de *Place*. Una *place* és una peça d'entorn virtual on els usuaris poden interactuar entre sí i fer ús dels diferents artefactes compartits disponibles. ANTS.CORE defineix una sessió com un grup d'objectes associats amb algun esquema de comunicacions comú i suportant una comunicació *full-duplex* multipunt entre un nombre arbitrari d'entitats d'aplicació connectades. Això és essencialment el que és una *place* a dins de MOVE i en qualsevol altre entorn de tipus MOO. Fent ús d'aquesta filosofia, s'identifica sessió amb *place*.

5.3.4.2 Artefactes compartits i propagació d'estat

Un altre concepte essencial és el suport explícit per a **artefactes compartits**. Normalment, els usuaris entren en una interacció col.laborativa amb d'altres usuaris en el context compartit, o amb components col.laboratius accessibles. Aquests components col.laboratius – normalment anomenats **artefactes** (*artifacts*) – van des de components síncrons a asíncrons, depenent del temps d'interacció. El suport explícit per a la creació i inserció de nous artefactes representa un factor diferenciador entre *toolkits* CSCW extensibles i no extensibles.

MOVE es basa en una arquitectura modular per a cada component. Hi ha diferents tipus de components disponibles, com ara eines de votació, transparències, videos, documents o multimedia. Tots aquests components segueixen el paradigma Model-Vista-Control.lador (MVC). Cada client té una còpia local del control.lador, que és l'eina utilitzada per a modificar l'estat del component, i la vista, que és la representació del component que es mostra a la pantalla.

Un dels principals objectius per dissenyar el sistema ha estat fer que fos fàcilment extensible i reutilitzable. Per fer-ho, cada component disposa d'un descriptor *XML* que conté totes les propietats necessàries i la informació relacionada als esdeveniments que es poden llençar, a més de les classes del component i els recursos locals. Aquest nou component s'empaqueta utilitzant el format estàndard *JAR*[16].

Un dels problemes més difícils de resoldre quan es programen entorns distribuits és el fet de mantenir l'estat dels components entre tots els altres participants. En el nostre cas, aquest problema es minimitza i es fa pràcticament transparent gràcies a l'ús del *framework* ANTS CSCW, ja que aquest ens garanteix que l'estat es propagarà a tots els usuaris en el mateix *place* en que estem nosaltres. D'aquesta manera, quan desenvolupem nous components per al nostre sistema, aquest problema ja està resolt i podem concentrar els nostres esforços de desenvolupament en altres aspectes.

[16] **JAR:** Java Archive.

S'ha desenvolupat un conjunt d'artefactes compartits que inclouen una eina de votació, una eina de transparències, una eina de barres, una de documents, una d'audio, una de video, videoconferència i video streaming; així com dues eines per captar l'atenció dels usuaris (les eines *hook* i *camera*). En aquests components s'han utilitzat els mecanismes de propagació d'estat proporcionats pel marc de treball ANTS CSCW.

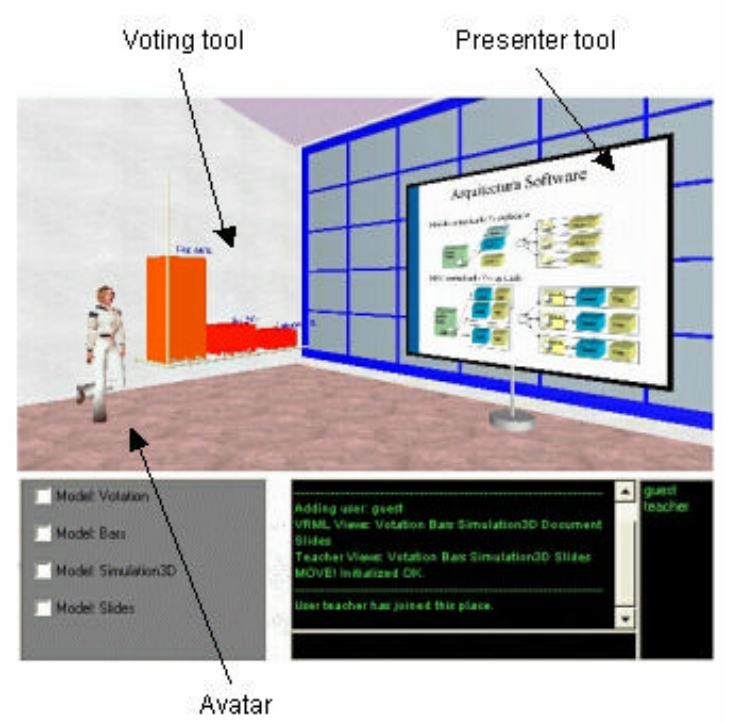


Figura 59. Captura de pantalla de Move que mostra els artefactes compartits de **transparències** i **votació**, així com un usuari (**avatar**).

Tot i així, hi ha alguns components que no poden utilitzar els mecanismes de propagació d'estat que el marc de treball ens proporciona. Un dels exemples que ens hem trobat és el component de moviment dels *avatars*[17]. És clar que és impossible transmetre tots i cadascun dels esdeveniments de posició generats pel moviment d'un cert usuari cap als altres usuaris del mateix *place*. Això generaria una enorme sobrecàrrega d'esdeveniments per la xarxa que fins i tot els clients serien incapaços de gestionar. En aquest cas, hem hagut d'optimitzar aquest component implementant filtres d'esdeveniments per proximitat. Tal i com és sabut, un dels punts febles dels entorns tridimensionals és l'immens temps de processament destinat a generar l'escena 3D. És per això que quan hi ha molts usuaris en el mateix *place*, el cost de *renderitzar-los* és inacceptable per al client. Per solucionar aquest problema, hem implementat un algorisme d'esdeveniments de proximitat, de manera que els clients només reben els esdeveniments dels avatars que estan en una certa distància llindar a prop d'ells, tal i com es veu en la següent figura:

[17] Més endavant es veurà com funciona detalladament. Malgrat tot, ara en fem un esbós.

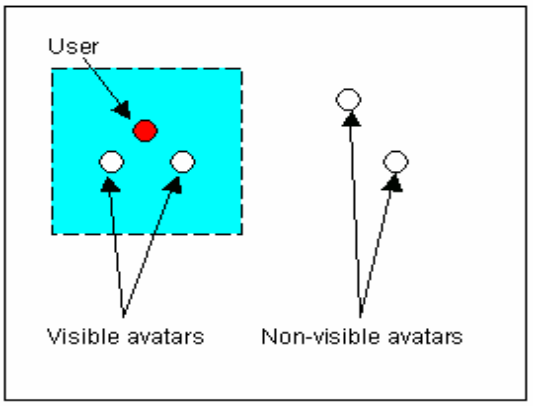


Figura 60. Només els avatars que estiguin en la distància llindar seran *renderitzats*.

5.3.4.3 Coordinació i consistència

Un altre aspecte clau en qualsevol marc de treball CSCW és el suport explícit per software de **polítiques de coordinació**. Aquestes polítiques poden ser fixades programàticament o declarativament, i es poden categoritzar en rols per al control d'accés i el control de la concurrència. La plataforma hauria de proporcionar unes extensions per inserir components de coordinació propis que s'interposarien de manera transparent en crides normals al component col.laboratiu.

En el nostre cas, i tenint en compte que MOVE es va dissenyar originàriament com un entorn virtual col.laboratiu amb fins educatius, les nostres polítiques de control d'accés s'han establert mitjançant la definició de dos rols bàsics, que són *teacher* i *student*. Fent servir aquesta aproximació, els membres del rol *teacher*, tindrien accés total a la manipulació de l'estat dels artefactes compartits, mentre que els membres del rol *student* només podrien observar el resultat d'aquests canvis fets pels professors.

En un futur, seria interessant el desenvolupament d'un sistema de control de la concurrència fent servir un component de reserva de *tokens* (*token-lock component*), per evitar que més d'un usuari pugués manipular un artefacte compartit al mateix temps. Aquest mecanisme tambe permetria als usuaris transportar artefactes des d'un *place* a un altre, assegurant la consistència de les dades entre sessions.

5.3.4.4 Consciència (*Awareness*)

L'*awareness* és una característica important dels entorns col.laboratius. La consciència o *awareness* es pot definir com una enteniment de les activitats dels altres, que proporciona un context de la teva pròpia activitat. Des de la nostra perspectiva, una plataforma de consciència hauria de proporcionar adquisició de dades de l'entorn en execució.

MOVE es beneficia dels serveis de consciència i monitorització d'ANTS CSCW. De fet, ANTS.AWS és la base de la infrastructura d'agents de MOVE. Dins de MOVE és possible programar *bots*[18] que realitzin tasques especials i que reaccionin a esdeveniments produits a dins l'entorn. Com a exemple, hem implementat un robot guia

[18] Més endavant es veurà com s'ha realitzat la programació dels *bots*.

que ensenya el *place* als usuaris nous que entren i que interactua amb els artefactes compartits quan un cert usuari es mou al costat d'ells. Es poden desenvolupar agents més complexos i de fet, els hem utilitzat per simular com es comporta el nostre sistema quan hi ha molts usuaris en un mateix *place*.

Tal i com es pot observar a continuació, el bus s'utilitza per a la propagació de l'estat i per a propòsits d'*awareness*. Per una banda, cada cop que l'usuari es mou, transmet els seus moviments al bus i aquests són rebuts per la resta dels clients en el context compartit (1 i 2). Per altra banda, el servei de monitorització (*Mediator*) escolta el bus (3) i engega els actuadors específics en resposta als esdeveniments filtrats. En aquest cas, el *Mediator* activa el robot guia que comença una visita guiada al *place*.

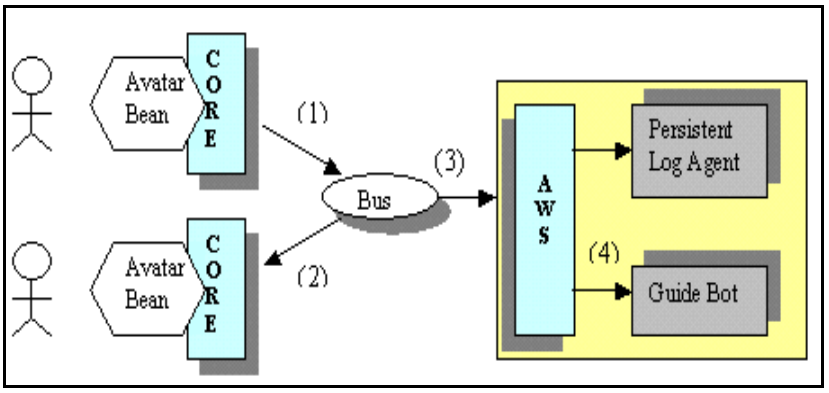


Figura 61. Components de MOVE i bots.

Ara mateix tenim *bots* simples o agents que s'executen, però en un futur aquests es podrien ampliar de manera que es poguessin tenir *bots* guia avançats, *bots* d'ajuda, o fins i tot *bots* que col.laboressin i interactuessin entre si usant protocols de comunicació d'agents.

Com que tots els esdeveniments del sistema es poden emmagatzemar, no seria difícil de construir un sistema de captura i repetició (*Capture & Replay*) per fer que els *bots*, per exemple, repetissin totes les accions que ha realitzat un usuari en una certa sessió. Seria també possible crear fitxers de registre (*log files*) d'aquests esdeveniments o fins i tot crear fitxers de registre específics per a una subscripció per components o rols. Es podrien també aplicar tècniques de *data mining* per extreure informació rellevant d'aquests fitxers de registre.

5.4 El mòdul move-builder

Per tal de crear l'entorn en el qual ens comunicarem farem ús del mòdul **move-builder** que ens ajudarà a crear l'entorn VRML en el que posteriorment interactuarem. Aquest mòdul també ens permetrà personalitzar els nostres *places* per tal que tinguin l'aspecte que realment desitgem; així, podrem afegir dos tipus d'objectes diferents dins d' un *Place*:

- **Objectes estàtics:** Objectes tals com ara cadires, taules, làmpades, etc, que seran carregats sempre al *place* un cop afegits.

- **Models:** Objectes tals com pantalles de transparències, ràdios, equips de vídeo, etc, els quals correspondran als models 3D de les *tools*, i que seran carregats dinàmicament al Place, depenent de si la *tool* a la que representen està activa o no.

L' aspecte que un client veu quan utilitza **move-builder** és el següent:

Figura 62. Aspecte del client **MOVE Builder**.

Per una banda tenim l'entorn 3D que representa el nostre *place*.

Per altra banda tenim els controls que ens ofereix la *tool* per tal d' interactuar amb el *place*.

Internament, però, l'estructura d'aquest mòdul està separada en una part que es troba a la part client i l'altra que està situada al servidor. Anem a veure primerament el diagrama de classes del package *ants.move.builder*.

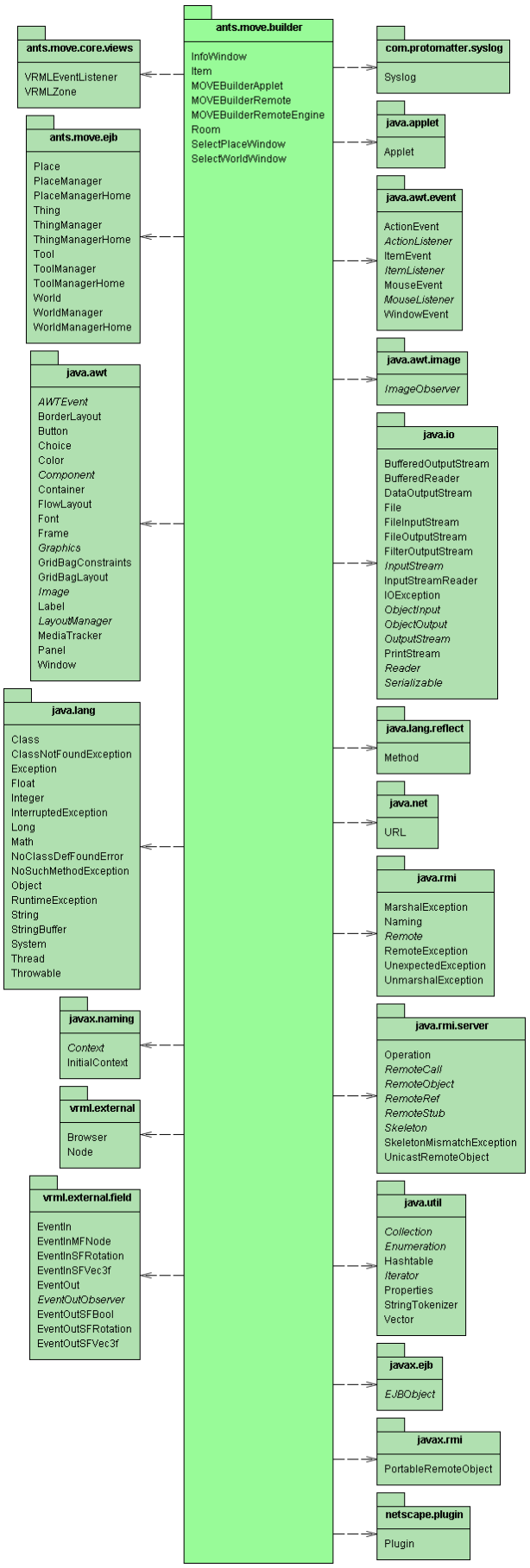


Figura 63. Diagrama de classes del package *ants.move.builder*.

D'aquesta manera, dins el package ens trobem les classes que s'executen només en el client, que són:

- **InfoWindow:** Mostra informació diversa a l'usuari.
- **MOVEBuilderApplet**: *Applet* que interacciona amb l'usuari.
- **SelectPlaceWindow**: Finestra que permet escollir un place destí en un *link*.
- **SelectWorldWindow**: Finestra que permet escollir un world destí en un *link*.

I les que s'executen només en el servidor:

- **MOVEBuilderRemoteEngine**: S'encarrega de garantir la persistència dels canvis.

Per tal de realitzar la comunicació entre el client i el servidor s'utilitza **RMI.** La classe d'interfície de **MOVEBuilderRemoteEngine** és **MOVEBuilderRemote.**

Client i servidor es comunicaran passant un objecte serialitzable de tipus **Room** que contindrà tots els elements de la *place.* Aquests elements seràn objectes de tipus **Item.**

5.4.1 Estructura interna

Cada cop que creem un nou *place* hem de crear el seu fitxer de dades VRML (amb extensió **.wrl*) associat. L'encarregat de realitzar això serà la classe **MOVEBuilderRemoteEngine**. Un cop hem creat la representació 3D del *place* tenim la possibilitat de modificar-ne l'aparença. Per fer-ho executarem el MOVEBuilderApplet. Primer que res, la part de servidor carregarà a un objecte de tipus **Room** tot el contingut de la *place*. Això es farà accedint al mòdul **move-ejb**, que ja hem descrit anteriorment. Tot seguit l'*applet* rebrà via **RMI** aquest objecte de tipus **Room**. En aquest punt, l'usuari ja estarà en disposició de modificar l'aspecte tridimensional del *place* al seu gust.

L'*applet* es comunicarà amb el *browser VRML* mitjançant la interície **EAI**, de la qual n'hem descrit el funcionament anteriorment. L'*applet* permet a l'usuari afegir o treure tant objectes estàtics com models així com afegir *links* cap altres *places.* També permet escalar, rotar i moure els objectes. Un cop haguem modificat el *place* tenim l'opció de guardar-lo.

5.4.1.1 Diagrames de classes d'*ants.move.builder.MoveBuilderRemoteEngine*

Passem a veure detalladament la classe **MOVEBuilderRemoteEngine.** La seva interfície **RMI** és la classe **MOVEBuilderRemote**, de la qual en podem observar el seu diagrama de classes en UML:

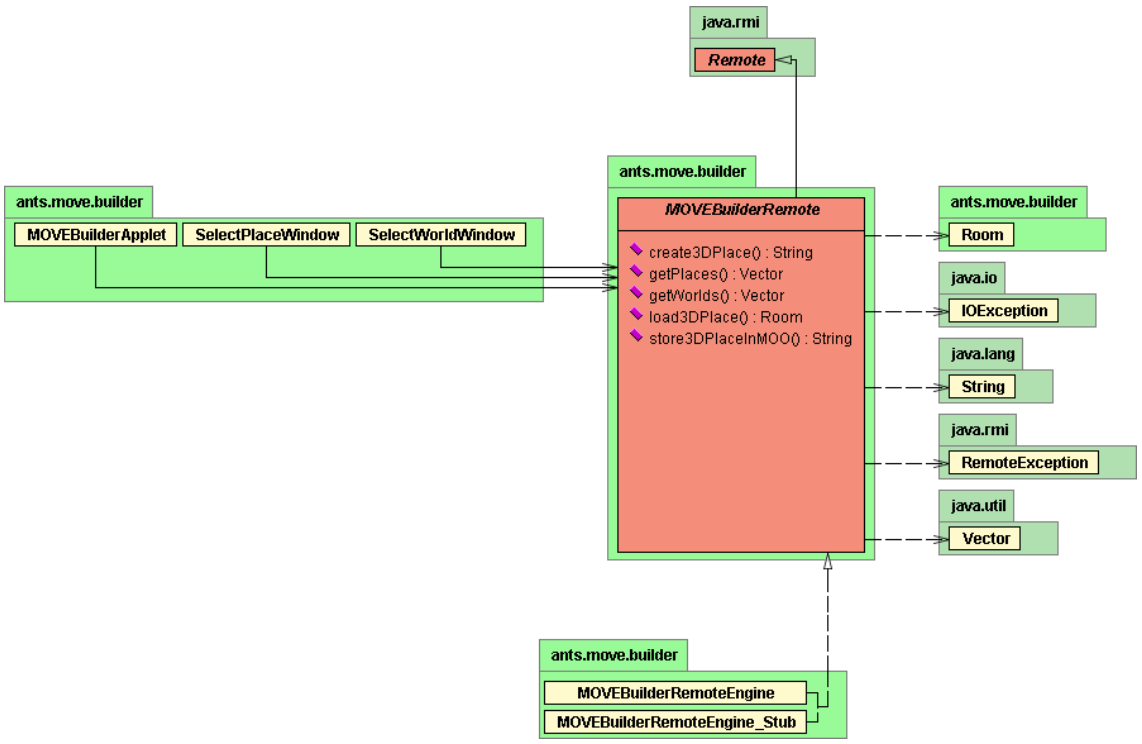


Figura 64. Diagrama de classes d'*ants.move.builder.MoveBuilderRemote*.

Com que es tracta d'una interfície RMI, només conté els mètodes que la classe *ants.move.builder.MoveBuilderRemoteEngine* haurà d'implementar. Per tant, veiem com s'hauran d'implementar tots aquests mèdodes: create3DPlace(), getPlaces(), getWorlds(), load3Dplace(), store3DPlaceInMOO(). El diagrama de classes de la classe d'implementació és el seguent:

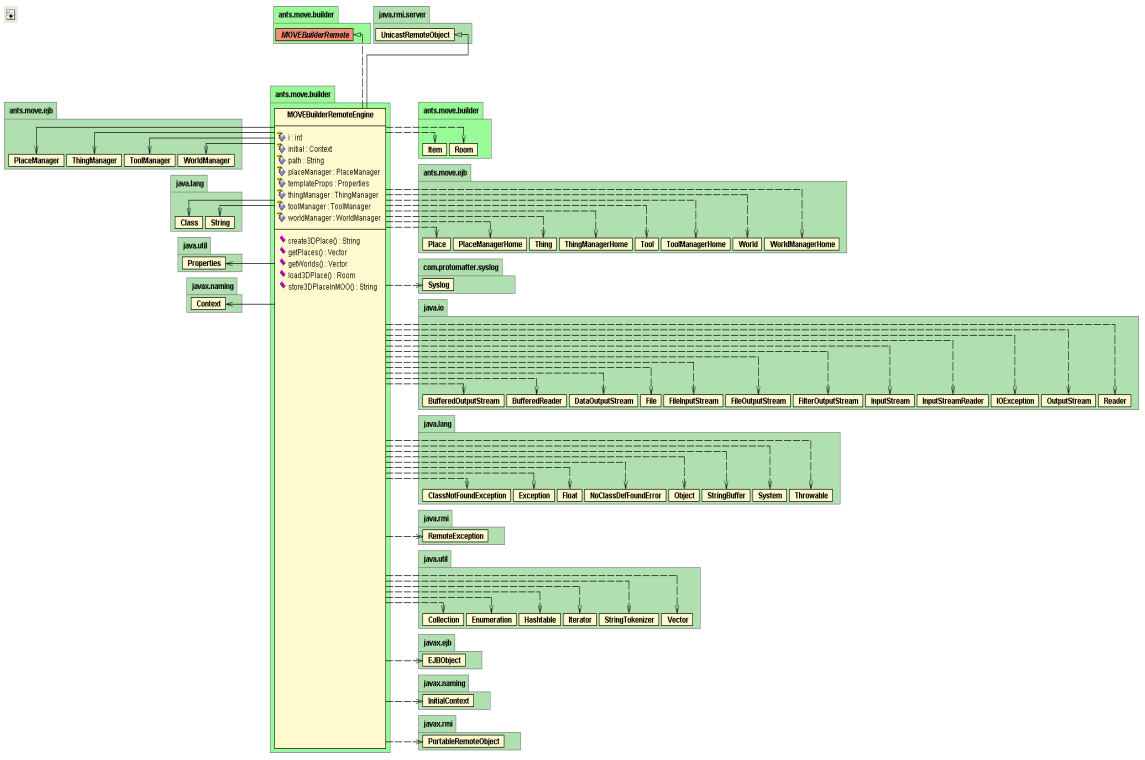


Figura 65. Diagrama de classes d'*ants.move.builder.MoveBuilderRemoteEngine*.

Passem a veure una mica més en detall les funcions més importants que implementa:

- **create3DPlace()**: Aquesta funció es crida a l'hora de crear un nou place. El que fa és crear una *thing* per cada *tool* que existeixi en el sistema. Això ho fem perquè seran aquestes *things* les que guardaran l'estat de la *tool* en el *place* (seran, per tant, els models o artefactes compartits). Seguidament crida el mètode **store3DPlaceInMOO()** passant com a paràmetre aquestes *things*.

- **load3Dplace()**: Aquesta funció és l'encarregada de carregar totes les *things* que hi ha en el place i passar-les a l'*applet*.

- **store3DPlaceInMOO()**: Primer de tot crea una crea un fitxer amb l'extensió **.wrl* que té com a nom la clau del *place*. Aquest fitxer es crea en el directori **/move-app/move-web/worlds/<clau_del_world>/<clau_del_place>/**. Tot seguit guarda les *things* que li han passat per paràmetre al **move-ejb.**

5.4.1.2 Diagrames de classes d'*ants.move.builder.MoveBuilderApplet*

Per tal d'interaccionar amb l'usuari en la part client tindrem el *browser VRML* i un *applet* Java (**MOVEBuilderApplet**). Per poder comunicar-se entre ells s'utilitza la interfície **EAI**. L'*applet* serà, doncs, l'encarregat de proporcionar una interfície amb l'usuari i gestionar tots els canvis que es produeixin en l'entorn. Inicialment l'*applet* es comunicarà via **RMI** amb el **MOVEBuilderRemoteEngine** per tal de carregar el *place*. Tot seguit es cedirà el control a l'usuari per tal que pugui fer els canvis necessaris. El manual de l'usuari es troba disponible on-line. Un cop realitzats els canvis, l'*applet* passa via **RMI** els canvis realitzats.

Per tal de conèixer tots els objectes **VRML** que estan disponibles per introduir en el *place* es disposa d'un arxiu anomenat '**items.properties**'. Aquest arxiu es troba en el directori **/move-app/move-web/rosources/util**. Els fitxers font de tots els objectes es troben en el directori /**move-app/move-web/rosources/things**.

El diagrama de classes en UML es presenta tot seguit.

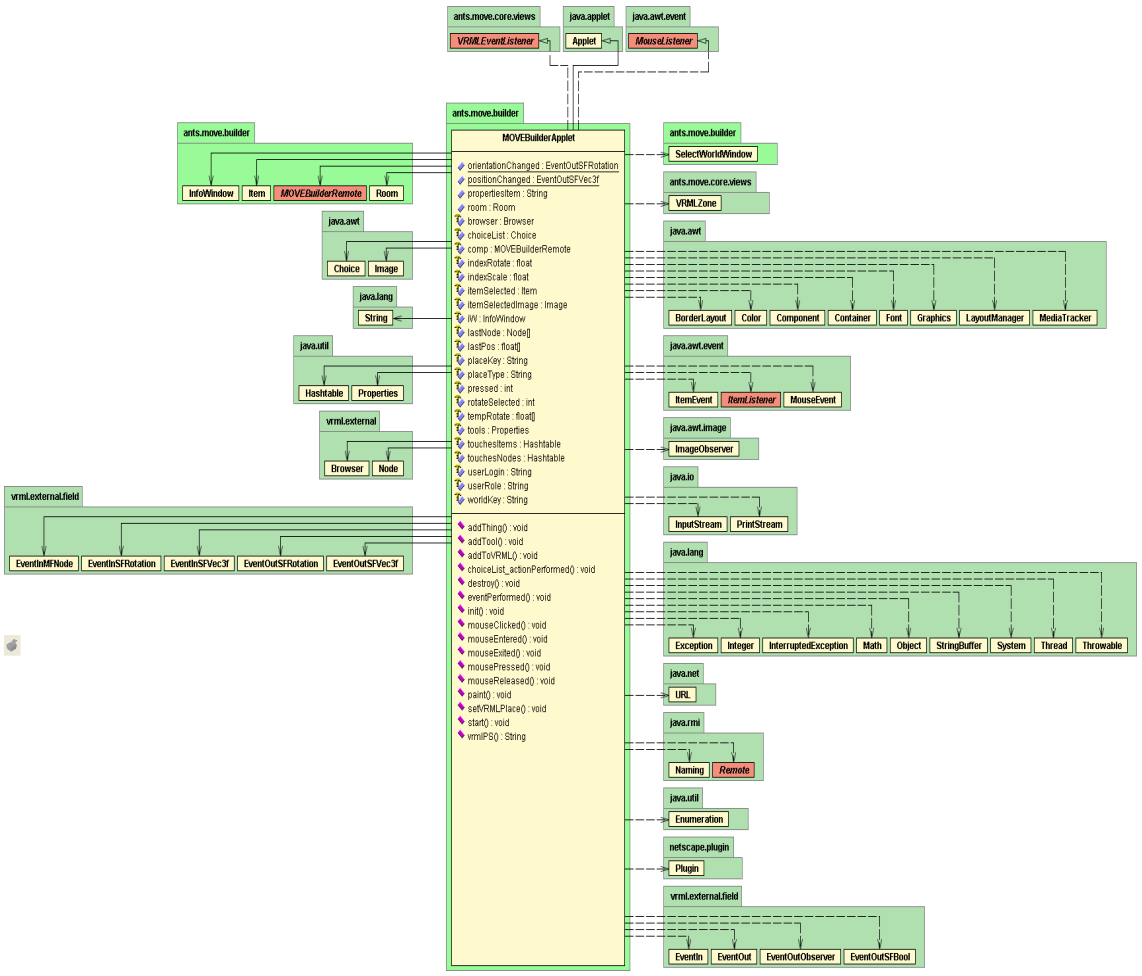


Figura 66. Diagrama de classes d'*ants.move.builder.MoveBuilderApplet*.

Els objectes serialitzables que s'utilitzen per passar les dades de client a servidor i viceversa tenen aquesta estructura:

Objecte *ants.move.builder.Item*: equival a una *thing*.

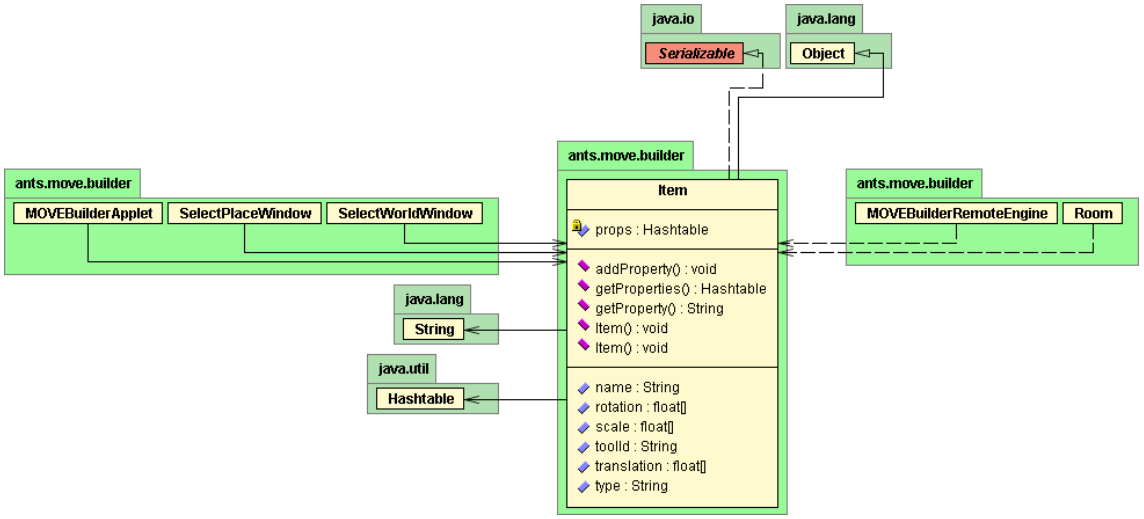


Figura 67. Diagrama de classes d'*ants.move.builder.Item*.

Objecte *ants.move.builder.Room*. Conté tots els *items* i equival a un *place*.

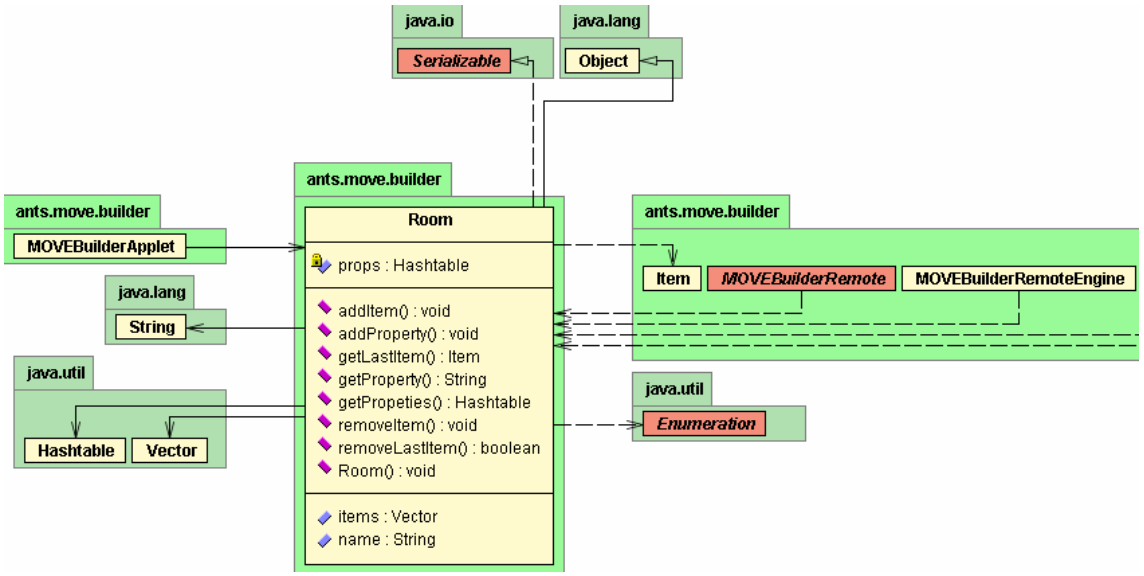


Figura 68. Diagrama de classes d'*ants.move.builder.Room.*

5.5 El mòdul *move-core*

MOVE es basa en una arquitectura modular per cada component (*tools*). Disposem de diferents tipus de components com poden ser eines de votació, transparències, vídeos, documents, arxius multimedia, etc. Aquests components estan basats en un patró del tipus **Model-Vista-Control.lador** (MVC) replicat en cada client. Per tant, cada client disposa d'una còpia local tant del control, que és l'eina que disposa l'usuari per modificar l'estat del component, com del model, que s'encarrega de contenir l'estat del component, i evidentment la vista, que és el resultat que l'usuari veu per pantalla. Aquesta arquitectura distribuïda permet a tots els usuaris poden modificar l'estat del component gràcies a que tenen accés al control.

Per tal de mantenir la consistència entre tots els clients a l'hora de mantenir l'estat del component, s'utilitza ANTS.CORE (tal i com ja hem vist anteriorment), que utilitzant un servei de notificació per subscripció fa possible que els models de tots els clients estiguin convenientment actualitzats i siguin consistents. El fet d'usar un patró MVC per cada component i replicat en cada client ens aporta una sèrie d'avantatges que podem resumir en: facilitat a l'hora de crear nous components i alt rendiment respecte altres solucions no replicades.

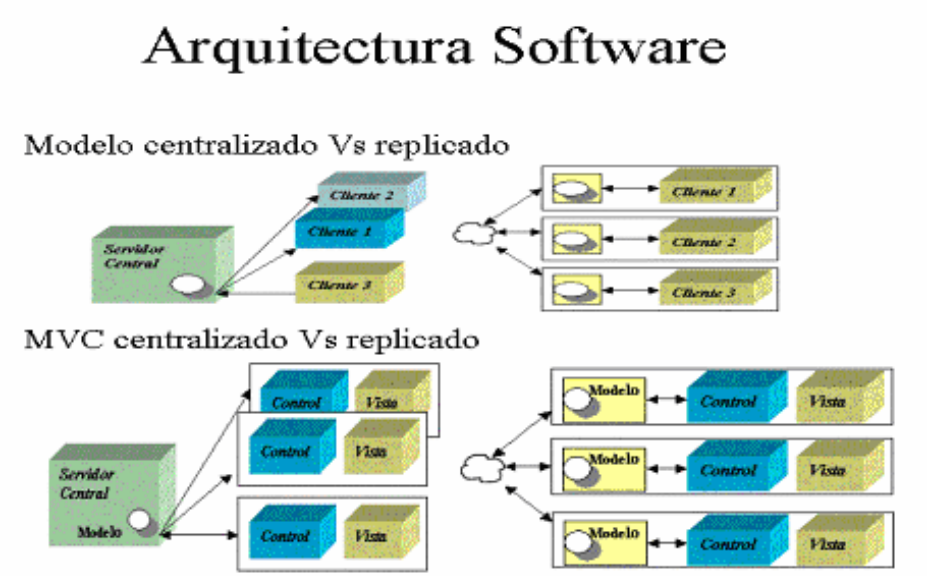


Figura 69. Comparativa entre un model centralitzat i un de replicat.

Utilitzar un MVC replicat permet una millora notable del rendiment respecte les solucions centralitzades ja que per tal de conèixer l'estat del component no cal realitzar cap comunicació a través del bus, degut a que cada client conté una còpia del model del component. Per tant, només transmetrem informació a través del bus en cas de modificació del component. Així doncs, utilitzant aquesta arquitectura es minimitza la transmissió a través del bus.

L'altre gran avantatge d'utilitzar un MVC replicat és la facilitat a l'hora de crear nous components. Gràcies al patró MVC l'estructuració del component és molt més elevada i això permet una localització molt ràpida del errors i la possibilitat de desenvolupar totes les parts d'un mateix component de forma paral·lela. Aquest tipus de

patró també ens permet posseir un sol model del component, però en canvi diversos tipus de control del component o fins i tot diverses vistes del component. Això és molt important ja que en un cas que tinguéssim diferents usuaris amb diferents privilegis podríem tenir un control per a cada rol d'usuari i d'aquesta manera cada usuari disposaria de les funcions especifiques que li estan permeses d'efectuar d'una manera ràpida i senzilla. De la mateixa manera també podríem tenir diferents vistes per a un mateix component: per exemple, podríem tenir una vista en 2D i una en 3D del mateix component.

Per tal de transmetre la informació d'actualització de l'estat dels components s'utilitza el mòdul **ants-core**, que ens proporciona el marc de treball ANTS CSCW. Tots els esdeveniments es retransmeten pel mateix bus però només responen a ells els components consumidors d'un esdeveniment determinat. És per això que podem determinar que l'actualització de l'estat dels components es basa en un model productor-consumidor. El funcionament és molt simple: en un primer moment quan un usuari canvia l'estat del model d'un component, per exemple, si tenim un component que és un projector de transparències en l'estat d'aquest model tindríem emmagatzemada la informació de quina transparència està carregada en aquell moment. Un canvi d'estat del component es realitzaria quan un usuari canvia la transparència. En aquest moment el model del component projector de transparències es converteix en productor d'un event de canvi de transparència. Tot seguit aquest event serà retransmès pel bus i tots els models que per defecte son consumidors dels events generats pel model del projector, actualitzaran el seu estat de manera que tots els clients percebran el canvi de transparència.

Move-core és la part principal situada en el client. La seva funció serà carregar tots el models, vistes i controls de les diferents *tools*, així com posar a disposició de l'usuari l'entorn 3D per tal que hi interactui. Degut a que els clients de MOVE s'executen en un navegador Web, la classe principal de **move-core** serà un *applet* (*ants.move.core.MoveCoreApplet*)**.** També disposarem del **browser VRML** que és la capa de presentació i és on veurem l'entorn 3D. Per tal de comunicar l'*applet* amb el **browser VRML** s'utilitza la interfície **EAI.**

Per crear una unitat funcional mínima es distribueixen el nucli de **move-core** i les *tools* separadament. Així, podem crear la nostra distribució de MOVE amb més o menys *tools* i afegir-ne de noves de manera senzilla. És per això que s'ha establert que la unitat funcional mínima conté l'*applet*, les classes necessàries per carregar els models, les vistes i els controls de les *tools*, el model i la vista del *chat* i finalment el model i la vista dels avatars.

5.5.1 Diagrames de classes d'*ants.move.core*

L'estructura del package *ants.move.core* es completa amb les classes incloses en *ants.move.core.views*, *ants.move.core.teacher* i *ants.move.core.models*. Com hem esmentat anteriorment, MOVE utilitza una arquitectura MVC. És per això que les funcions d'aquests packages són:

- **ants.move.core:** Està composat bàsicament per l'*applet*. Com ja és sabut, un *applet* és una petita aplicació que s'executa en un navegador Web i que té certes

restriccions de seguretat. En aquesta classe es duen a terme les operacions d'inicialització de l'entorn del client. La seva tasca principal serà la de fer la captura del Browser VRML per tal de tenir accés a les dades que componen l'entorn VRML. A més és l'encarregat de cridar els mòduls que realitzaran la càrrega dels models (*ants.core*), vistes, (*ants.move.core.views*), i controls (*ants.move.core.teacher*). A part, inicialitzarà els components per tal de carregar els avatars (*ants.move.core.model.avatar*), així com el chat. El seu diagrama de classes en UML és el que es mostra a la pàgina següent.

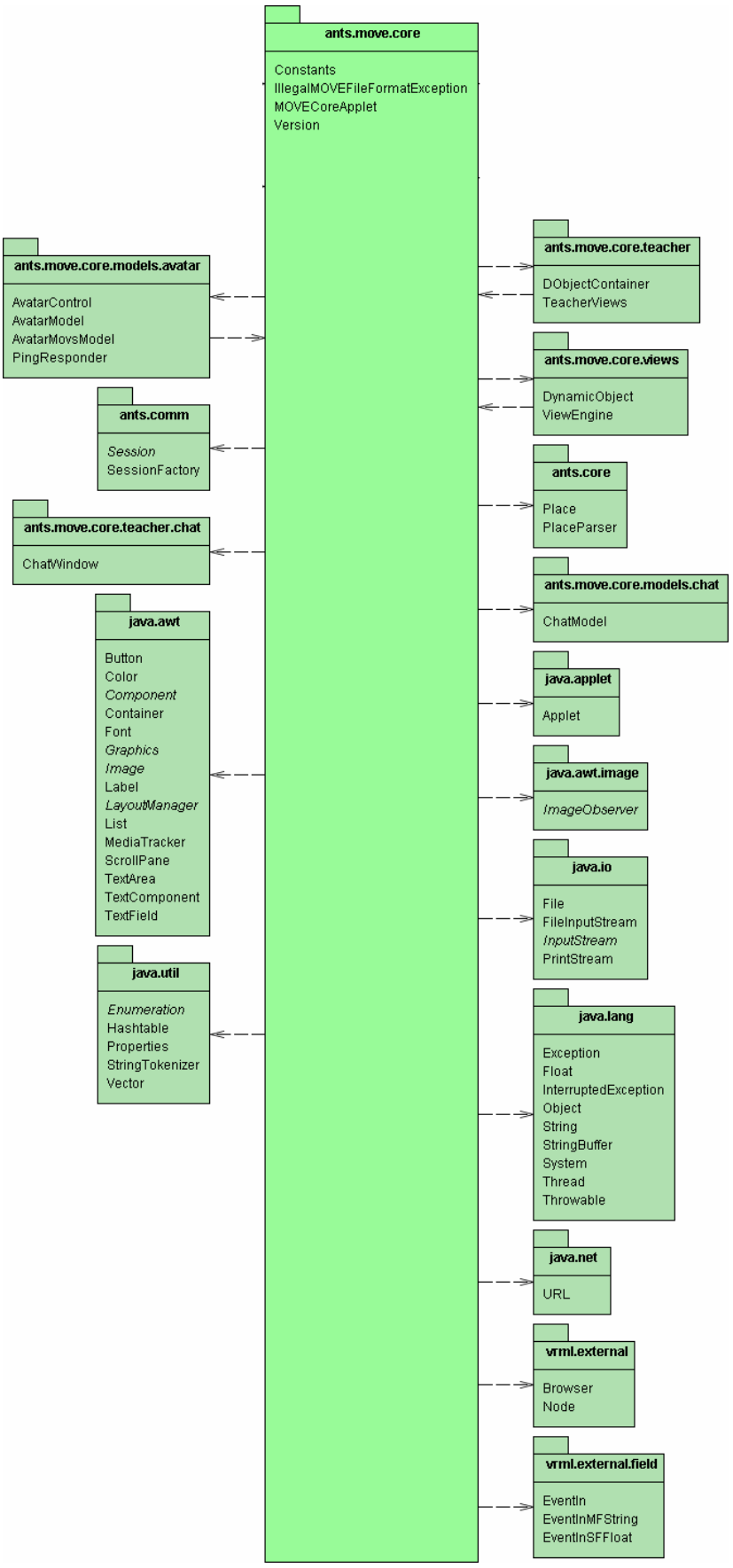


Figura 70. Diagrama de classes del package *ants.move.core*.

- **ants.move.core.views:** Dins d'aquest package hi trobem les classes necessàries per inicialitzar les vistes dels diferents components. Hem de tenir en compte que ens podem trobar components (*tools*) de molts diferents tipus. Podem trobar-nos components que no tenen representació 3D, de manera que no es farà res. Per als components que disposen de representació 3D, aquest package és l'encarregat d'instanciar-ne les classes que componen la vista i seran aquestes les que inseriran l'objecte 3D dins l'entorn. El diagrama de classes UML del package és el següent:

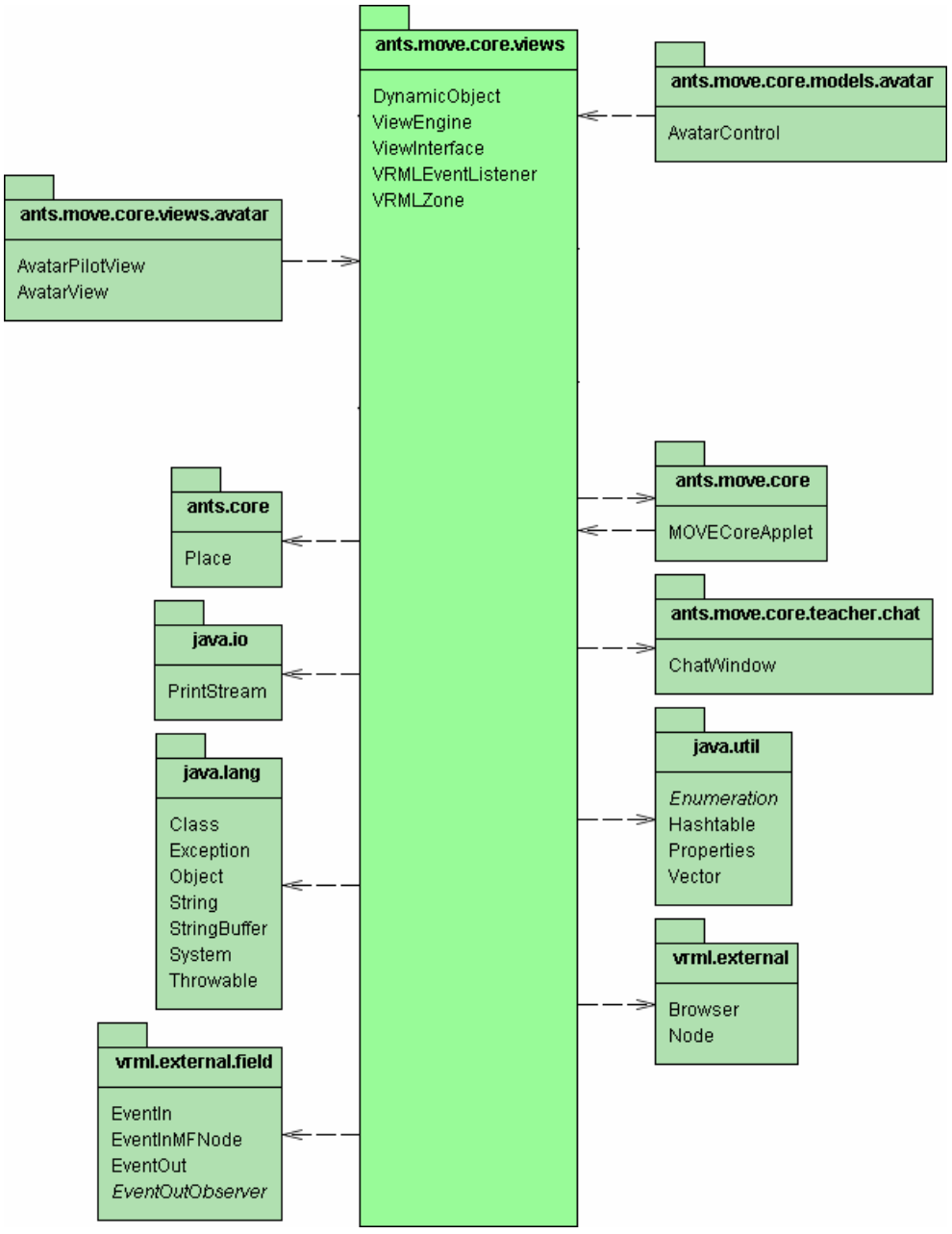


Figura 71. Diagrama de classes del package *ants.move.core.views*.

- **ants.move.core.teacher:** Aquest package és l'encarregat de carregar els controls dels diferents components que estan actius en el *place*. Hem de tenir en compte que només es carregaran els controls dels components en el cas de tenir permís

per utilitzar-los. A més, degut a la gran varietat de components existents només serà necessari carregar els controls dels components que no disposen d'una vista en 3D. Això és degut a que en els components que disposen de vista en 3D, serà l'usuari quan premi sobre el component el que donarà l'ordre de carregar el control del component i per tant, serà la mateixa vista la que carregarà el control corresponent. Per tant, en els casos que no es disposa de vista 3D serà necessari posar a disposició de l'usuari un mecanisme per mostrar el control del component. És en aquests casos quan aquest package té sentit. La seva utilitat és la de carregar els controls dels components que no disposen de vista 3D. Aquests components actualment són la *camera* i el *hook*. En la pantalla inferior esquerra de l'applet apareixeran uns checkboxes que permetran mostrar el control del component. El diagrama UML de classes corresponent al package és el següent:

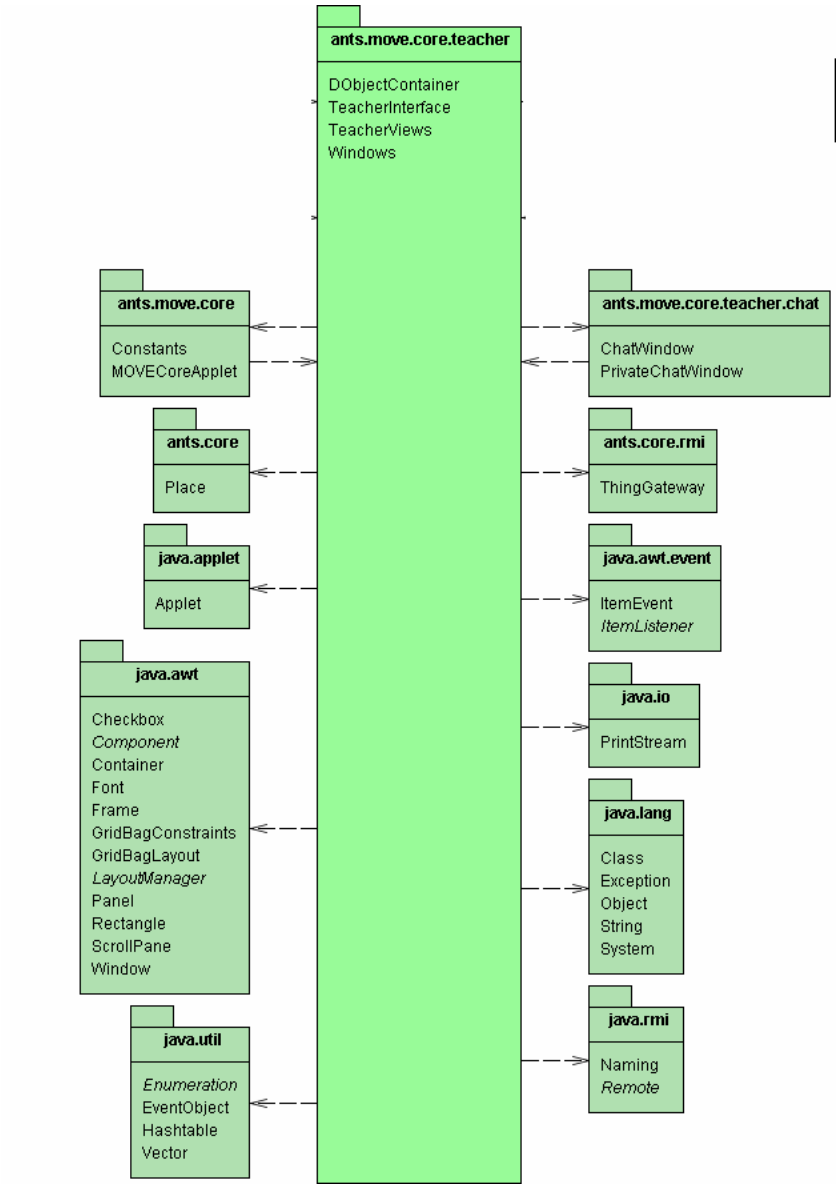


Figura 72. Diagrama de classes del package *ants.move.core.teacher*.

5.5.2 Diagrames de classes dels Avatars

Els avatars componen una de les parts més importants del sistema ja que són les representacions tridimensionals dels altres usuaris que es troben en el mateix *place*. Els packages que fan referència a la gestió dels diferents avatars són *ants.move.core.models.avatar* i *ants.move.core.views.avatar*. En el primer disposarem de tot el necessari per a la gestió i en el segon tindrem les classes necessàries per la seva representació.

El diagrama de classes en UML del package *ants.move.core.models.avatar* és el següent:

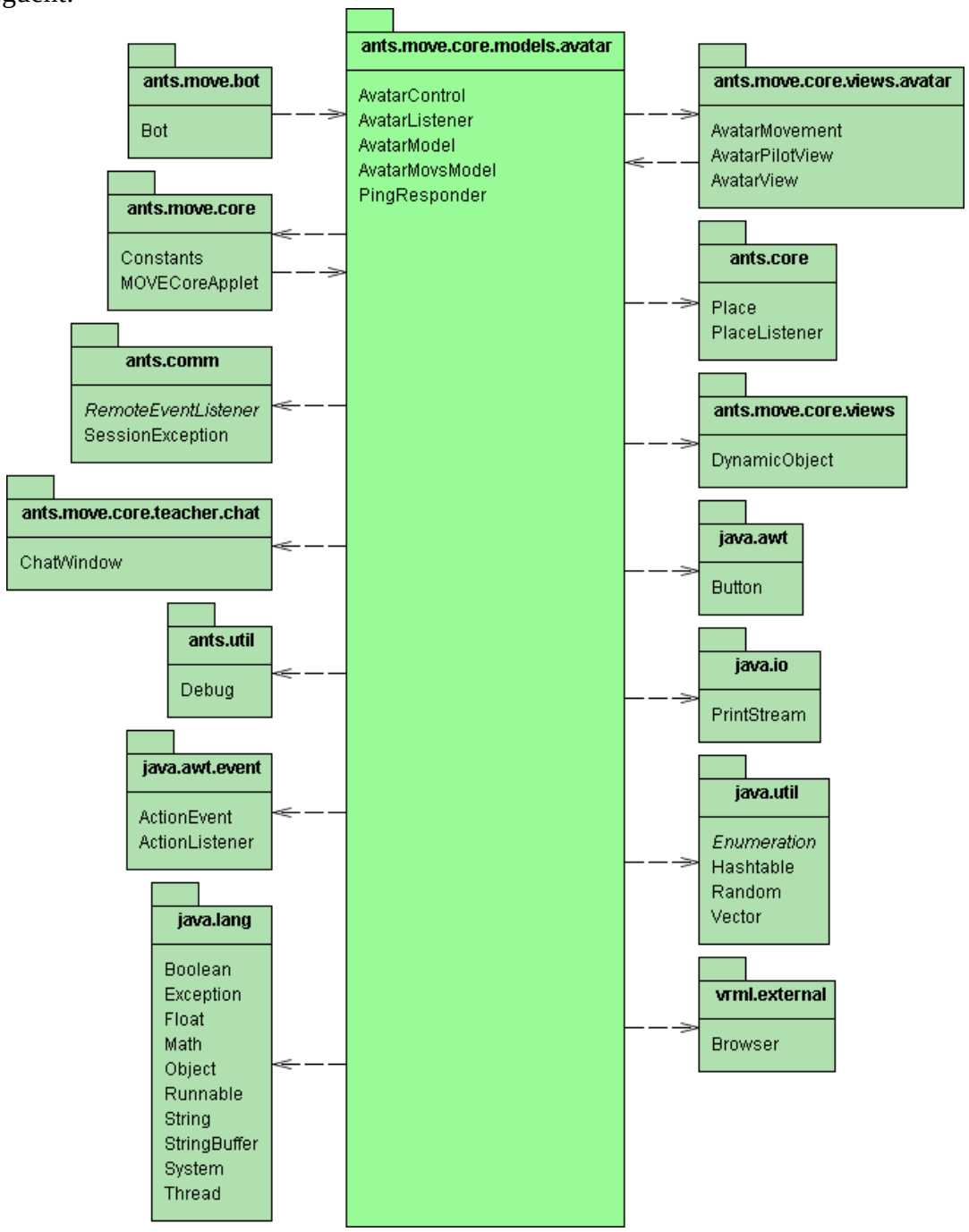


Figura 73. Diagrama de classes del package *ants.move.core.models.avatar*.

Dins d'aquest package (*ants.move.core.models.avatar*) disposarem de les classes necessaries per tal de gestionar els avatars. Les classes que componen el package són les següents:

- *AvatarControl*
- *AvatarListener*
- *AvatarModel*
- *AvatarMovsModel*
- *PingResponder*

Disposarem d'una única instància de la classe *AvatarControl*. Aquesta classe és l'encarregada de consumir tots els esdeveniments de moviment que produeixen els altres clients. Hem de tenir en compte que aquests esdeveniments han passat anteriorment pel *proxy* de **move-server,** cosa que farà que no sigui necessari descartar esdeveniments inútils, ja que aquests (si n'hi havia), ja hauran sigut descartats. D'igual manera, tampoc caldrà gestionar tots els altres esdeveniments relacionats amb els avatars. D'aquesta manera usant una única subscripció es consumiran tots els esdeveniments que produeixen els altres clients. Una altra solució hagués estat crear un model per a cada client i que aquests consumissin els esdeveniments produïts pel seu client però per qüestions de rendiment s'ha optat per la primera opció.

Disposarem d'una instància a la classe *AvatarModel* associada a cada avatar, on es guardarà la informació referent a l'avatar: posició, nom... *AvatarListener* serà la interfície que hauran d'implementar les vistes dels avatars per tal que el model pugui actualitzar-les. *PingResponder* serà la classe encarregada d'informar al servidor que el client està actiu. En cas que el client es bloquegés, la classe *PingResponder* deixaria de respondre als esdeveniments que li llença el servidor (a través de la classe *ants.core.rmi.UserGateway*, tal i com ja hem vist anteriorment) i el servidor avisaria a tots els clients que hi ha un client que ha deixat de respondre. Finalment, la classe *AvatarMovsModel* gestiona tots els esdeveniments que es produeixen quan els avatars realitzen moviments gesticulats.

Hem de distingir dos tipus d'avatars, l'avatar de tipus **pilot**, que serà la representació del propi client i la resta d'avatars (que no són pilot). L'avatar **pilot** no disposa de representació 3D, però per tal de saber la posició del client s'utilitza un sensor de proximitat (node *ProximitySensor* proporcionat pel llenguatge VRML). Aquesta part es detallarà posteriorment.

El diagrama UML de classes del package *ants.move.core.views.avatar* és el següent:

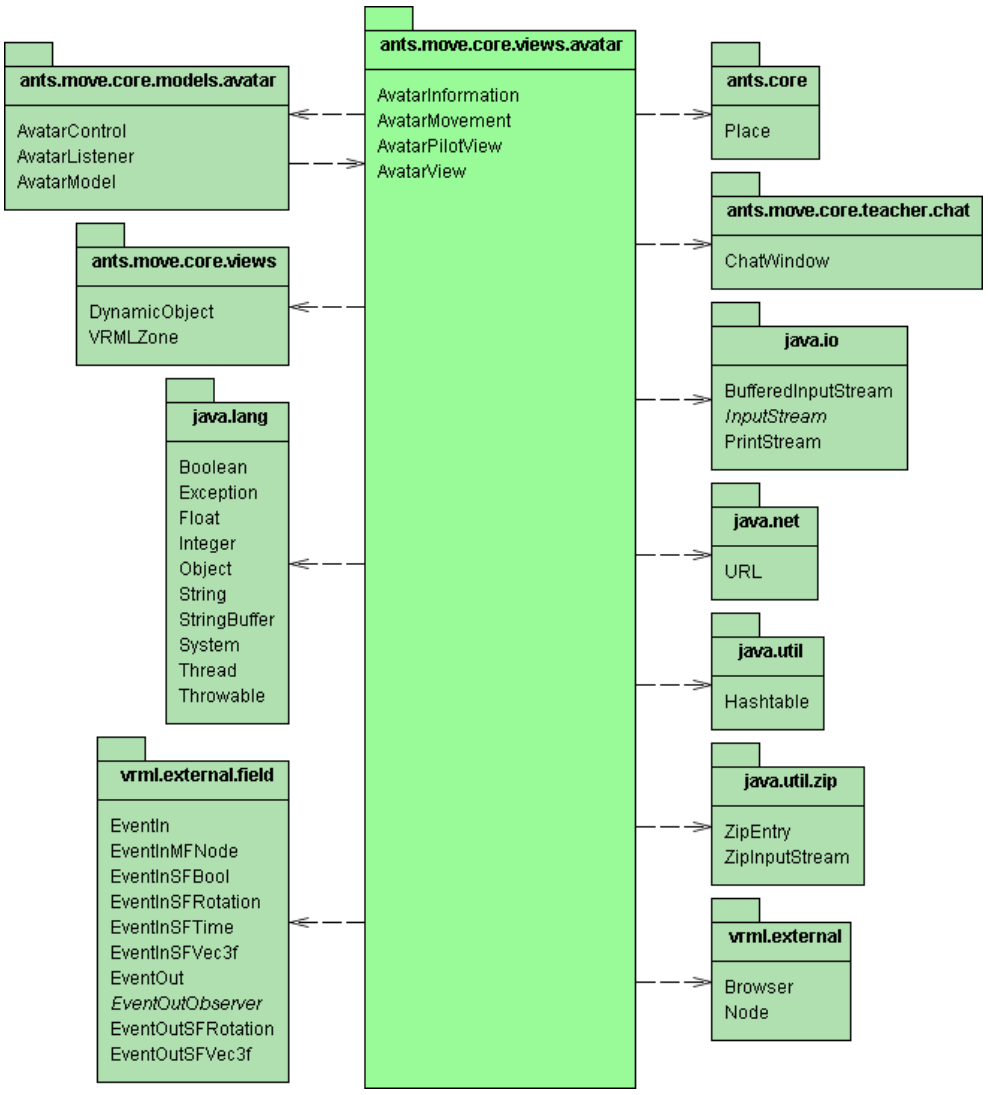


Figura 74. Diagrama de classes del package *ants.move.core.views.avatar*.

Dins d'aquest package hi trobem les següents classes:

- *AvatarInformation*
- *AvatarMovement*
- *AvatarPilotView*
- *AvatarView*

La classe *AvatarInformation* manté la informació referent als nodes VRML que indiquen la posició, orientació i temps de moviment dels avatars. La classe *AvatarMovement* s'executarà en un *thread* diferent i serà l'encarregada d'anar movent els avatars gràcies a la informació emmagatzemada a *AvatarInformation*. Disposarem d'un únic *thread* que realitzarà tots els moviments de tots els avatars. El seu funcionament serà el següent: cada cop que un avatar pilot es mou, la seva classe model associada llença un esdeveniment. Tot seguit, aquest esdeveniment és capturat pel *proxy* de **move-server**. Si l'usuari consumidor és prou aprop, el *proxy* li enviarà l'esdeveniment. Seguidament, *AvatarControl* consumirà l'esdeveniment i avisarà a la vista de l'avatar en qüestió: *AvatarView*. Tot seguit comunicarà la nova posició al model de l'avatar *AvatarModel* i actualitzarà *AvatarInformation* amb els nous valors. Aquest

s'encuarà a *AvatarMovement* i el *thread* es despertarà ja que tindrà un nou moviment per tractar. A continuació, si veu que no té més moviments encuats, el *thread* es tornarà a adormir.

S'ha de tenir en compte que en realitat es distingeixen dos tipus d'avatars. Existeixen els avatar **pilot** i els avatar **no-pilot**. Els avatars pilot representen a un usuari real connectat. Per entendre'ns, un avatar pilot representa cadascun dels clients que estan connectats al nostre sistema. Així doncs, si tenim, per exemple, 4 usuaris simultanis en una zona (suposant que cadascun es troba en una màquina diferent), hi haurà 4 avatars pilot, un en cada client. Els avatars no-pilot representen als altres avatars pilot en el món d'un client. D'aquesta manera, si continuem amb l'exemple anterior, un client representa un avatar pilot i en la seva representació del món tindrà 3 avatars no-pilot que representen als altres usuaris connectats. Una altra característica que diferencia als dos tipus d'avatars és que els pilot no tenen representació gràfica, mentre que els no-pilot sí que en tenen i a més se'ls veu com es mouen. Per aclarir conceptes, en l'exemple anterior hi hauria un total de 4 avatars pilot (1 en cada client) i 12 avatars no-pilot (3 en cada client) distribuits entre les 4 màquines connectades entre sí.

Com hem vist anteriorment disposem de dos tipus diferents d'avatars: els pilot i els no pilot. On es fan evidents les seves diferències és en la seva vista. Un avatar no pilot ve representat per un objecte VRML, mentre que un avatar pilot no té representació 3D, sinó que és representat pel moviment de la camera del client.

La classe *AvatarView* és la classe que utilitzem per tractar la vista dels avatars no pilots. Disposem d'una instància de la classe per cada client que existeixi en el mateix *place* menys per la que representa l'usuari. La interacció entre el browser VRML i la classe es realitza mitjançant EAI. Per tal de representar els avatars s'ha seguit l'especificació **H-Anim v1.1**. Aquests avatars poden estar animats de manera que mouen braços i cames quan es mouen. Per tal de facilitar la creació de nous avatars s'ha afegit compatibilitat amb els avatars creats pel programa **AvatarStudio 2.0**. Si volem un major rendiment a canvi de menors prestacions visuals també es disposa de la possibilitat de que la representació dels avatars sigui un objecte VRML qualsevol.

Figura 75. Un avatar compatible amb H-Anim v1.1.

Per un altre costat ens trobem amb l'avatar pilot, la classe que utilitzarem serà *AvatarPilotView***.** Com hem esmentat anteriorment la representació de l'avatar pilot serà la camera de l'usuari. Però com es capturen els esdeveniments de moviment de qualsevol avatar pilot ? Doncs bé, disposem d'un element dins el codi *VRML* que ens serveix per a aquest fet. Aquest element és el *ProximitySensor* PS, el codi *VRML* del qual és aquest:

```
DEF PS ProximitySensor {
        center 0 0 0
        size 10000 10000 10000
        enabled TRUE
}
```

El *ProximitySensor* actua de forma que captura tots els esdeveniments de moviment d'un usuari en una zona cúbica de 10.000 metres de costat (suficient per a qualsevol *place*). La classe *AvatarPilotView* captura, mitjançant EAI els esdeveniments llençats per aquest *ProximitySensor* i pot saber en tot moment quina és la posició de l'avatar pilot. Per tal de no enviar masses esdeveniments només s'envien als altres clients els esdeveniments de moviment que superen en 0.4 m (o 0.2 radians en l'orientació) en qualsevol direcció l'anterior posició de l'avatar pilot. Disposem de la possibilitat de modificar aquests nombres modificant els paràmetres **positionResolution i rotationResolution** que estan situats en el fitxer de propietats '**move.properties**' del directori '**/move-app/move-web/**'.

5.5.3 Diagrames de classes del Chat

Per acabar, l'últim component que inclou **move-core** és el Chat. Aquest component el podem apreciar a la part inferior de la pantalla. Tal i com el seu nom indica és un chat que ens permet comunicar-nos amb els altres usuaris dins el mateix *place*. El diagrama de classes UML del package *ants.move.core.models.chat* és el següent:

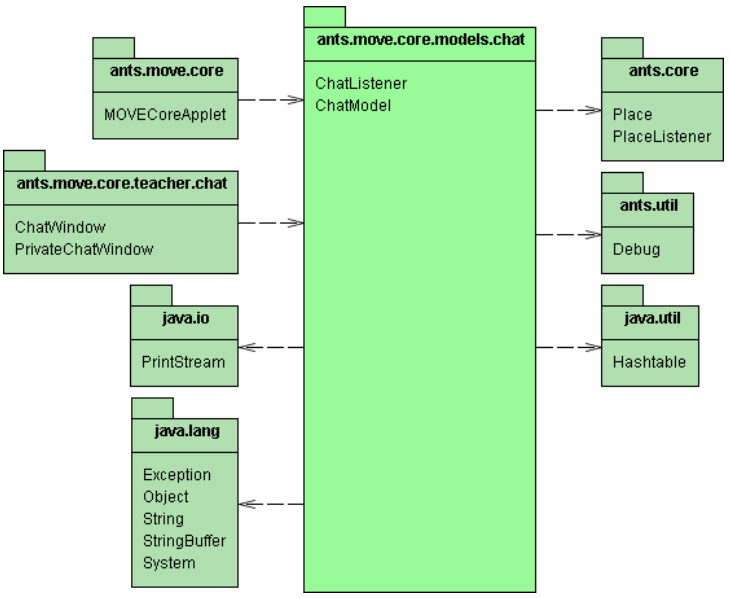


Figura 76. Diagrama de classes del package *ants.move.core.models.chat.*

Mentre que el diagrama de classes del package *ants.move.core.teacher.chat* és:

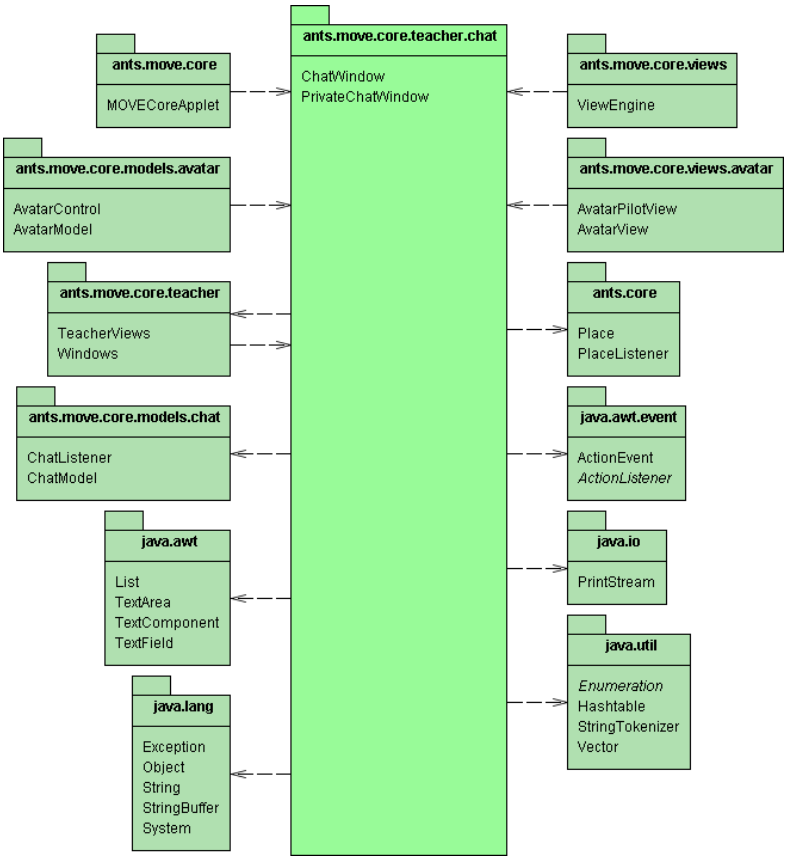


Figura 77. Diagrama de classes del package *ants.move.core.teacher.chat.*

La classe *ChatModel* és la que llença els esdeveniments i la classe *ChatWindow*, que és la que escriu els missatges a la finestra, és la que els consumeix.

El funcionament és el següent: quan un usuari escriu un missatge, ho fa en la *ChatWindow*. Quan prem la tecla **<ENTER>**, la instància d'aquesta classe invoca el mètode `send_msg()` de la classe *ChatModel*, i des d'aquesta classe, es genera l'esdeveniment que reben les altres instàncies de *ChatWindow* que hi ha en els diversos clients connectats.

Com que l'únic esdeveniment que es llença és el contingut d'un missatge, l'ús d'un camp *action* és del tot innecessari. Per enviar el missatge, s'utilitza un camp anomenat *msg*.

Tot això es pot veure reflectit en la següent figura:

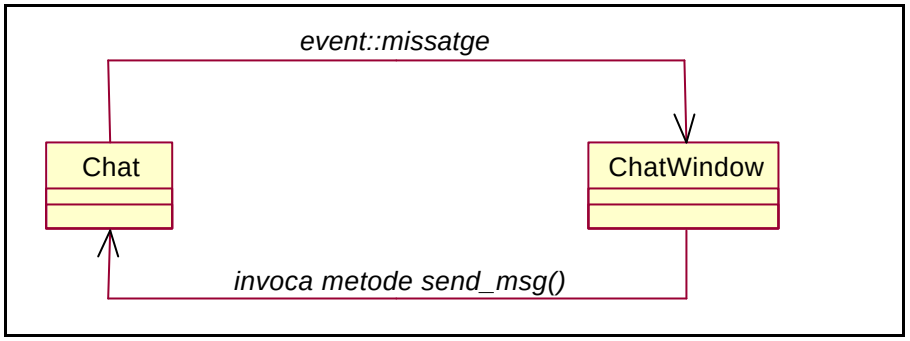


Figura 78. Flux d'esdeveniments del Chat.

5.6 El mòdul *move-server*

Tal i com ja hem comentat anteriorment, existeix un component que s'encarrega del filtratge dels esdeveniments de moviment generats pels clients. Aquest component és l'anomenat *move-server* i la seva funció, doncs és filtrar els esdeveniments de moviment que produeixen els avatars quan caminen. Per tal de millorar el rendiment de l'aplicació s'ha fet que els usuaris només puguin veure els avatars més propers a ells dins d'un cert llindar. És per això que no cal que un usuari rebi els moviments de tots els usuaris del sistema sinó que només cal fer-li arribar els moviments creats pels usuaris que té a prop seu. Si no es prenguessin mesures per filtrar els esdeveniments de moviment, degut a la gran quantitat que se'n produeix, això arribaria a saturar els clients, sense tenir en compte el degradatge del rendiment global de l'aplicació.

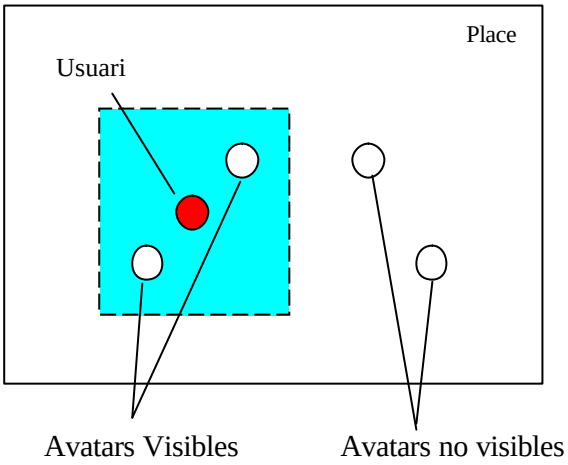


Figura 79. Només dels avatars que estiguin en l'àrea d'influència se'n rebran els events de moviment.

Per tal que un esdeveniment es propagui fins un client, l'avatar generador de l'esdeveniment de moviment ha d'estar dins l'àrea d'influència de l'avatar que rep l'esdeveniment. Aquesta àrea d'influència ve determinada pel paràmetre *influenceArea*, expressat en metres, situat a l'arxiu de configuració '**move.properties**'.

Ara que ja coneixem la utilitat d'aquest package, anem doncs a veure'l en detall.

5.6.1 Diagrames de classes i funcionament

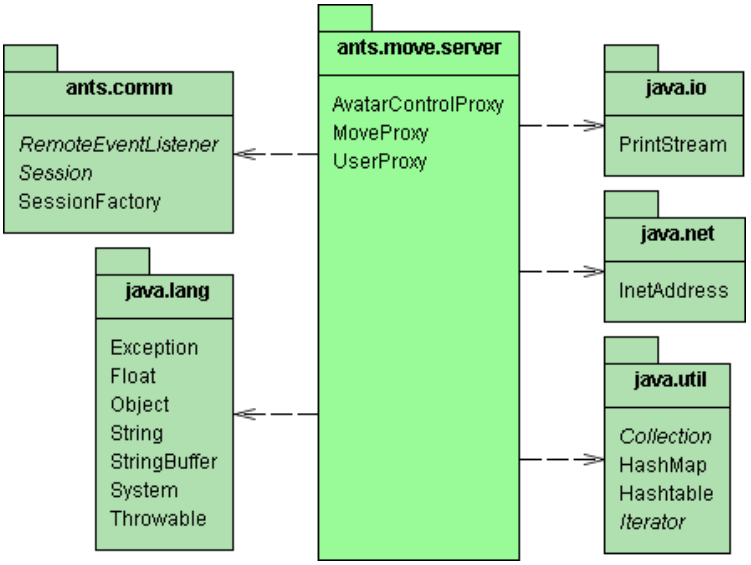


Figura 80. Diagrama de classes del package *ants.move.server*.

Com es pot veure, aquest package és completament independent del servidor d'aplicacions. Això és degut a que l'únic que realitza és la captura dels esdeveniments que produeixen els avatars i la retransmissió als clients interessats.

Es composa de 3 classes:

- **MoveProxy:** Executable del package, que es connecta al servei de notificacions i crea una instància de la classe *AvatarControlProxy.*

- **AvatarControlProxy:** Captura els esdeveniments d'entrada i sortida d'un usuari en un place, a l'igual que els de moviment d'un avatar. Gestiona tots els places existents alhora.

- **UserProxy:** Existeix una instància per a cada usuari. S'actualitza amb la posició en que es troba l'usuari cada cop que es mou i en funció d'aquesta li retransmet a l'usuari els moviments dels altres.

Un esquema del funcionament bàsic d'aquest package vindria expressat en el següent diagrama:

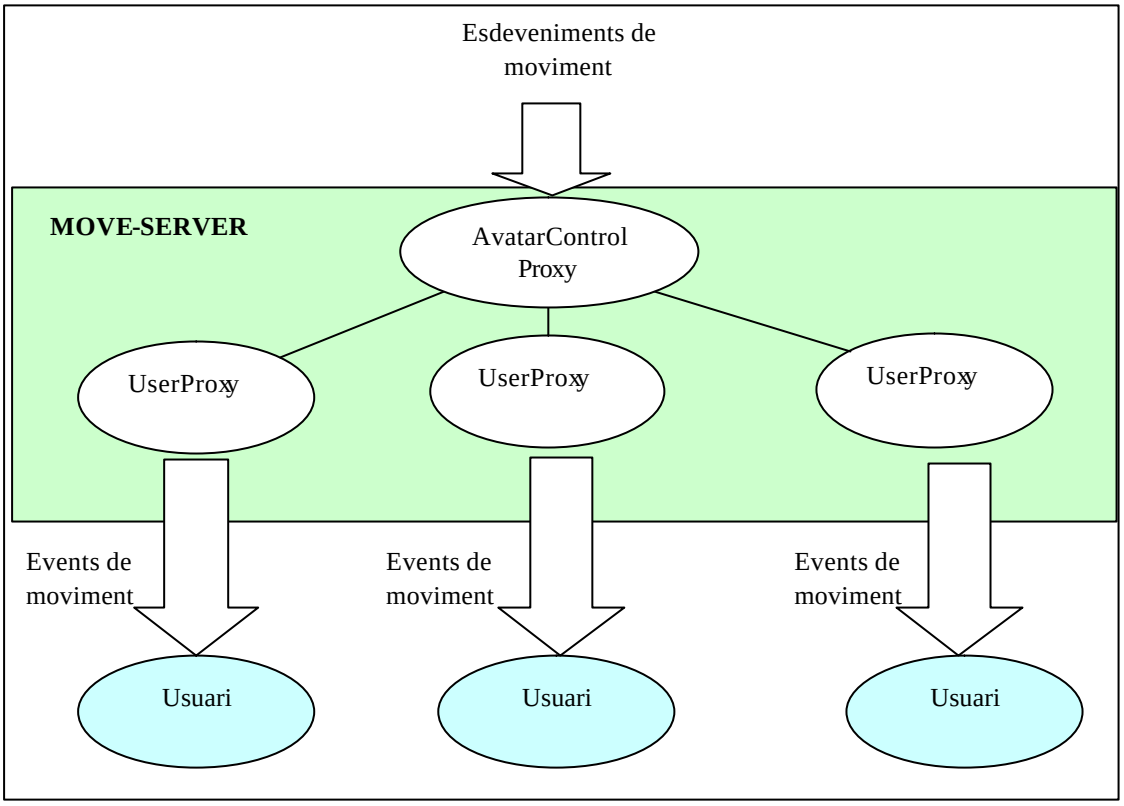


Figura 81. Esquema de funcionament d'*ants.move.server*.

La classe **AvatarControlProxy** captura tots els moviments que produeixen tots els avatars. Aquests esdeveniments tenen com a estructura aquest paràmetres:

```
CLASS = AVATAR
SRC = MOVE
```

A part d'aquests camps (iguals en tots els esdeveniments) també s'inclou del nom de l'usuari que ha produït l'esdeveniment i el *world* i *place* en que es troba. A més també captura els esdeveniments que es produeixen quan un usuari entra al sistema, de manera que es crea una instància de la classe *ants.move.server.UserProxy***.** També captura els esdeveniments de sortida d'un usuari del sistema, de manera que llavors destrueix la instància de la classe *ants.move.server.UserProxy* de l'usuari corresponent, creada anteriorment.

Cada cop que es produeix un moviment d'un usuari, la classe *ants.move.server.AvatarControlProxy* en captura l'esdeveniment i verifica que l'usuari que l'ha creat sigui vàlid (ha de tenir una instància de la classe *ants.move.server.UserProxy*) i llavors actualitza la instància d'*UserProxy* del propietari de l'esdeveniment amb els valors de la posició actual. Tot seguit recorre tots els usuaris (totes les instàncies de la classe *ants.move.server.UserProxy*) que es troben dins el mateix place. A continuació es comprova si l'esdeveniment s'està produint dins l'àrea d'influència de l'usuari propietari del proxy. En cas negatiu no es fa res i en cas afirmatiu es retransmet l'esdeveniment al client amb l'estructura següent:

```
SENDER = MOVEPROXY
OWNER = <login del destinatari de l'event>
```

METHOD = **move**

A banda d'aquestes dades, també s'envien els camps com ara el *world*, el *place*, el productor de l'esdeveniment i la posició.

Degut a la utilització d'àrees d'influència ens podem trobar amb el problema que un avatar entri i marxi de l'àrea d'influència d'un usuari. Per tal de controlar què fer en aquests casos la classe *UserProxy* serà l'encarregada de detectar-ho i enviarà els esdeveniments corresponents per tal d'esborrar l'avatar, en el cas de sortir, o pintar-lo, en el cas d'entrar.

Coneixedors ja del funcionament d'aquest mòdul anem doncs a veure els diagrames de classe en UML de les classes que el formen.

5.6.1.1 Diagrama de la classe *ants.move.server.MoveProxy*

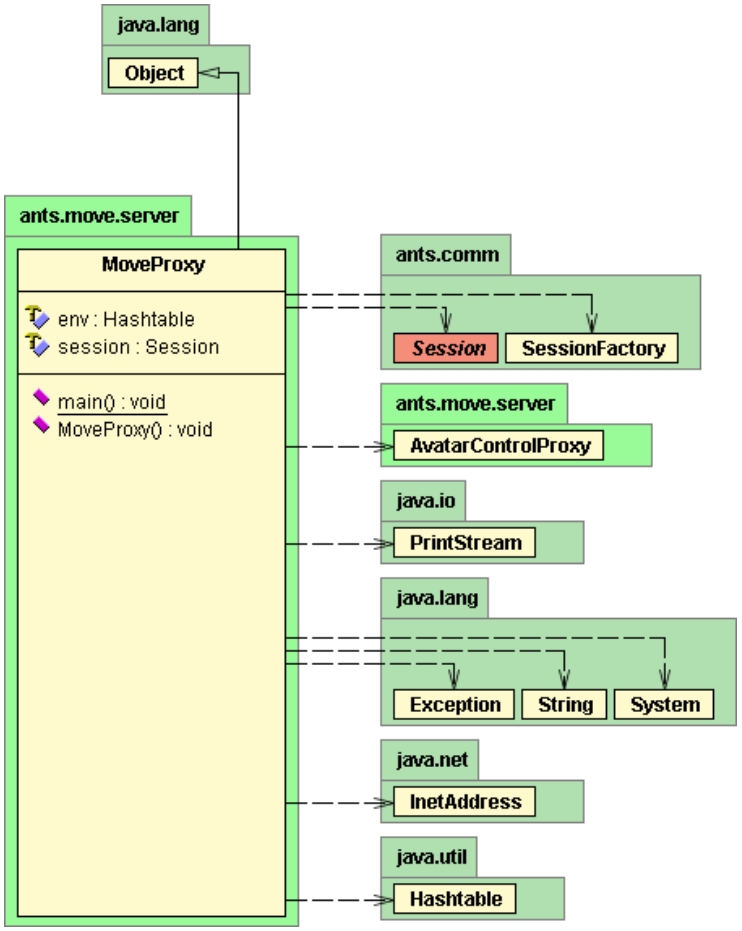


Figura 82. Diagrama de classes de la classe *ants.move.server.MoveProxy*.

5.6.1.2 Diagrama de la classe *ants.move.server.AvatarControlProxy*

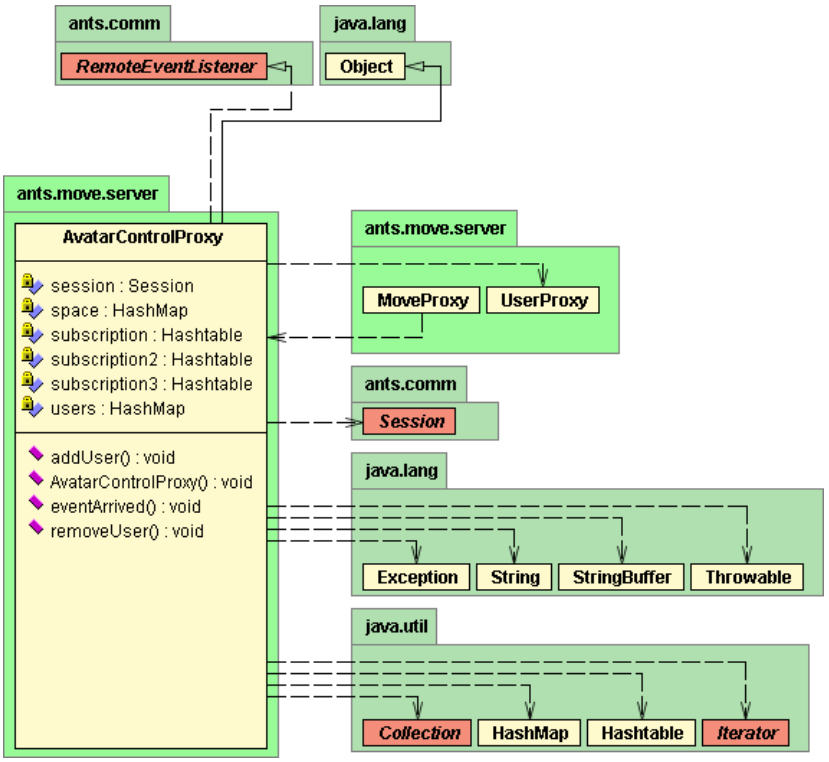


Figura 83. Diagrama de classes de la classe *ants.move.server.AvatarControlProxy*.

5.6.1.3 Diagrama de la classe *ants.move.server.UserProxy*

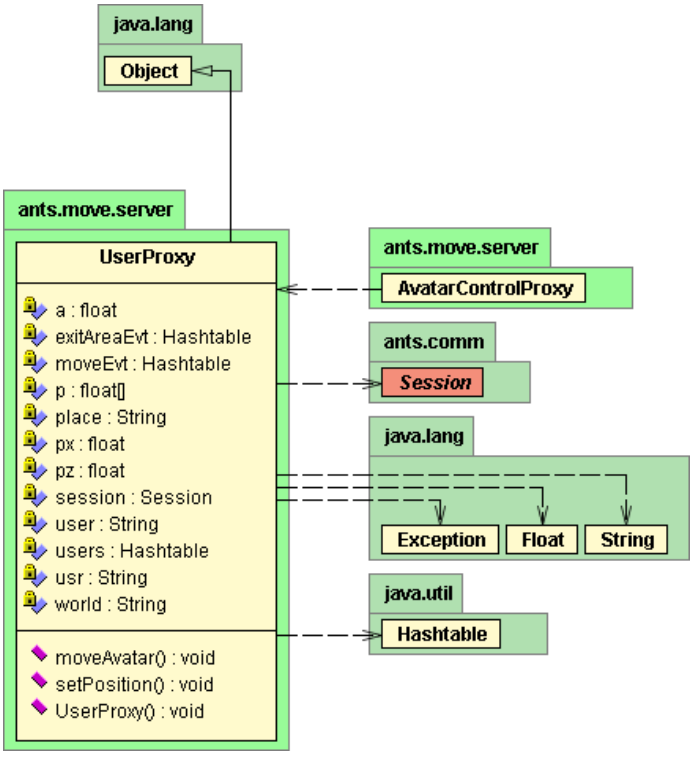


Figura 84. Diagrama de classes de la classe *ants.move.server.UserProxy*.

5.7 Les diferents *tools* del sistema

El nostre entorn intentarà simular la vida real, amb el major grau de detall possible, de forma que els usuaris es vegin immersos en una experiència tridimensional envolvent que potencii la comunicació entre aquests, els professors i entre ells mateixos.

Però el sistema no només consistirà en un entorn multiusuari col.laboratiu en temps real, sinó que agafant això com a base, es desenvoluparan una sèrie d'eines o *tools* que explotaran en la mesura del que sigui possible, les característiques de les tres dimensions i dels entorns en els que l'estat de les diferents entitats es troba sincronitzat a través de la xarxa. Això es portarà a terme sempre tenint com a objectiu el destí final del sistema, que ha de facilitar el procés d'aprenentatge per als estudiants i l'ensenyament als professors.

Respecte a aquestes *tools* del sistema, cal afegir que se'n proporcionen unes quantes per defecte. Això no vol dir pas que no se'n puguin afegir més. Una característica de l'aplicació és que disposa d'una **arquitectura extensible**, que fa que si un programador/a vol afegir alguna *tools* més, l'únic que ha de fer és crear-la a banda i compilar-la. Un cop fet això, sense necessitat de recompilar la nostra aplicació, aquesta *tool* pot afegir-se a l'escena sense cap més complicació, ja que la instanciació de les classes de les *tools* (models) es realitza en temps d'execució.

A continuació fem un petit comentari sobre les *tools* que proporcionem:

- **Pantalla de transparències (*Slides*):** Aquest objecte dinàmic serveix per a presentar transparències en 2D. És una aplicació col.laborativa molt valuosa que representa una contribució rellevant a l'aprenentatge a distància utilitzant la xarxa.

- **Simulacions 3D:** Al trobar-nos immersos en un espai tridimensional, és possible crear una presentació en la que les diapositives que la formen siguin elements tridimensionals, a l'estil dels hologrames. Fent servir el mateix mecanisme que en la *tool* anterior, és possible utilitzar objectes tridimensionals com a transparències i passar a la següent o tornar enrera. Mentre que en dues dimensions només es poden mostrar imatges planes, amb les tres dimensions es guanya la profunditat, amb el que les transparències passen a tenir volum. Això implica la possibilitat de poder contemplar l'objecte des de tots els angles possibles apreciant-lo en la seva totalitat. Amb aquesta eina també es poden presentar simulacions basades en un model matemàtic subjacent demostrat. Els sistemes basats en ordinador per al tele-ensenyament (*Computer Based Training Systems (CBT)*) han estat usats durant bastants anys amb un èxit relatiu. Amb els nous avenços en les telecomunicacions i gràfics 3D, els sistemes *CBT* estan guanyant terreny i el mercat de simuladors d'aprenentatge ha experimentat un creixement exponencial que era difícil de predir fa uns anys. Les simulacions educacionals han demostrat ser molt útils en cursos *online* existents en la web. Les simulacions 3D donaran una impressió més realista que els tradicionals applets de simulació plans.

- **Documents:** Amb la *tool* de documents podrem pujar qualsevol tipus de fitxer a MOVE i deixar-lo dins de l'entorn 3D del *place*, al lloc que desitgem. Podrem

agafar un document, portar-lo a un altre lloc del *place*, moure'l i veure'n el seu contingut.

- **URL:** La *tool* URL ens permetrà crear URLs i deixar-les dins de l'entorn 3D del *place*, al lloc que desitgem. Podrem agafar una URL, portar-la a un altre lloc, moure-la i visitar-la.

- **Càmera:** Aquesta *tool* ens pot ser molt útil en les situacions en què volem controlar un punt de vista determinat. Pot ser molt interessant col.locar la càmera al davant de la pantalla de transparències o bé davant de les barres 3D per poder controlar quin és el seu estat actual. En aquest cas la càmera esdevé una eina molt útil per poder veure una determinada posició sense moure'ns del lloc on estem.

- **Video:** La utilitat d'aquesta *tool* ve clarament definida. Es pot utilitzar de la mateixa manera que el seu homònim a la vida real. Moltes de les presentacions o classes que fa un professor venen complementades mitjançant la projecció d'un video explicatiu o divulgatiu que permet aclarir o extendre alguns punts de l'explicació. D'aquesta manera, la nostra aula virtual també està dotada d'aquest mitjà que permet que tots els alumnes i professors puguin disfrutar de la projecció d'un video adequat a la temàtica de la classe.

- **Audio:** Aquesta *tool* es pot utilitzar per emmagatzemar classes en audio i després reproduir-les en un *place* les vegades que es vulgui. Permet pujar pistes d'audio i se'n reprodeix el so de manera espaial, és a dir, que depenent de la posició i orientació en la que ens trobem respecte al model d'*audio*, sentirem el so amb més o menys intensitat.

- **Banner:** La *tool* Banner permet crear anuncis publicitaris (o no) que poden ser carregats i visualitzats dins l'entorn 3D del *place*. Un banner, doncs, consistirà en una seqüència de fotogrames que aniran apareixent de forma automàtica en la vista del model, un a un.

- **Hook:** La *tool* Hook permet inhabilitar el moviment de tots els altres usuaris del mateix *place* i forçar-los a que segueixin els nostres moviments per l'entorn tridimensional, des del nostre punt de vista. Molt útil per tal de realitzar visites guiades, impossibilitant als altres usuaris que vagin per lliure.

- **VideoStream:** Permet visualitzar *streams* de video dins l'entorn 3D del *place*.

- **Votació:** Amb aquesta eina podrem crear temes als quals tot usuari del *place* pugui donar la seva opinió en forma de vot. Cal tenir en compte que, com en el cas d'una votació real, només ens serà permès votar un sol cop.

- **NetMeeting:** Aquesta *tool* utilitza les propietats dels components *ActiveX* que ofereix *Microsoft Windows NetMeeting*, per tal de permetre videoconferència entre usuaris d'un mateix *place* de MOVE.

Només comentarem detalladament dos tipus de *tools*, ja que són els únics dos tipus diferents existents en quant a model de programació seguit. Comentarem, doncs, la *tool* de transparències 2D i la de documents.

5.7.1 La *tool* de presentacions 2D (Slides)

Totes les *tools* disposen d'una classe **Model**, una classe de **Vista** i una classe de **Control**. En el cas de la *tool* de presentacions 2D, tenim que aquestes classes són, respectivament, *ants.move.core.models.slides.SlidesModel*, *ants.move.core.views.slides.SlidesView* i *ants.move.core.teacher.slides.SlidesWindow*.

Començarem comentant la classe del model, **SlidesModel**, que com tots els models, hereda de la classe *ants.core.Thing*. Tal i com s'ha vist anteriorment, quan es carrega un model, el primer que se li executa és el mètode *init()*, que se subscriu al tipus d'esdeveniments que ha d'escoltar, així com inicialitza l'estat inicial del model (en aquest cas, obté la transparència que hi havia l'últim cop en la pantalla i la carrega). La classe del model proporciona els mètodes necessaris per avançar, retrocedir, anar a la primera, a la última transparència i carregar nous jocs de transparències. També s'encarrega d'escoltar sobre els possibles canvis en les seves propietats provocats per esdeveniments llençats per altres models de la mateixa *thing*. Així doncs, implementa el mètode *propertyChange()* de la classe *ants.core.PropertyChangeListener*, de manera que quan rebi un esdeveniment de canvi de propietat al que està subscrit, s'executarà aquest mètode i s'actuarà en conseqüència. Passem a veure el diagrama de classes de la classe *ants.move.core.models.slides.SlidesModel*.

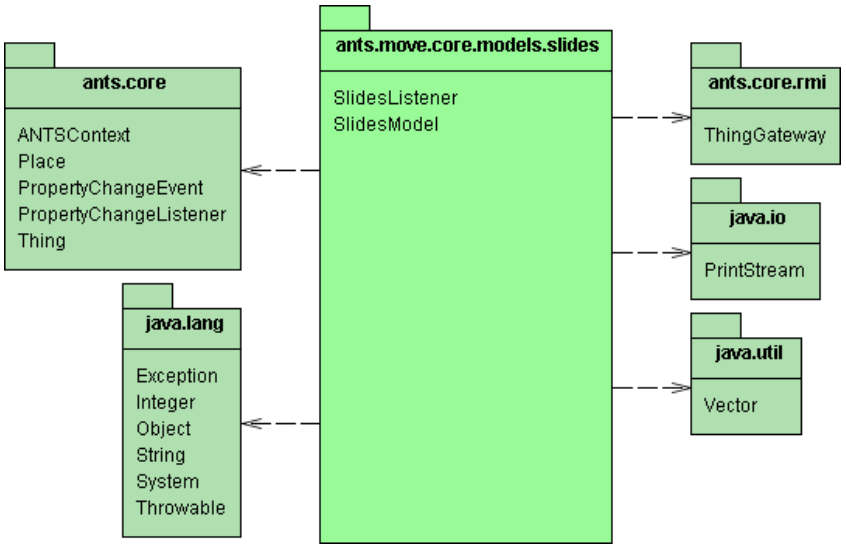


Figura 85. Diagrama de classes del package *ants.move.core.models.slides*.

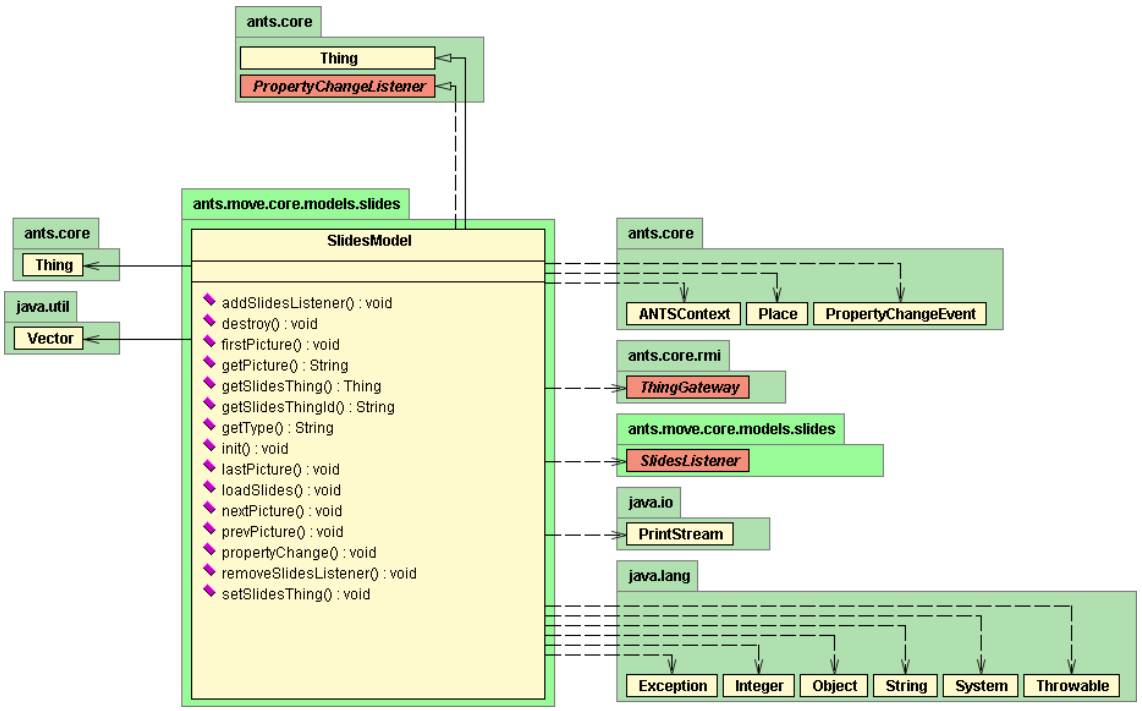


Figura 86. Diagrama de classes de la classe *ants.move.core.models.slides.SlidesModel*.

La classe **Model** proporciona una classe d'interfície (*ants.move.core.models.slides.SlidesListener*), que tant la classe **Vista** com la classe **Model** implementaran si volen escoltar esdeveniments que afectin a la seva *tool* i model.

Així doncs, la classe **Model** és la que conté i manté les dades de l'objecte. Per contra, la classe de **Vista** només s'encarrega de visualitzar aquestes dades que manté la classe **Model**. La classe de **Vista** és *ants.move.core.views.slides.SlidesWindow*.

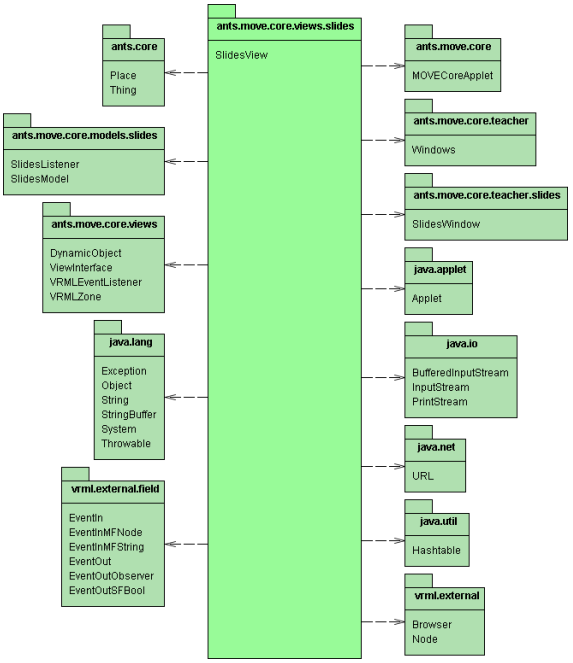


Figura 87. Diagrama de classes del package *ants.move.core.views.slides*.

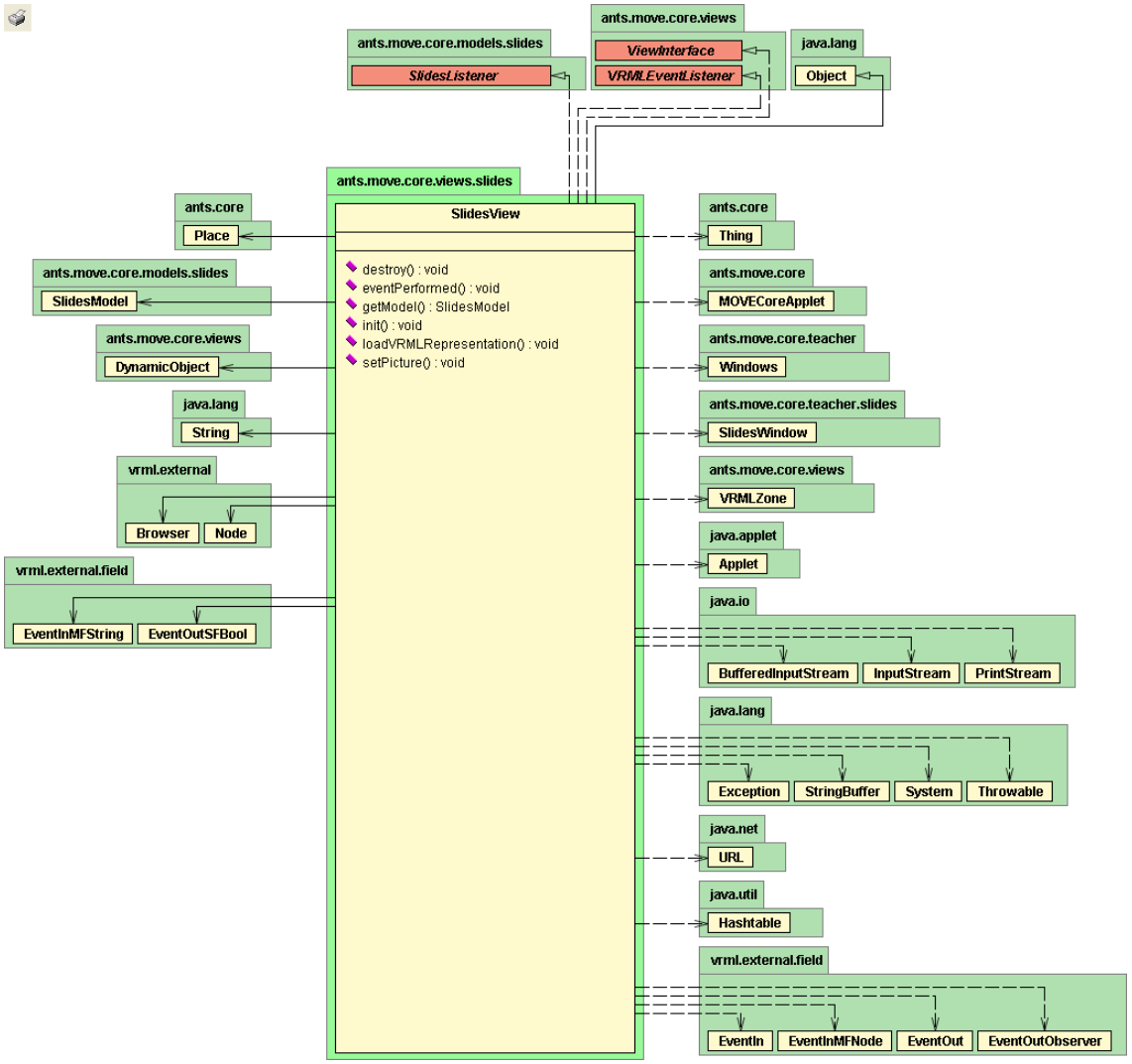


Figura 88. Diagrama de classes de la classe *ants.move.core.views.slides.SlidesView*.

Aquesta classe, doncs, serà l'encarregada de mostrar la representació tridimensional del model *Slides*. Primerament, amb l'execució del seu mètode *init()*, obté una referència al model, carrega la seva representació *VRML*, és a dir, que carrega quin serà l'aspecte de l'objecte que contindrà les transparències (aquest objecte es trobarà en un fitxer **.wrl* i dins del seu contingut haurà de tenir definit un node amb el nom "TEX", de manera que s'enruta aquest node cap al node que capturem). Finalment, afegirà un *listener* per a poder escoltar els esdeveniments que ocorrin en aquell model, de manera que quan canviï el valor del fitxer a mostrar, canviï també la imatge a la pantalla.

Finalment, ens queda el component del **Control**, que és l'encarregat de modificar l'estat del model.

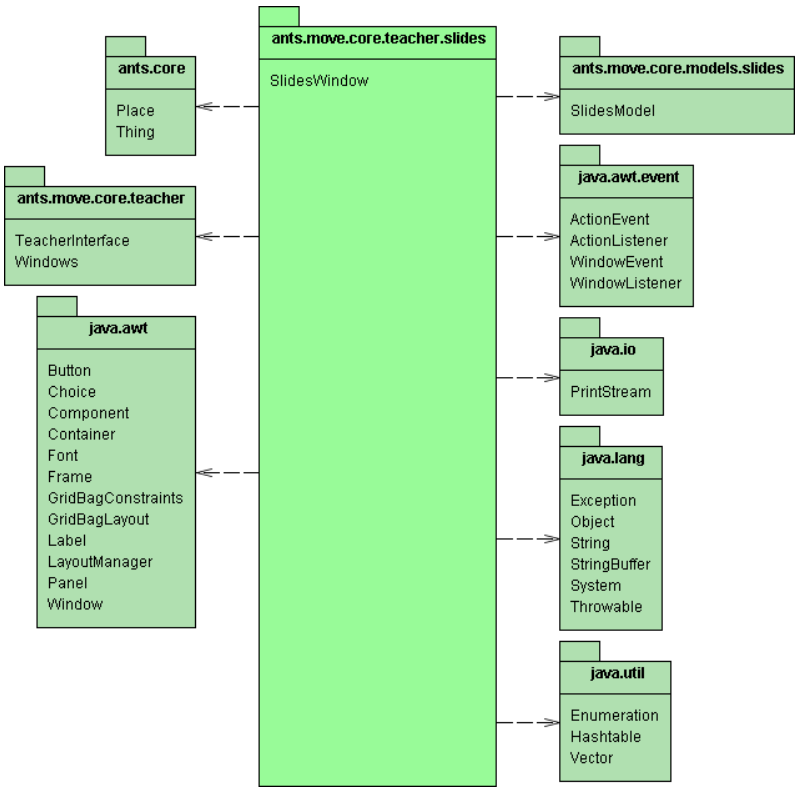


Figura 89. Diagrama de classes del package *ants.move.core.teacher.slides*.

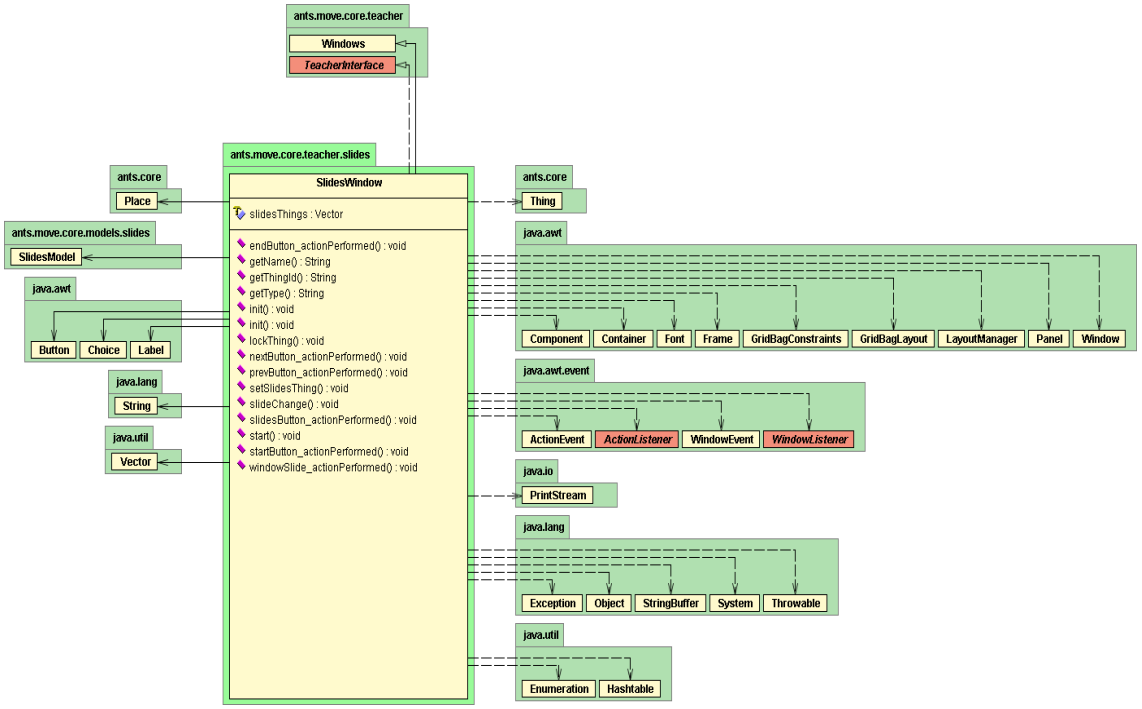


Figura 90. Diagrama de classes de la classe *ants.move.core.teacher.slides.SlidesWindow*.

Tal i com es pot veure, en realitat aquesta classe només presenta una finestreta a l'usuari per tal que pugui avançar, retrocedir, canviar de joc de transparències, etc. En el seu mètode *init()*, obté una referència al model i totes les *things* d'aquell *place* que són jocs de transparències, per tal que l'usuari pugui triar quin conjunt de transparències vol

utilitzar. Cal remarcar que quan l'usuari vol, per exemple, avançar una transparència, el que es fa realment és, des de la classe **Control**, cridar el mètode *nextPicture()* de la classe **Model**. La classe **Model** canviarà el seu estat intern canviant el valor de la propietat de la transparència actual (i actualitzant també la base de dades) i a més, enviarà un esdeveniment, que capturarà la classe **Vista** de manera que aquesta canviarà la representació en pantalla i mostrarà la nova transparència.

5.7.2 La *tool* de documents

Anem a descriure la *tool* de documents degut a que juntament amb la *tool* URL es diferencia substancialment de les altres *tools*. La *tool* **Doc** la utilitzarem per fer *uploads* de fitxers en el *place* de manera que aquests quedaran disponibles per als altres usuaris. Com la resta de tools, utilitza el patro MVC. La gran diferència amb les altres *tools* radica en que no diposa d'un objecte VRML únic per la *tool* on es realitza la gestió de la mateixa sinó que per a cada fitxer que pugem al servidor apareixerà en la pantalla un cub, amb l'icona corresponent al tipus d'arxiu, en la posició que es troba l'usuari en el moment de fer l'*upload*.

L'usuari podrà realitzar l'*upload* de l'arxiu clickant sobre l'icona corresponent en la interfície gràfica. Això farà que s'executi un **JSP** que realitzarà l'*upload* de l'arxiu al servidor i enviarà un esdeveniment al client de manera que aquest col.locarà l'objecte VRML a l'entorn tridimensional. Tot seguit enviarà un altre esdeveniment a tots els altres clients comunicant que un nou objecte de tipus **Doc** s'ha afegit al *place*.

Si ppremem sobre ell disposarem de la possibilitat de moure l'objecte VRML, així com agafar-lo i deixar-lo en un altre lloc. També podrem veure'n la informació referent a la data de pujada, descripció, propietari i nom, a part de la possibilitat d'eliminar-lo.

La representació en l'entorn 3D de la *tool* **Doc** és el següent objecte:

Figura 91. Icona de representació de la *tool* **Doc**.

La imatge que apareix en el cub, dependrà del tipus de document de que es tracti. MOVE reconeixerà alguns dels tipus de documents més freqüents:

Adobe Portable Document Format (PDF) 

Microsoft Word 

Microsoft Access 

Microsoft Excel 

Microsoft Powerpoint 

Fitxer ZIP

Document HTML

Fitxer POSTSCRIPT

Fitxer Multimedia

Fitxer Genèric

Tal i com hem esmentat, s'utilitza el patró MVC. És per això que els diferents packages que composen la tool són *ants.move.core.models.doc*, *ants.move.core.teacher.doc* i *ants.move.core.views.doc*.

Igual que en totes les *tools*, el model es carregarà al connectar-se al *place* gràcies al mètode *loadModels()* que proporciona **ants-core. DocModel** carregarà tants models del tipus **DocThing** com objectes de tipus **Doc** existeixin en el *place*. Tot seguit, **move-core** carregarà la vista. En aquest punt disposarem d'una classe **DocView** que serà l'encarregada de crear tantes instàncies de **DocThingView** com objectes de tipus Doc existeixin en el *place*. El control corresponent només es carregarà al prémer sobre un objecte de tipus **Doc** en l'entorn VRML, sempre que tinguem permís per a interactuar-hi. En cas afirmatiu es carregarà el control (**DocControl**) corresponent a l'objecte de tipus **Doc** per tal de poder-hi interactuar.

Anem a veure els diagrames de classes dels packages que componen la tool **Doc**.

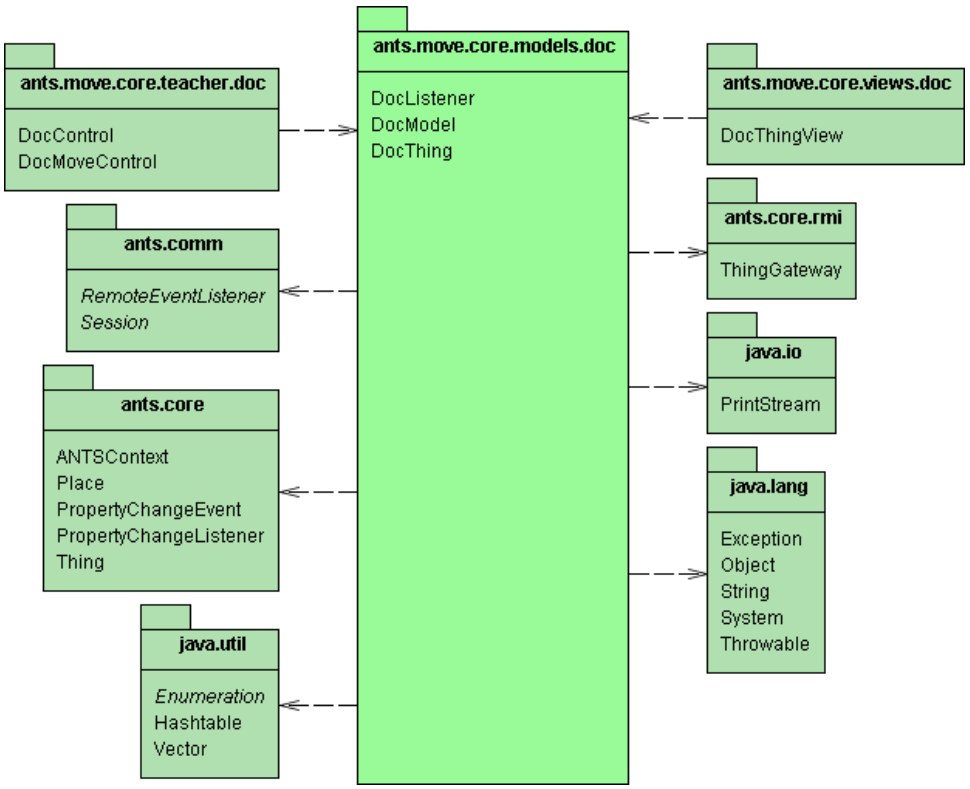


Figura 92. Diagrama de classes del package *ants.move.core.models.doc*.

La interfície que han de seguir les vistes per tal que els models pugui fer-hi efectius els canvis que esdevenen és la seguent:

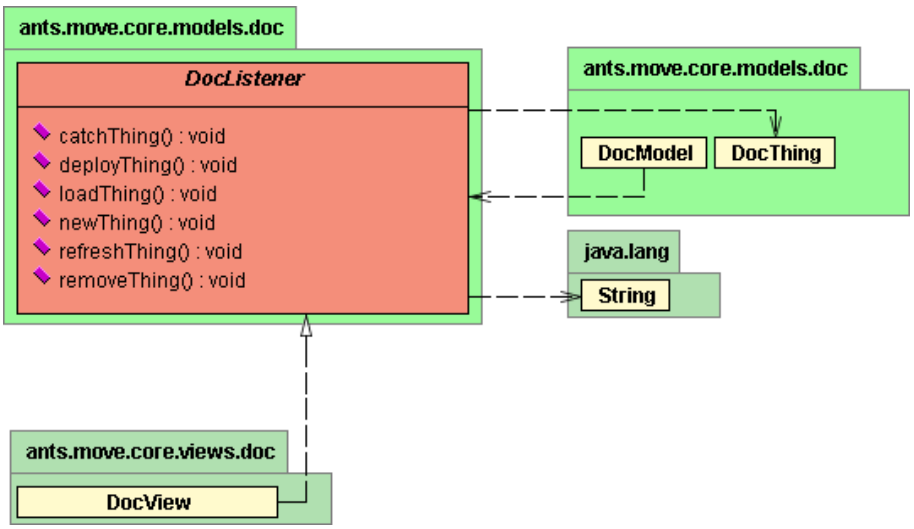


Figura 93. Diagrama de classes de la classe *ants.move.core.models.doc.DocListener*.

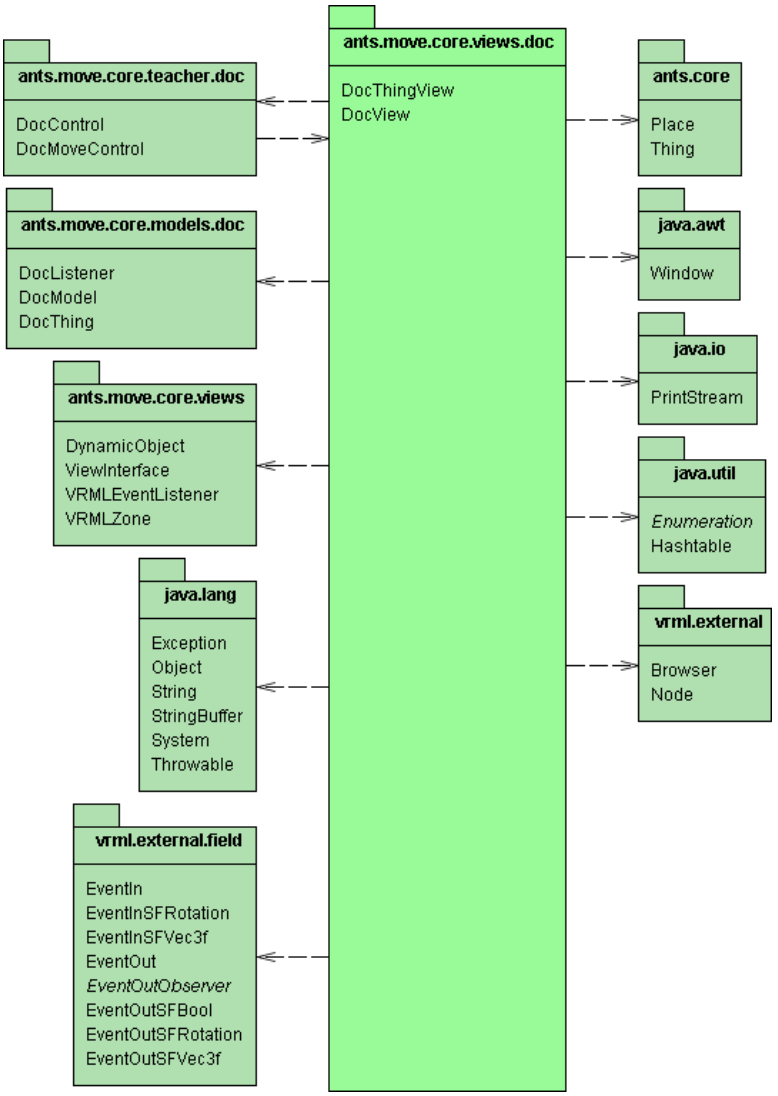


Figura 94. Diagrama de classes del package *ants.move.core.views.doc*.

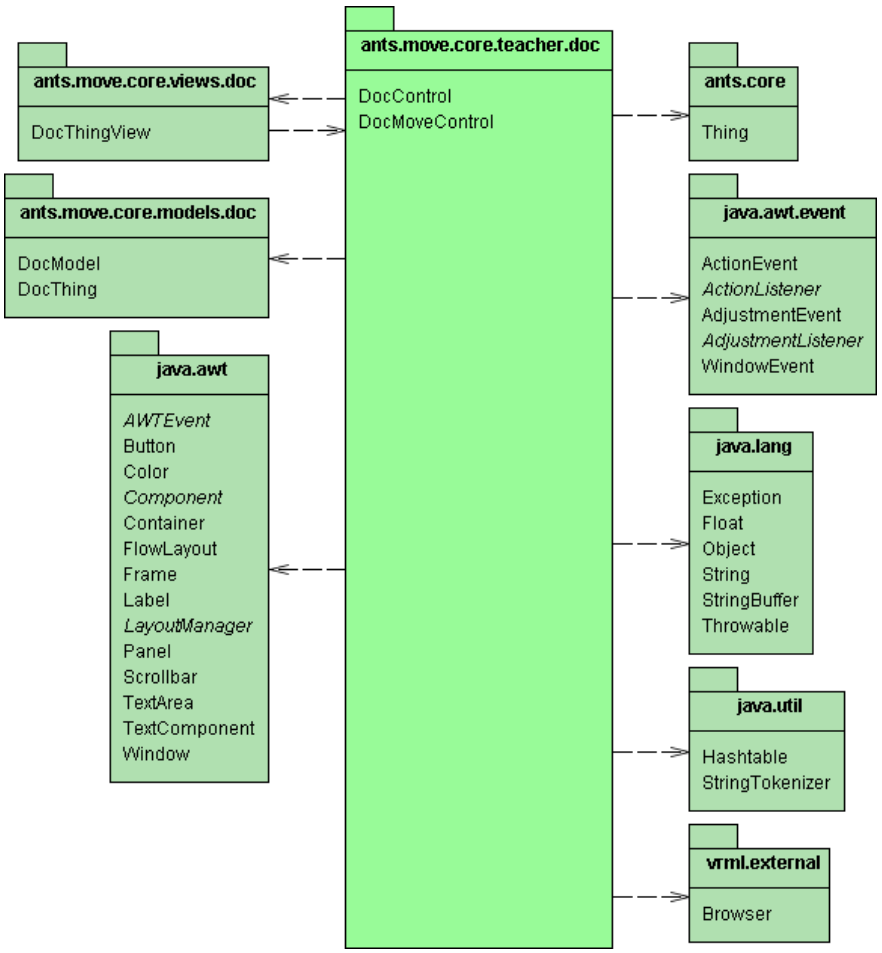


Figura 95. Diagrama de classes del package *ants.move.core.teacher.doc.*

5.8 El sistema de *bots*

Gaudir de la possibilitat de crear nous *bots* és d'una gran importància ja que augmenta les possibilitats del sistema. D'aquesta manera, podem crear *bots* que reaccionin els events que es produeixin en el *place* i fins i tot podem crear *bots* que es relacionin entre ells per tal d'assolir un determinat objectiu.

Un client determinat veu un *bot* com si es tractés d'un altre client de MOVE. Per el sistema, la utilització de *bots* és totalment transparent ja que es tracten com si fossin usuaris. Llencen els mateixos tipus d'esdeveniments tant de moviment com d'entrada i sortida. A més, un *bot* pot carregar els diferents models de les *tools* que estan actives en un *place* i utilitzar-les com si fos un usuari. Així, els altres usuaris veuran els canvis que el *bot* produeix en els models del *place*. D'aquesta manera, podem executar un *bot* en qualsevol maquina i el sistema el tractarà igual que si fos un client. El *bot* igual que un usuari normal es connecta al *place* utilitzant les funcions del package **ants-core** i carrega els models de les *tools*. A partir d'aquest punt ja pot moure's i utilitzar les *tools* com si fos un usuari normal del *place*.

Es disposa de la possibilitat de crear *bots* de diferents tipus mitjançant l'herència de la classe *ants.move.bot.Bot*, calent només implementar-ne el mèdode *ai()*.

A tall d'exemple hem realitzat la implementació d'un *bot* guia, el qual fa una petita explicació dels components i característiques d'un *place*.

El diagrama de classes en UML de la classe *ants.move.bot.Bot* és el mostrat a la pàgina següent.

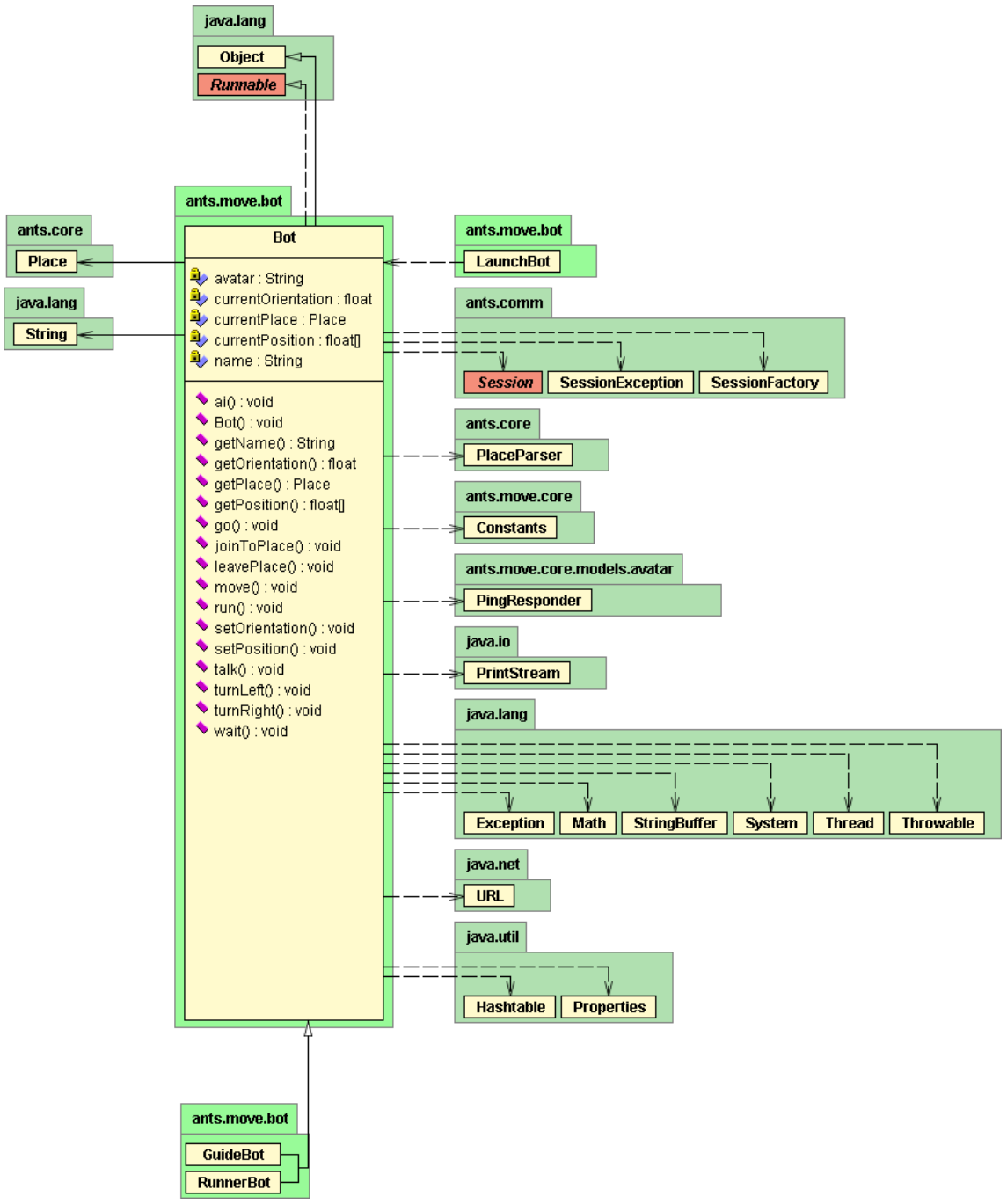


Figura 96. Diagrama de classes de la classe *ants.move.bot.Bot.*

Per tal d'executar un *bot* cal seguir els següents passos:

- Implementar un nou *bot* mitjançant l'herència de la classe *ants.move.bot.Bot* i implementar el seu mètode *ai()* amb les accions a realitzar.

  Exemple:

```
package ants.move.bot;

public class RunnerBot extends Bot {
   public RunnerBot (String name, String avatar) {
      super (name, avatar);
   }

   public void ai() {
      // Accions a realitzar
   }
}
```

- El fitxer font ha d'estar al directori '**src/ants/move/bot**'.

- Compilar-lo mitjançant l'eina *Jakarta ANT*:

```
$ <directori_move>/demo/move-app/move-web/ant compile-moveBot
```

- Modificar el fitxer '**bot.properties**' del directori '**<directori_move>/demo/move-app/move-web/**'

```
host=http://<nom_màquina>/move/
world=<id del world on es connectara el bot>
place=<id del place on es connectara el bot>
name=<nom del bot>
avatarUrl=http://<nom_màquina>/move/resources/avatars/<fitxer .wrz>
botClass=<nom del fitxer .class del bot>
```

- Executar el script '**bot.bat**' del directori '**<directori_move>/demo/move-app/move-web/**'

7. Manual de l'usuari

7.1 Introducció

7.1.1 Què és Move

MOVE és una aplicació web que permet la interacció de diferents usuaris en un entorn tridimensional.

MOVE ofereix diferents eines (*tools*) per tal de facilitar la col.laboració i interacció entre els usuaris, com per exemple:

- Pantalla de transparències virtual
- Simulacions 3D
- Reproductor de videos
- Reproductor de pistes d'audio
- etc . . .

L'arquitectura de MOVE és totalment extensible i configurable, ja que permet afegir noves tools i nous entorns 3D.

7.1.2 Pre-requisits per usar MOVE

Per poder utilitzar MOVE cal:

- Un **navegador web** per tal d'accedir a l'interficie web de l'aplicació (Netscape Navigator o MS Internet Explorer).
- Un **plug-in VRML** instal.lat en el navegador per tal de visualitzar els entorns 3D (Blaxxun Contact o Cosmo Player).
- En el supòsit de que la versió del Internet Explorer sigui la 4.0 o superior, també caldrà instal.lar el suport RMI per la màquina virtual java de Microsoft. (HOB RMI).

7.1.3 Informació addicional

Podeu trobar més informació sobre MOVE a la seva pàgina oficial: http://ants.etse.urv.es/move

7.2 Utilització de MOVE

7.2.1 Autentificació

El primer que ens trobem, abans d'entrar a MOVE, és la pantalla d'**autentificació**. Només podrem seguir endavant si hem estat anteriorment registrats com a usuaris per un administrador.

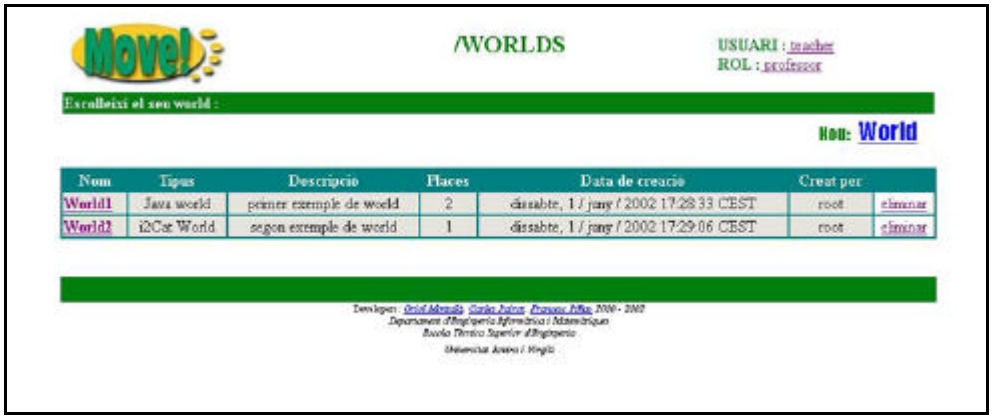
Haurem d'omplir les dues caselles amb el nostre *login* i *password*, respectivament. Un cop introduides les dades cal prémer el botó i si les dades són correctes entrarem al MOVE. En cas contrari, apareixerà una finestra d'alerta, donant-nos la possibilitat d'intentar-ho novament.

A part d'això, és aquí també on podrem elegir l'idioma amb el que volem que se'ns presenti la informació:

- Català
- Castellà
- Anglès

7.2.2 Worlds

Una vegada hem estat autentificats dins de MOVE, el següent que ens trobarem serà la zona de **WORLDS**.



Aquí podrem veure una taula amb tots els Worlds que hi ha creats actualment. La informació que podrem veure d'ells és la següent:

- **Nom:** nom del World
- **Tipus:** tipus de World
- **Descripció:** descripció del World
- **Places:** nº de places que conté aquest World
- **Data de creació:** data en què ha estat creat
- **Creat per:** login de l'usuari que l'ha creat

Podrem entrar en qualsevol d'ells només clicant sobre el seu nom, o bé eliminar-lo clicant sobre el text **eliminar**.

Per altra banda també podrem crear nous Worlds. Com podem veure en l'anterior imatge, en la part superior dreta de la taula apareix World. Doncs bé, si cliquem sobre la paraula World anirem a la següent pantalla:

Per crear un nou World només caldrà omplir els següents camps,

- **Nom:** nom del World

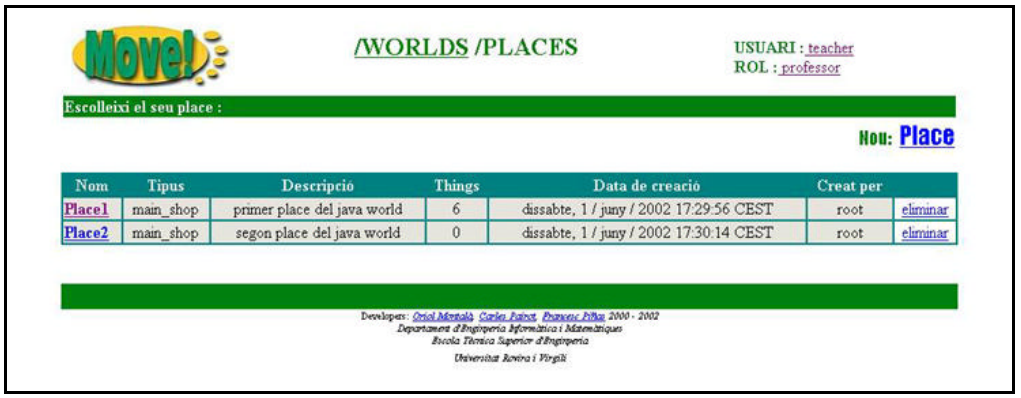
- **Descripció:** descripció del World
- **Tipus:** tipus de World

i seguidament prémer el botó crear. Automàticament tornarem a la pantalla anterior i veurem com el nostre nou World ja apareix a la taula de Worlds disponibles.

Només ens restarà clicar sobre el seu nom i ja podrem entrar en el nostre World creat recentment.

7.2.3 Places

Després d'haver entrat en algun dels Worlds disponibles ens trobarem a la zona **PLACES** del World seleccionat amb anterioritat.

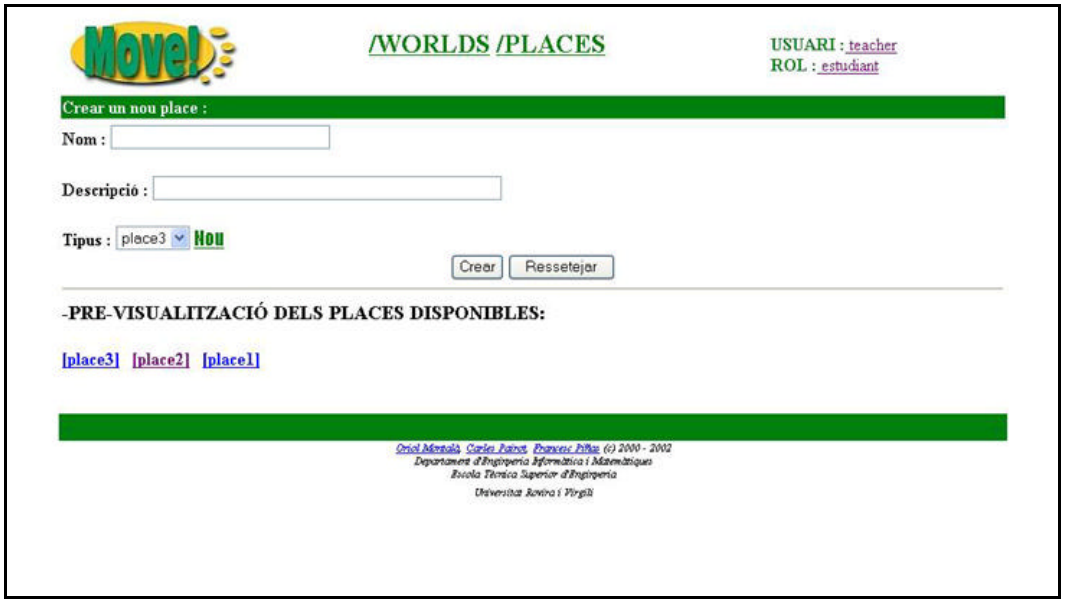


Aqui podem veure una taula amb tots els Places que hi ha creats actualment dins del World on estem. La informació que podem veure d'ells és la següent:

- **Nom:** nom del Place
- **Tipus:** tipus de Place
- **Descripció:** descripció del Place
- **Things:** nº de things que conté aquest Place
- **Data de creació:** data en què ha estat creat
- **Creat per:** login de l'usuari que l'ha creat

Podem entrar en qualsevol d'ells només clicant sobre el seu nom, o bé eliminar-lo clicant sobre eliminar .

Per altra banda també podem crear nous Places. Com podem veure en l'anterior imatge, a la part superior dreta de la taula apareix Place. Doncs bé, si cliquem sobre aquesta opció, anirem a la següent pantalla:



Per crear un nou Place només caldrà omplir els següents camps,

- **Nom**
- **Descripció**
- **Tipus**

Abans de decidir el *tipus de place*, podem visualitzar-lo clicant sobre el seu nom en l'apartat de *"PRE-VISUALITZACIÓ DE PLACES DISPONIBLES"*.

Una vegada introduides totes les dades haurem de prémer el botó 

. Automàticament tornarem a la pantalla anterior i veurem com el nostre nou Place ja apareix a la taula de Places disponibles.

7.2.4 Things

Després d'entrar a un Place ens trobarem a la zona **THINGS** del Place i World seleccionats amb anterioritat.

Aqui trobarem una taula amb totes les Things que hi ha en el Place on ens trobem. Cada Thing de la taula tindrà la següent informació:

- **Thing:** icona corresponent al tipus de Thing

  Els diferents tipus de icones amb què ens trobarem seran els següents:

  -  thing de tipus **Slides**
  -  thing de tipus **Simulation3D**
  -  thing de tipus **Audio**
  -  thing de tipus **Banner**
  -  thing de tipus **Document**
  -  thing de tipus **URL**
  -  thing de tipus **Votation**
  -  thing de tipus **VideoStream2**
  -  thing de tipus **Video**

- **Nom:** nom de la Thing
- **Descripció:** descripció de la Thing
- **Data de creació:** data en què ha estat creada
- **Creat per:** login de l'usuari que la creada

A més tindrem la oportunitat d'eliminar aquelles Things que ja no ens interessin, excepte les de tipus *document* i *URL*, les quals sols podran ser eliminades des del interior de l'entorn 3D del Place.

A part d'aquesta taula també podem observar que apareixen una sèrie d'icones a la part superior d'aquesta:

Es tracta de la barra de Tools actives en aquest Place i que, a més, permeten crear Things del seu tipus solament fora de l'entorn 3D:

- Audio
- Banner
- Simulation3D
- Slides
- Video
- VideoStream2

Clicant sobre la corresponent icona de la Tool anirem a la pantalla on podrem crear Things per aquesta Tool.

Pel que fa a la icona  , aquesta ens permetrà accedir a la pantalla de configuració de les Tools:

Aquí podrem activar o desactivar aquelles Tools a les quals tenim accés, per tal d'utilitzar-les o no dins el Place.

El cas especial serà la Tool Builder. Si el nostre rol pot utilitzar-la apareixerà a la part superior dreta de la taula la paraula **Builder**. En cas contrari, no.

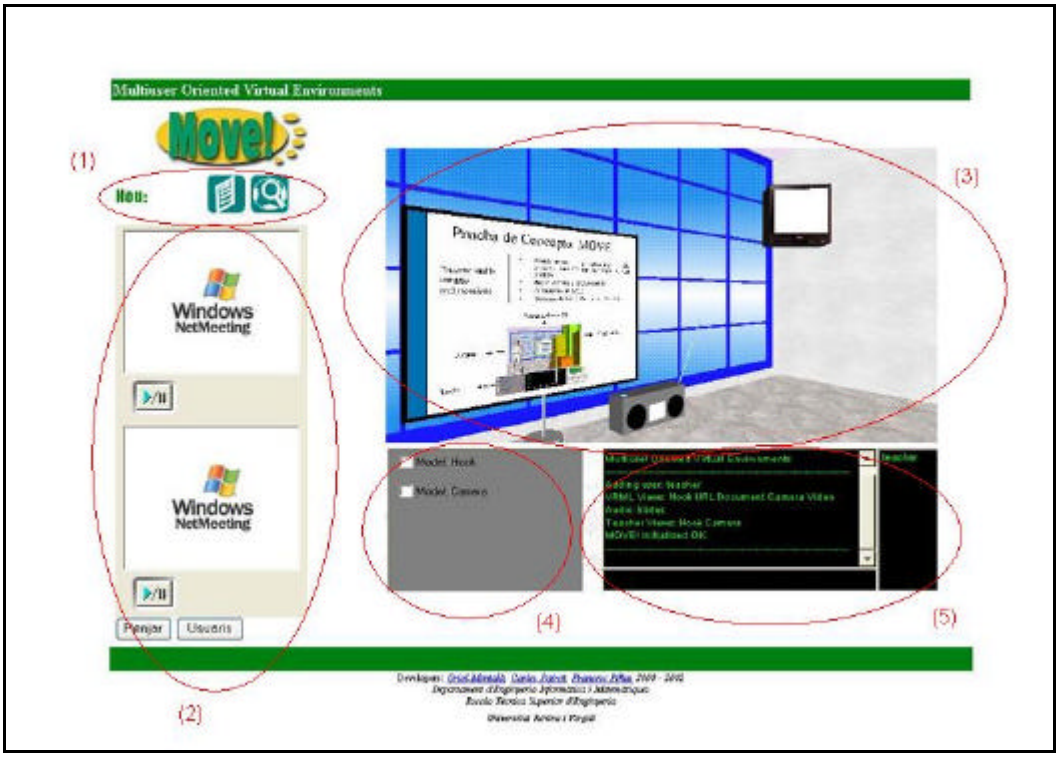
Si cliquem sobre ella accedirem a la pantalla del Builder.

Una vegada finalitzades les nostres accions en aquesta zona THINGS, ja només ens quedara clicar sobre **ENTRAR** i entrar definitivament a l'entorn 3D del Place.

7.2.5 Entorn 3D del Place

És aqui on podrem dir que realment estem dins el MOVE!.

La pantalla que tindrà l'usuari de MOVE! davant seu serà la següent:



Podem veure que consta de diferents parts:

1. **Barra de Tools:** Està composta de les Tools que permeten només la creació de Things del seu tipus dins l'entorn 3D, i que a més estan actives dins aquest Place:

   - Document
   - URL

   Podrem crear Things d'aquests tipus clicant sobre la icona de la Tool que representen.

2. **Tool Netmeeting:** Sols apareixerà si ha estat activada per a ser utilitzada dins aquest Place

3. **Entorn 3D del Place:** Podrem moure'ns per ell i interectuar amb tots els models que hi hagi carregats de les Tools i sobre els quals tinguem permís.

4. **Models sense representació 3D en el Place:** Activant la casella corresponent accedirem al control del model.

5. **Finestra de xat:** Ens permetrà comunicar-nos amb tots els usuaris que estan en el Place. A més, podrem enviar missatges privats a qualsevol d'ells introduint la següent comanda:

/priv *login_usuari missatge*

Per últim, si cliquem sobre el logo del MOVE tornarem a la zona THINGS del Place.

7.3 Descripció de les tools

7.3.1 Audio

La Tool **audio** ens permetrà pujar pistes d'audio al MOVE i escoltar-les després dins l'entorn 3D del Place.

La principal característica d'aquesta Tool és que reprodueix el so en 3 dimensions, és a dir, depenent de la nostra posició i orientació respecte el model audio, sentirem la pista d'audio amb més o menys intensitat.

7.3.1.1 Pujar pista d'audio

Per tal de crear una Thing de tipus audio el que haurem de fer és clicar sobre l'icona  que apareix en la barra de tools actives i seguidament ens apareixerà la següent pantalla:

Per a poder crear la Thing hem d'omplir els següents camps:

- **Nom de la pista d'audio:** nom de la pista
- **Descripció de la pista d'audio:** descripció de la pista
- **Escolleixi una pista d'audio:** fitxer que conté la pista d'audio. (tipus suportats: **AIFF**, **WAV**, **MPEG 1** i **2**)

Un cop introduides les dades en el petit formulari ens cal prémer el botó 

per tal que la nova Thing sigui creada i pujada al servidor del MOVE. Si tot ha anat bé serem redirigits a la pantalla on es mostra la llista de Things que hi ha en el Place, i observarem que la Thing que hem creat ja apareix entre elles.

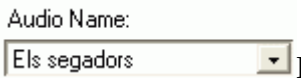
7.3.1.2 Escoltar pista d'audio

Per escoltar la Thing de tipus audio sense necessitat d'entrar a l'entorn 3D de MOVE només cal clicar sobre la seva icona en la taula on apareixen totes les Things que hi ha en el Place.

7.3.1.3 Funcionament del model audio

El model audio apareix en l'entorn 3D representat pel següent objecte:

Per tal d'accedir al control del model audio sols haurem de clicar sobre ell i ens apareixerà la següent finestra:

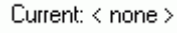


Ens mostra una llista de totes les Things de tipus audio que hi ha creades actualment en el Place

Carrega la pista d'audio que hi ha seleccionada en el camp *AudioName*

Comença a reproduir la pista d'audio

Finalitza la reproducció de la pista d'audio



Indica el nom de la pista d'audio que hi ha actualment carregada

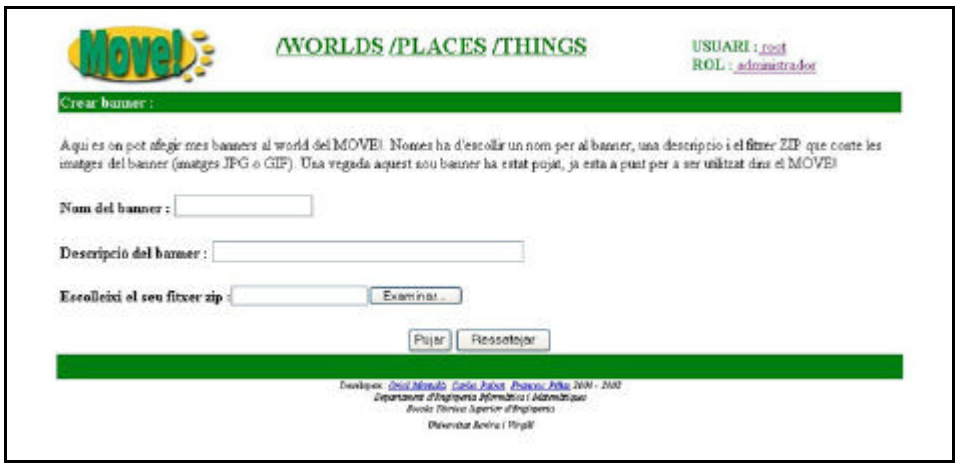
7.3.2 Banner

La Tool **banner** permetrà crear anuncis publicitaris (banners), els quals podran ser carregats i visualitzats en el model banner, dins l'entorn 3D del Place.

Un *banner* consistirà en una seqüència de fotogrames que aniran apareixent de forma automàtica en el model, un a un.

7.3.2.1 Pujar banner

Per tal de crear una Thing de tipus banner el que haurem de fer és clicar sobre l'icona  que apareix en la barra de tools actives i seguidament ens apareixerà la següent pantalla:



Per a poder crear la Thing hem d'omplir els següents camps:

- **Nom de banner:** nom del banner
- **Descripció del banner:** descripció del banner
- **Escolleixi el seu fitxer zip:** fitxer **ZIP** que contindrà tots els fotogrames del banner

Un cop introduides les dades en el petit formulari ens cal prémer el botó  per tal que la nova Thing sigui creada i pujada al servidor del MOVE!. Si tot ha anat bé serem redirigits a la pantalla on es mostra la llista de Things que hi ha en el Place, i observarem que la Thing que hem creat ja apareix entre elles.

7.3.2.2 Pre-visualització del banner

Per poder veure la Thing de tipus banner sense necessitat d'entrar a l'entorn 3D de MOVE només cal clicar sobre la seva icona en la taula on apareixen totes les Things que hi ha en el Place.

Un cop fet això veurem aquesta pantalla, on podrem veure el banner en funcionament

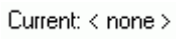
7.3.2.3 Funcionament del model banner

El model banner apareix en l'entorn 3D representat pel següent objecte:

Per tal d'accedir al control del model banner sols haurem de clicar sobre ell i ens apareixerà la següent finestra:

Ens mostra una llista de totes les Things de tipus banner que hi ha creades actualment en el Place.

Carrega el banner que hi ha seleccionat en el camp *banner* i el posa automàticament en funcionament.



Indica el nom del banner que hi ha actualment carregat

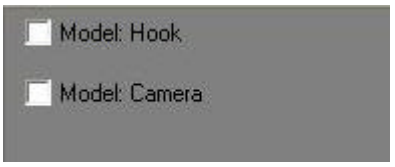
7.3.3 Camera

Amb la Tool **camera** podrem deixar una càmera fictícia en qualsevol lloc de l'entorn 3D del Place, i en un moment determinat, canviar el nostre punt de vista pel de la càmera.

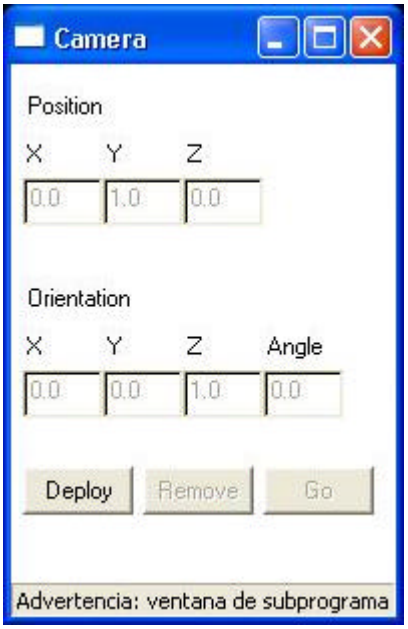
7.3.3.1 Funcionament del model camera

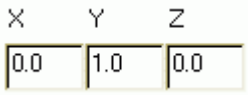
El model camera, com ja hem dit en la introducció del capítol, no té representació 3D en el Place.

Per a poder accedir al model haurem d'anar a la següent zona de la pantalla on apareix l'entorn 3D del Place,

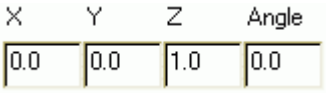


i activar la casella corresponent. Seguidament ens apareixerà la finestra de control del model:



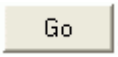


Ens diu la posició on es troba la càmera.



Ens diu quina n'és la seva orientació.

Situa la càmera en la posició on ens trobem actualment.



Canvia el nostre punt de visió pel de la càmera.

Retira la càmera del seu lloc.

7.3.4 Document

Amb la Tool **document** podrem pujar qualsevol tipus de fitxer a MOVE i deixar-lo dins de l'entorn 3D del Place, al lloc que desitgem. Podrem agafar un document, portar-lo a un altre lloc del Place, moure'l, i veure el seu contingut.

7.3.4.1 Pujar document

Per tal de crear una Thing de tipus document el que haurem de fer és clicar sobre l'icona  que podem trobar a la pantalla on apareix l'entorn 3D del Place, sota el logotip de MOVE!. Se'ns obrirà una finestra com aquesta:

Per a poder crear la Thing hem d'omplir els següents camps:

- **Nom del document:** nom del document
- **Descripció del document:** descripció del document
- **Escolleixi un document:** fitxer que conté el document

Un cop introduides les dades en el petit formulari ens cal prémer el botó 

per tal que la nova Thing sigui creada i pujada al servidor de MOVE. Si tot ha anat bé la finestra es tancarà i veurem com dins el Place on estem apareix la representació 3D del nou document.

7.3.4.2 Veure el document

Per poder veure la Thing de tipus document sense necessitat d'entrar a l'entorn 3D de MOVE només cal clicar sobre la seva icona en la taula on apareixen totes les Things que hi ha en el Place.

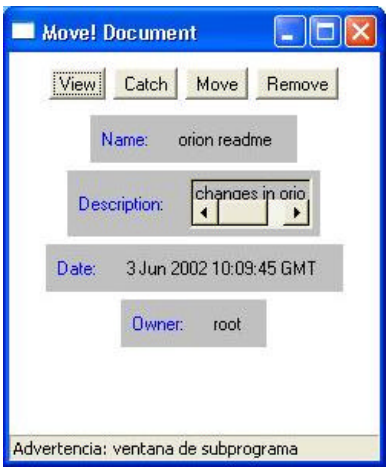
7.3.4.3 Funcionament del model document

El model document apareix en l'entorn 3D representat per el següent objecte:

La imatge que apareix en el cub, dependrà del tipus de document de que es tracti. MOVE! reconeixerà alguns dels tipus de documents més freqüents:

- fitxer **PDF**
- document de **WORD**
- document de **ACCESS**
- document de **EXCEL**
- presentació **POWERPOINT**
- fitxer **ZIP**
- fitxer **HTML**
- fitxer **POSTSCRIPT**
- fitxer **MULTIMEDIA**
- fitxer genèric

Per tal d'accedir al control del model document sols haurem de clicar sobre ell i ens apareixerà la següent finestra:



View Ens permet veure el contingut del document.

Catch Ens permet agafar el document i portar-lo a un altre lloc.

Move Ens permet canviar la posició actual del document.

Remove Elimina el document del MOVE.

Name: orion readme Nom que té el document.

Description: changes in orio Descripció del contingut del document.

Date: 3 Jun 2002 10:09:45 GMT Data en què ha estat pujat el document al MOVE.

Owner: root Propietari del document.

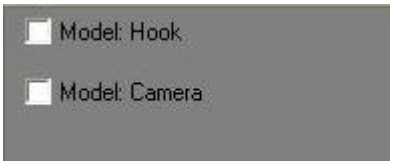
7.3.5 Hook

La Tool **hook** ens permet innhabilitar el moviment de tots els altres usuaris del mateix Place i forçar a que segueixin els nostres moviments per l'entorn 3D des del nostre punt de vista.

7.3.5.1 Funcionament del model hook

El model hook, com passa amb el model camera, tampoc té representació 3D en el Place.

Per a poder accedir al model haurem d'anar a la següent zona de la pantalla de l'entorn 3D,



i activar la casella corresponent. Seguidament ens apareixerà la finestra de control del model:

Fa que el punt de visió de tots els usuaris passi a ser el nostre.

Al prémer aquest botó la finestra canviarà per la que ve a continuació:

Retorna a cada usuari el seu punt de visió original.

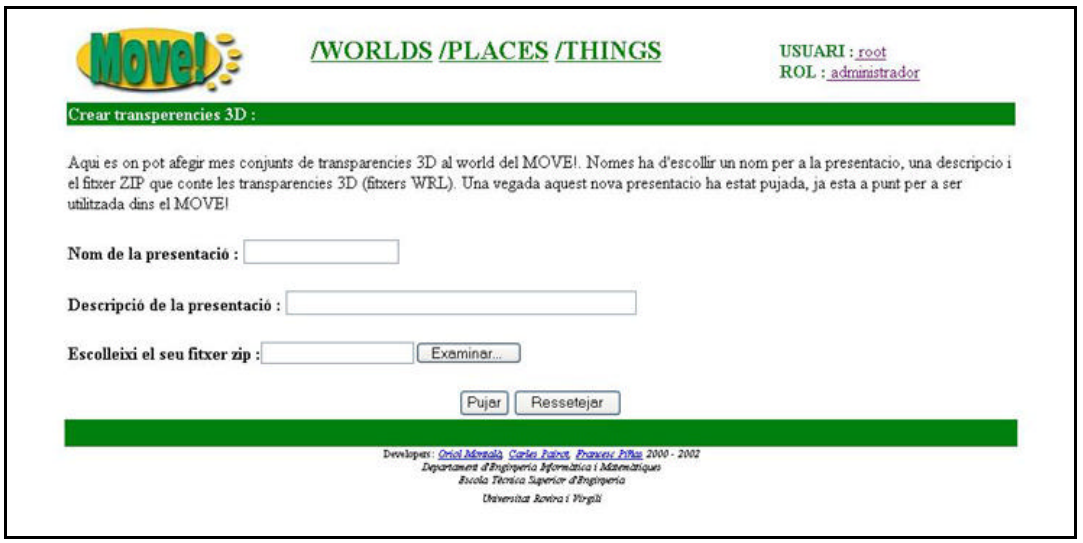
7.3.6 Simulation3D

La Tool **simulation3D** permetrà pujar conjunts de simulacions 3D al MOVE!, per a poder visualitzar i interactuar amb aquestes simulacions un cop dins l'entorn 3D del Place.

Un *conjunt de simulacions 3D* consistirà en un grup de fitxers **VRML**.

7.3.6.1 Pujar simulació 3D

Per tal de crear una Thing de tipus simulation3D el que haurem de fer és clicar sobre l'icona  que apareix en la barra de tools actives i seguidament ens apareixerà la següent pantalla:



Per a poder crear la Thing hem d'omplir els següents camps:

- **Nom de la presentació:** nom per al conjunt de simulacions 3D.
- **Descripció de la presentació:** descripció de la presentació de simulacions 3D.
- **Escolleixi el seu fitxer zip:** fitxer **ZIP** que conté el conjunt de simulacions 3D (fitxers **WRL**).

Un cop introduides les dades en el petit formulari ens cal prémer el botó  per tal que la nova Thing sigui creada i pujada al servidor del MOVE. Si tot ha anat bé serem redirigits a la pantalla on es mostra la llista de Things que hi ha en el Place, i observarem que la Thing que hem creat ja apareix entre elles.

7.3.6.2 Funcionament del model simulation3D

El model audio apareix en l'entorn 3D representat per el següent objecte:

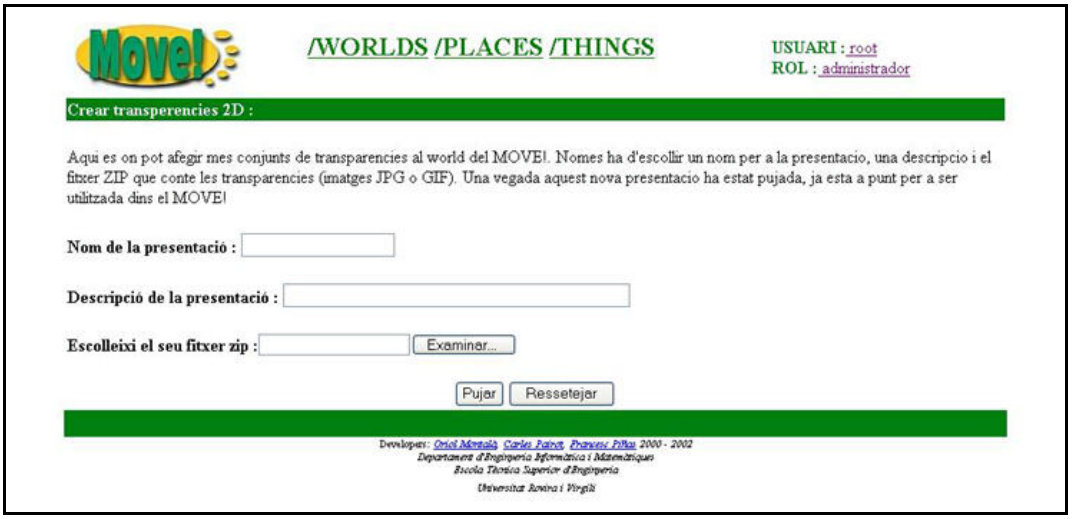
Per tal d'accedir al control del model simulation3D sols haurem de clicar sobre ell i ens apareixerà la següent finestra:

7.3.7 Slides

Amb la Tool **slides** tindrem la possibilitat de pujar conjunts de transparències (imatges **JPG** o **GIF**) al MOVE. Després podrem carregar i visualitzar les nostres presentacions sobre el model slides, un cop dins l'entorn 3D del MOVE. També podrem veure totes les transparències que formen la nostra presentació sense necessitat d'entrar a l'entorn 3D.

7.3.7.1 Pujar conjunt de transparències

Per tal de crear una Thing de tipus slides el que haurem de fer és clicar sobre l'icona  que apareix en la barra de tools actives i seguidament ens apareixerà la següent pantalla:



Per a poder crear la Thing hem d'omplir els següents camps:

- **Nom de la presentació:** nom per al nostre conjunt de transparències.
- **Descripció de la presentació:** descripció de la presentació.
- **Escolleixi el seu fitxer zip:** fitxer **ZIP** que conté el conjunt de transparències (imatges **JPG** o **GIF**).

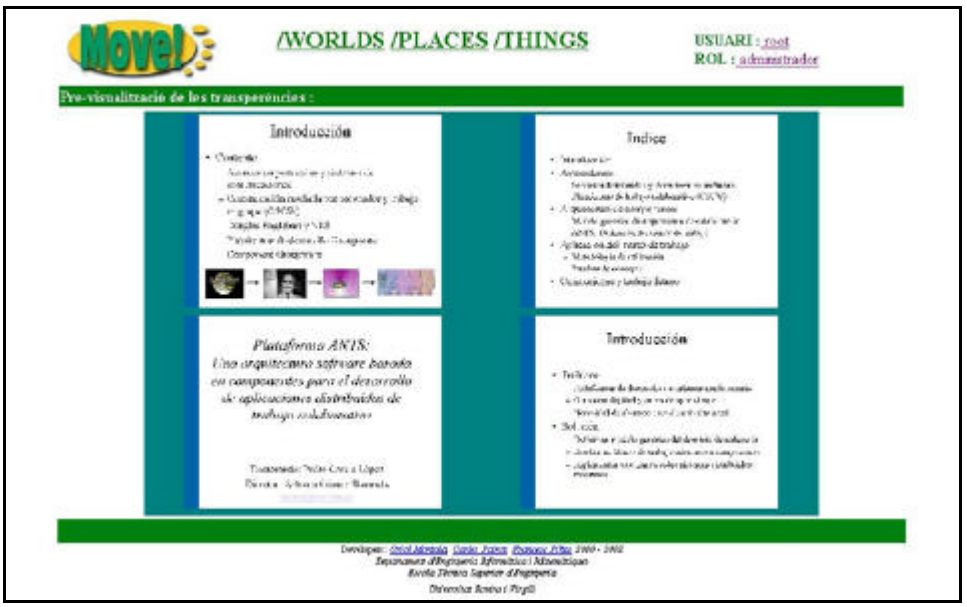
Un cop introduides les dades en el petit formulari ens cal prémer el botó 

per tal que la nova Thing sigui creada i pujada al servidor del MOVE. Si tot ha anat bé serem redirigits a la pantalla on es mostra la llista de Things que hi ha en el Place, i observarem que la Thing que hem creat ja apareix entre elles.

7.3.7.2 Pre-visualització de la presentació

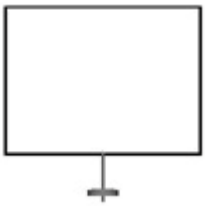
Per poder veure la Thing de tipus slides sense necessitat d'entrar a l'entorn 3D de MOVE! només cal clicar sobre la seva icona en la taula on apareixen totes les Things que hi ha en el Place.

Un cop fet això veurem aquesta pantalla, on podrem veure tot el conjunt de transperències que formen la presentació.



7.3.7.3 Funcionament del model slides

El model slides apareix en l'entorn 3D representat per el següent objecte:



Per tal d'accedir al control del model slides sols haurem de clicar sobre ell i ens apareixerà la següent finestra:

Ens mostra una llista de totes les Things de tipus slides que hi ha creades actualment en el Place.

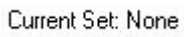
Carrega el conjunt de transparències que hi ha seleccionat en el camp *Slides Sets.*

Ens mostra la següent transparència de la presentació.

Ens mostra l'anterior transparència de la presentació.

Anem a la primera transparència de la presentació.

Anem a l'ultima transparència de la presentació.



Indica el nom de la presentació que hi ha actualment carregada.

7.3.8 URL

La Tool **URL** ens permetrà crear URL's i deixar-les dins l'entorn 3D del Place, al lloc que desitgem. Podrem agafar una URL, portar-la a un altre lloc, moure-la, i visitar-la.

7.3.8.1 Crear URL

Per tal de crear una Thing de tipus URL el que haurem de fer és clicar sobre l'icona  que apareix a la pantalla de l'entorn 3D, sota el logotip de MOVE!. Se'ns obrirà una finestra com aquesta:

Per a poder crear la Thing hem d'omplir els següents camps:

- **URL:** URL
- **Descripció:** descripció de la URL

Un cop introduides les dades en el petit formulari ens cal prémer el botó 

per tal que la nova Thing sigui creada i pujada al servidor del MOVE. Si tot ha anat bé la finestra es tancarà i veurem com dins el Place on estem apareix la representació 3D de la nova URL.

7.3.8.2 Visitar URL

Per poder visitar la Thing de tipus URL sense necessitat d'entrar a l'entorn 3D de MOVE! només cal clicar sobre la seva icona en la taula on apareixen totes les Things que hi ha en el Place.

7.3.8.3 Funcionament del model URL

El model URL apareix en l'entorn 3D representat per el següent objecte:

Per tal d'accedir al control del model URL sols haurem de clicar sobre ell i ens apareixerà la següent finestra:

Ens permet visitar la URL.

Ens permet agafar la URL i portar-la a un altre lloc.

Ens permet canviar la posició actual de la URL.

Elimina la URL del MOVE.

URL.

Descripció del que ens trobarem en la URL.

Data en què ha estat creada la URL.

Usuari que ha creat la URL.

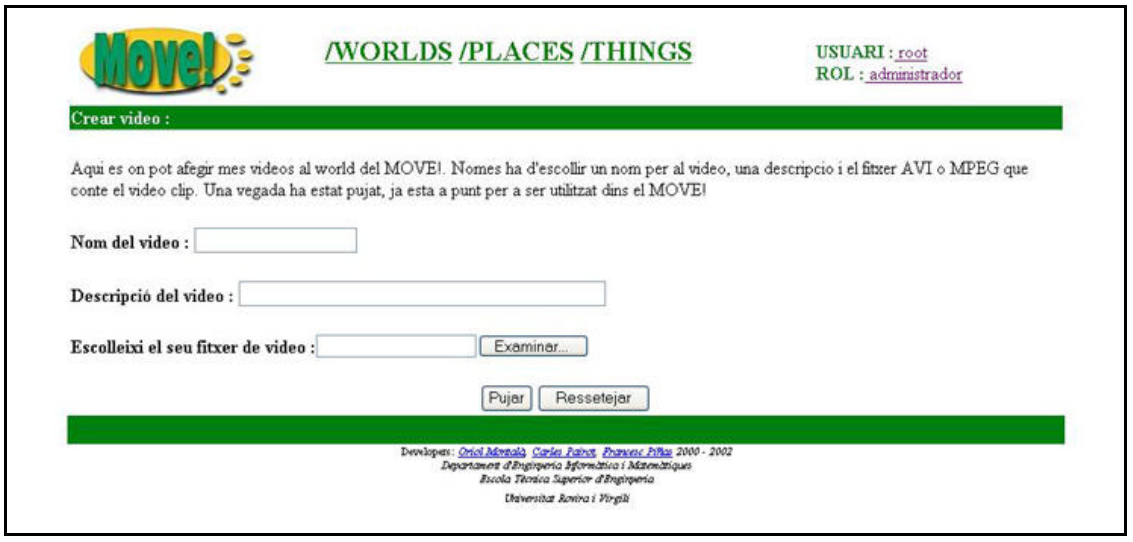
7.3.9 Video

La Tool **video** permetrà pujar videos multimedia al MOVE! i visualitzar-los després dins l'entorn 3D del Place.

També podrem veure els video clips sense necessitat d'entrar a l'entorn 3D.

7.3.9.1 Pujar video

Per tal de crear una Thing de tipus video el que haurem de fer és clicar sobre l'icona  que apareix en la barra de tools actives i seguidament ens apareixerà la següent pantalla:



Per a poder crear la Thing hem d'omplir els següents camps:

- **Nom del video:** nom del video clip.
- **Descripció del video:** descripció del video clip.
- **Escolleixi el seu fitxer de video:** fitxer multimedia que contindrà el video clip (fitxer **AVI** o **MPEG**).

Un cop introduides les dades en el petit formulari ens cal prémer el botó 

per tal que la nova Thing sigui creada i pujada al servidor del MOVE.

Si tot ha anat bé serem redirigits a la pantalla on es mostra la llista de Things que hi ha en el Place, i observarem que la Thing que hem creat ja apareix entre elles.

7.3.9.2 Pre-visualització del video

Per poder veure la Thing de tipus video sense necessitat d'entrar a l'entorn 3D de MOVE només cal clicar sobre la seva icona en la taula on apareixen totes les Things que hi ha en el Place.

7.3.9.3 Funcionament del model video

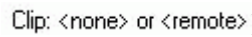
El model video apareix en l'entorn 3D representat per el següent objecte:

Per tal d'accedir al control del model video sols haurem de clicar sobre ell i ens apareixerà la següent finestra:

Ens mostra una llista de totes les Things de tipus video que hi ha creades actualment en el Place.

Comença a reproduir el video.

Finalitza la reproducció del video.



Indica el nom del video que hi ha actualment carregat en el model.

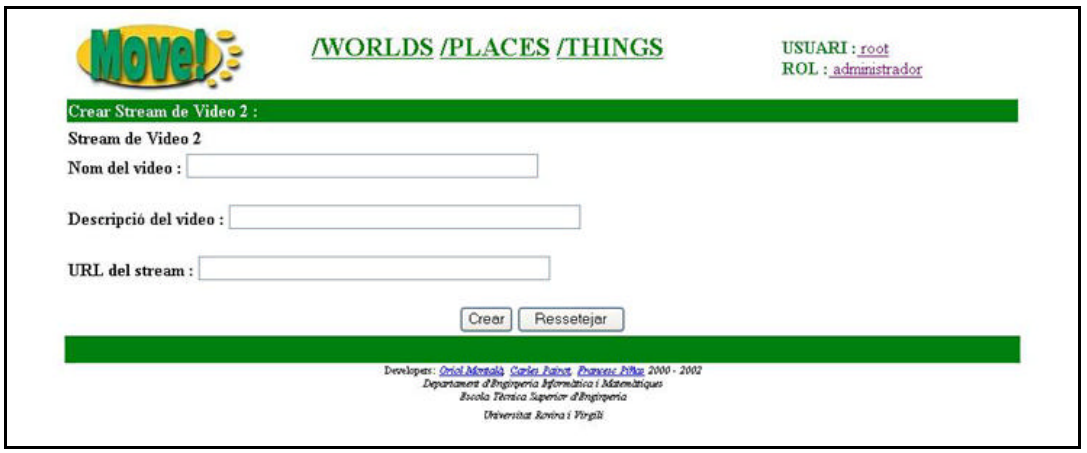
7.3.10 VideoStream2

La Tool **videoStream** ens permetrà crear streams de vídeo i reproduir-los després dins l'entorn 3D del Place.

També podrem veure els streams fora de l'entorn 3D.

7.3.10.1 Crear Stream de video 2

Per tal de crear una Thing de tipus videoStream2 el que haurem de fer és clicar sobre l'icona  que apareix en la barra de tools actives i seguidament ens apareixerà la següent pantalla:



Per a poder crear la Thing hem d'omplir els següents camps:

- **Nom del video:** nom del video clip.
- **Descripció del video:** descripció del video clip.
- **URL del stream:** URL del stream del video.

Un cop introduides les dades en el petit formulari ens cal prémer el botó  per tal que la nova Thing sigui creada i pujada al servidor del MOVE. Si tot ha anat bé serem redirigits a la pantalla on es mostra la llista de Things que hi ha en el Place, i observarem que la Thing que hem creat ja apareix entre elles.

7.3.10.2 Pre-visualització del stream de video 2

Per poder visualitzar la Thing de tipus videoStream2 sense necessitat d'entrar a l'entorn 3D de MOVE! només cal clicar sobre la seva icona en la taula on apareixen totes les Things que hi ha en el Place.

7.3.10.3 Funcionament del model videoStream2

El model videoStream2 apareix en l'entorn 3D representat per el següent objecte:

Per tal d'accedir al control del model videoStream2 sols haurem de clicar sobre ell i ens apareixerà la següent finestra:

Ens mostra una llista de totes les Things de tipus videoStream2 que hi ha creades actualment en el Place.

Carrega el Stream de video que hi ha seleccionat en el camp *Video Streams.*

NOTA: el video stream NONE no es refereix a cap stream. El carregarem en el model quan no volguem cap stream de video en reproducció.

7.3.11 Votation

Amb la Tool **votation** podrem crear temes als quals tot usuari del nostre Place pugui donar la seva opinió en forma de vot.

Cal tenir en compte que, com a votació real, sols ens serà permès votar una sola vegada.

7.3.11.1 Crear votació

Per tal de crear una Thing de tipus votation el que haurem de fer és clicar sobre l'icona 

que apareix en la barra de tools actives i seguidament ens apareixerà la següent pantalla:

Per a poder crear la Thing hem d'omplir el següent camp:

- **Tema:** tema al qual farà referència la votació.

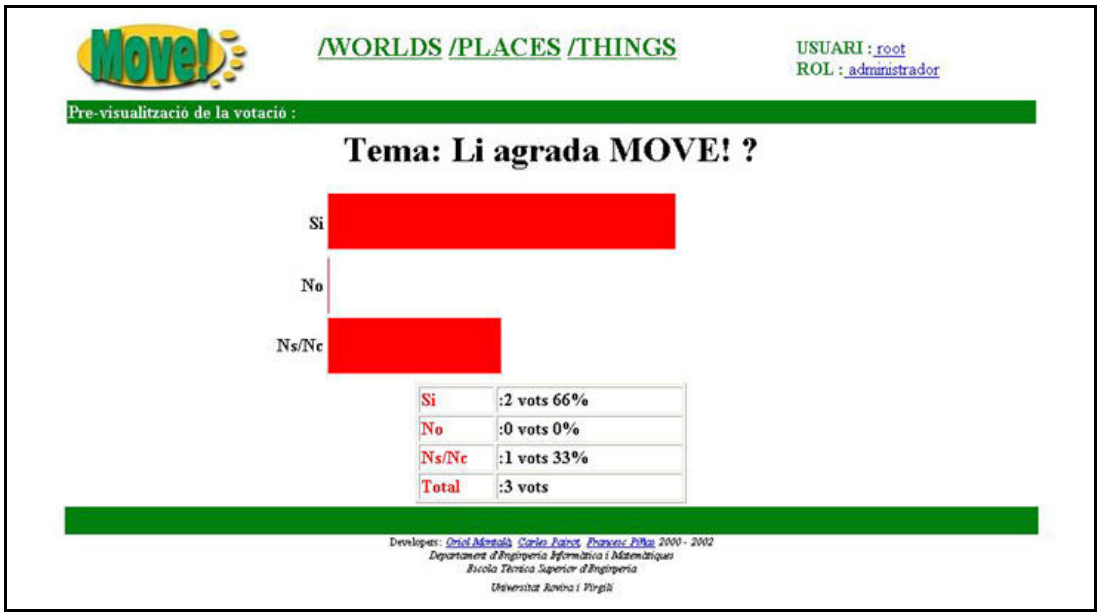
Un cop introduides les dades en el petit formulari ens cal prémer el botó 

per tal que la nova Thing sigui creada i pujada al servidor del MOVE. Si tot ha anat bé serem redirigits a la pantalla on es mostra la llista de Things que hi ha en el Place, i observarem que la Thing que hem creat ja apareix entre elles.

7.3.11.2 Pre-visualització de la votació

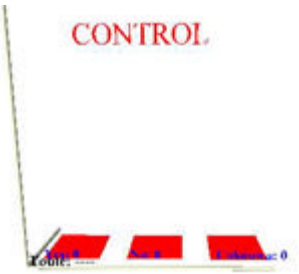
Per poder veure la Thing de tipus votation sense necessitat d'entrar a l'entorn 3D de MOVE! només cal clicar sobre la seva icona en la taula on apareixen totes les Things que hi ha en el Place.

Un cop fet això veurem aquesta pantalla, on podrem veure l'estat actual en què es troba la votació.

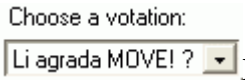


7.3.11.3 Funcionament del model votation

El model votation apareix en l'entorn 3D representat per el següent objecte:



Per tal d'accedir al control del model votation sols haurem de clicar sobre ell i ens apareixerà la següent finestra:



Ens mostra una llista de totes les Things de tipus votation que hi ha creades actualment en el Place.

Carrega la votació que hi ha seleccionada en el camp *Topic.*

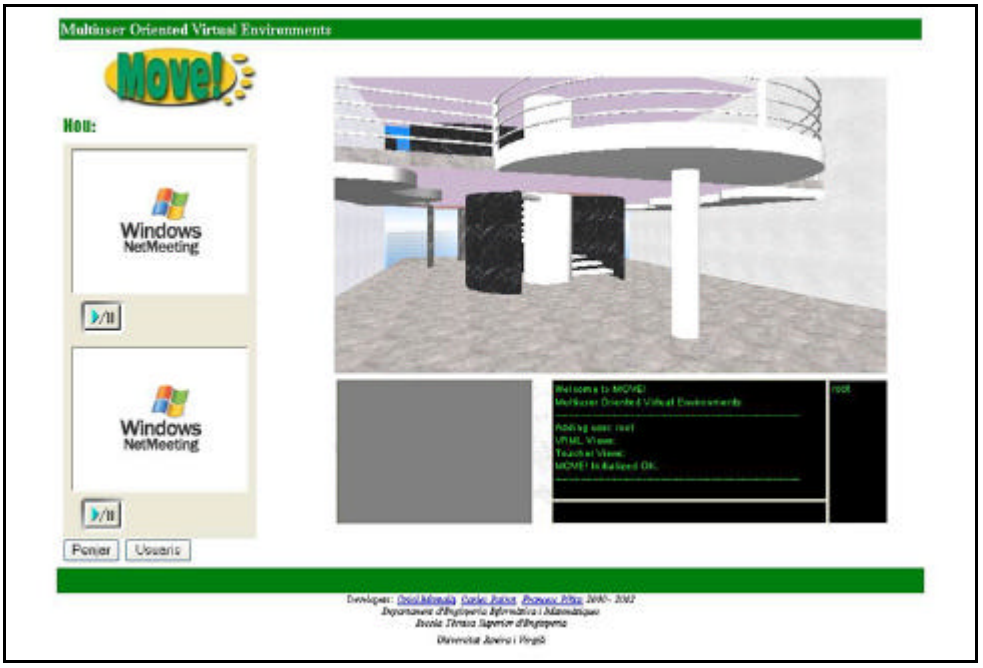
Per a poder donar el nostre vot haurem de clicar sobre alguna de les 3 barres del model que representen els 3 tipus de vots que podem efectuar:

- **SI:** vot a favor de la votació.
- **NO:** vot en contra de la votació.
- **NsNc:** vot en blanc.

7.3.12 Netmeeting

La Tool **Netmeeting** utilitza les propietats de components ActiveX que ofereix el **Windows NetMeeting** de Microsoft per tal de permetre videoconferència entre usuaris d'un mateix Place dins el MOVE.

Si el nostre rol té permís d'utilització d'aquesta Tool dins un determinat World, i a més la hem activat per tal de poder-la usar dins un determinat Place, veurem que, quan entrem a l'entorn 3D del Place, aquesta apareixerà en el marge esquerra de la pantalla:



Com podem veure la Tool consta de les següents parts:

Pantalla de **video remot**. En ella podrem veure el video que estem rebent de l'usuari amb el que estem mantenint la videoconferència

Pantalla de **video local**. En ella apareixerà el vídeo que estem enviant a l'altre usuari

Finalitza la trucada que haviem establert amb l'usuari remot.

Mostra la següent finestra, la qual conté una llista de tots els usuaris que hi ha actualment en el Place:

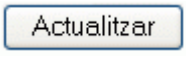
USUARIS EN AQUEST PLACE:

Usuari	Nom	Cognoms	IP	
root	root	root	127.0.0.1	Trucar

Actualitzar

- **Usuari:** login que identifica a l'usuari.
- **Nom:** nom de l'usuari.
- **Cognoms:** cognoms de l'usuari.
- **IP:** direcció IP de la màquina de l'usuari.

Intenta establir una connexió entre l'usuari remot i nosaltres.



Actualitza la taula d'usuaris.

7.3.13 Builder

La Tool **Builder** ens permetrà personalitzar els nostres Places per tal que tinguin l'aspecte que realment desitgem.

Podrem afegir dos tipus d'objectes diferents al Place:

- **objectes estàtics:** Objectes tals com cadires, taules, làmperes, etc, que seran carregats sempre al Place un cop afegits.
- **models:** Objectes tals com pantalles de transperències, ràdios, equips de vídeo, etc, els quals correspondran als models 3D de les Tools, i que seran carregats dinàmicament al Place, depenent de si la Tool a la que representen està activa o no.

L'aspecte del Builder és el següent:

Per una banda tenim l'entorn 3D que representa el nostre Place.

Per altra banda tenim els controls que ens ofereix la Tool per tal d'interectuar amb el Place.

A continuació descriurem breument tot el que ens ofereix el control del Builder:

save guarda els canvis efectuats en el Place.

add afegeix al Place l'objecte que tenim seleccionat.

remove elimina del Place l'objecte que tenim seleccionat.

 permet moure l'objecte amunt-avall, esquerra-dreta.

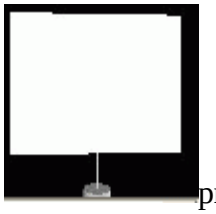

permet apropar o allunyar de nosaltres l'objecte.

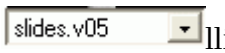

augmenta el tamany de l'objecte.

disminueix el tamany de l'objecte.


permet realitzar una rotació, negativa o positiva, de l'objecte sobre l'eix de coordenades 

que tenim actualment seleccionat.


pre-visualització de l'objecte que tenim actualment seleccionat.

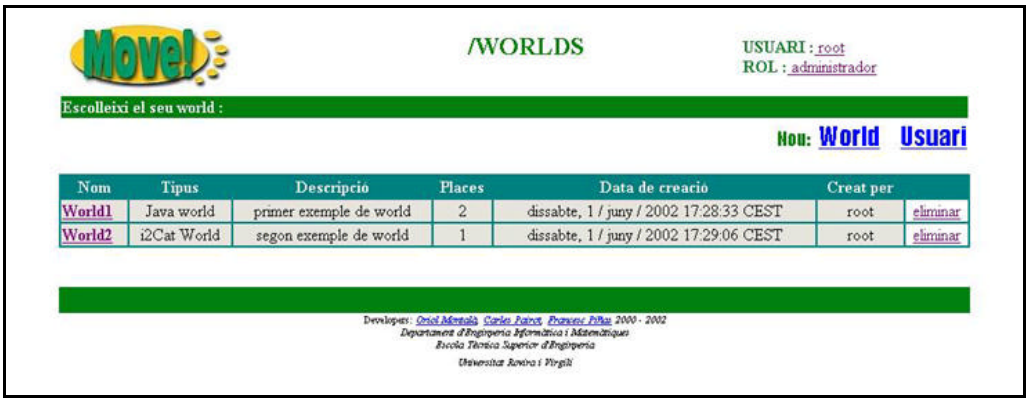


llista d'objectes que podem afegir al nostre Place.

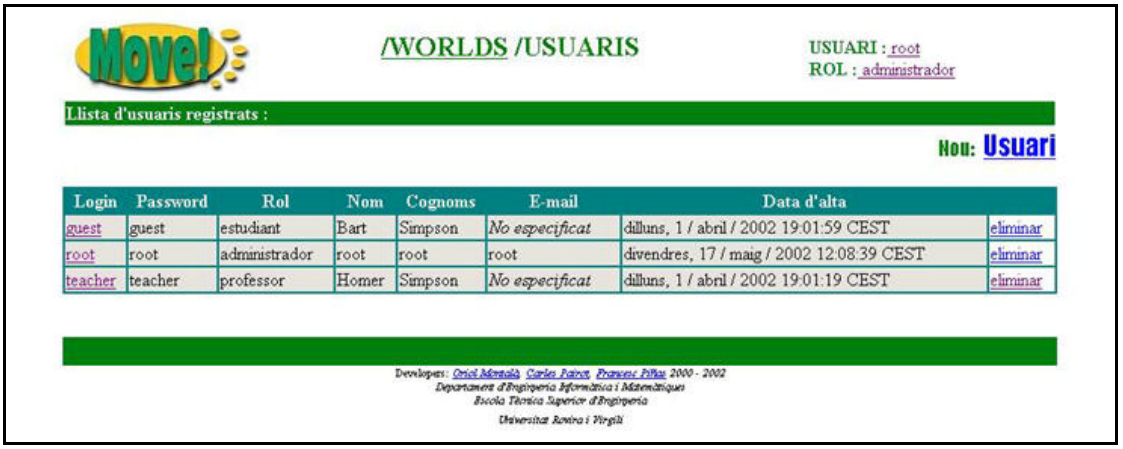
8. Manual de l'administrador

8.1 Informació sobre els usuaris registrats

Qualsevol usuari que entri al MOVE! com a **administrador**, es trobarà que a la pantalla on es mostren els tots els worlds disponibles, a part d'aparèixer l'opció de crear un nou World, ara també apareix una nova opció, **Usuari**



Si cliquem sobre aquesta opció anirem directament a la zona **USUARIS**.



En aquesta pantalla podem veure un llistat de tots els usuaris que hi ha registrats al MOVE i informació sobre cadascun d'ells:

- **Login:** login per a identificar a l'usuari.
- **Password:** contrasenya d'usuari per tal de poder entrar.
- **Rol:** tipus d'usuari:
  - Administrador.
  - Professor.
  - Estudiant.

- Depenent del tipus d'usuari aquest tindrà més o menys privilegis dins el MOVE.

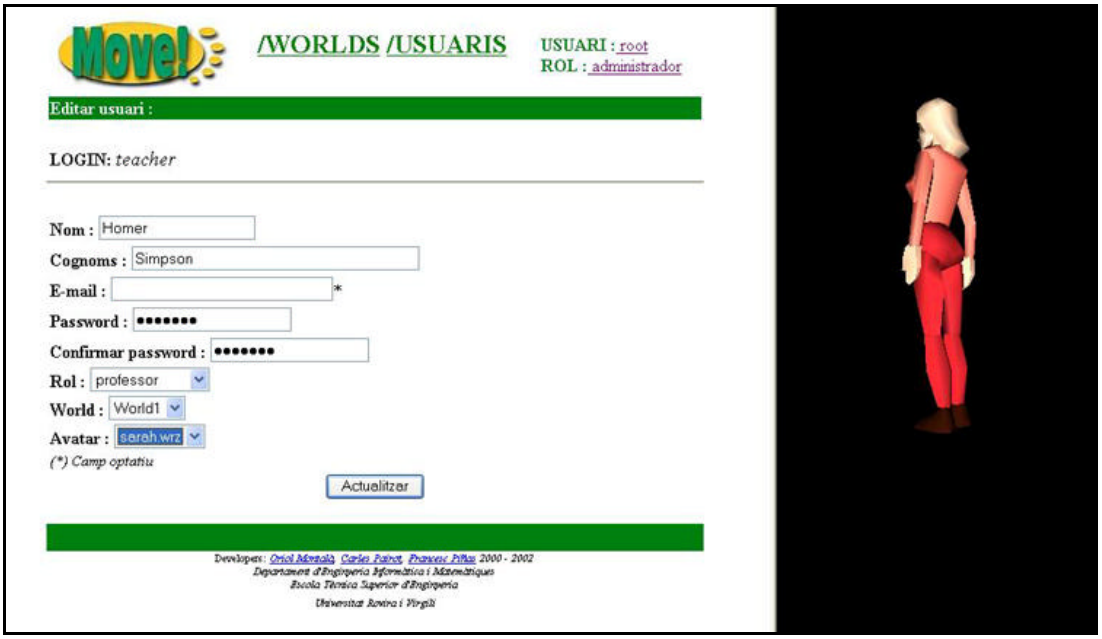
- **Nom:** nom de l'usuari.
- **Cognoms:** cognoms de l'usuari.
- **E-mail:** adreça de correu electrònic on poder localitzar l'usuari.
- **Data d'alta:** data en què l'usuari ha estat registrat en el MOVE per un administrador.

Podrem eliminar qualsevol dels usuaris clicant sobre eliminar o bé modificar-ne les seves propietats clicant sobre el seu **login**.

Si per el contrari el que volem és crear un nou usuari haurem de clicar sobre Usuari

8.2 Modificar les dades d'un usuari existent

En la pantalla d'edició de les propietats d'un usuari ens trobem el següent:



A la part esquerra podem veure el perfil de l'usuari,

- **Login**
- **Nom**
- **Cognoms**
- **E-mail**
- **Password**

- **Rol**

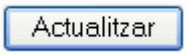
i dues noves propietats que encara no haviem vist:

- **World:**
- **Avatar:** aparença que tindrà l'usuari dins l'entorn 3D del MOVE!

A la part dreta de la pantalla podem veure una pre-visualització del avatar (humanoide 3D) que hi ha actualment seleccionat en el camp *Avatar*.

A l'hora de modificar les dades actuals de l'usuari caldrà tenir en compte una sèrie de coses:

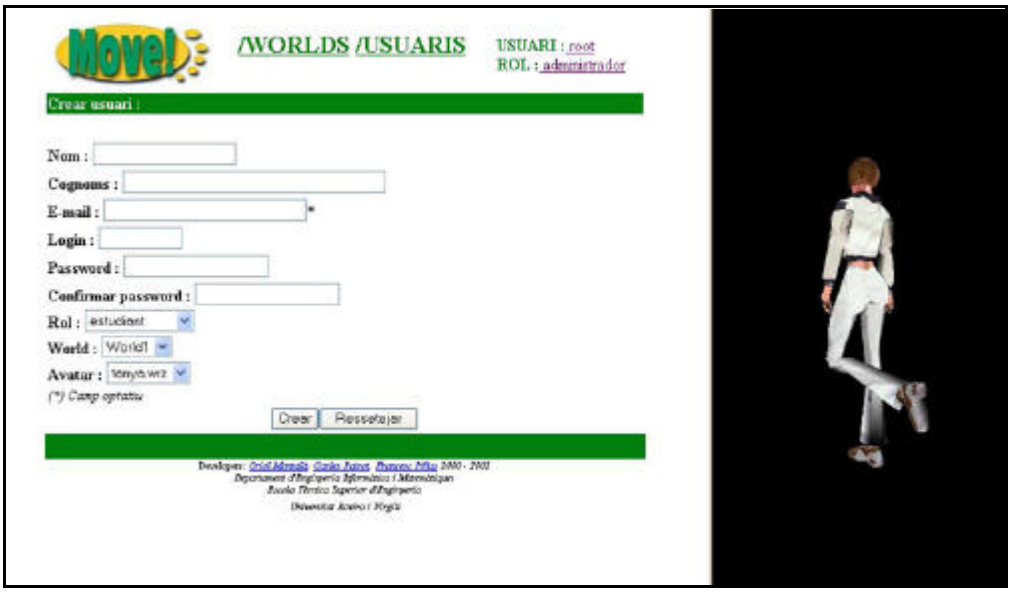
- el login és **immodificable**.
- tots els camps són **obligatoris**, excepte el camp *E-mail.*
- tan el valor del camp *Password* com el del camp *Confirmar Password* hauran de **coincidir**.

Una vegada hagim acabat d'actualitzar les dades de l'usuari i comprovat que efectivament són correctes, caldrà prémer el botó 

i les noves dades de l'usuari seran guardades dins la Base de Dades del MOVE.

8.3 Registrar un nou usuari

Quan elegim l'opció de *registrar un nou usuari* ens apareix la següent pantalla:



Igualment com passava amb la pantalla de *modificació de dades d'un usuari*, aqui també tenim la pantalla dividida en 2 parts:

- La part que correspon al formulari que haurem d'omplir per a crear el nou usuari.
- La part de pre-visualització de l'aparença que tindrà l'usuari en l'entorn 3D del MOVE, i que correspon amb el valor actual de la casella *Avatar*.

Totes les caselles que apareixen en el formulari s'han d'omplir obligatoriament, excepte la que correspon al *E-mail*, la qual és optativa.

El significat de cada casella ja ha estat descrit amb anterioritat dins d'aquest capítol.

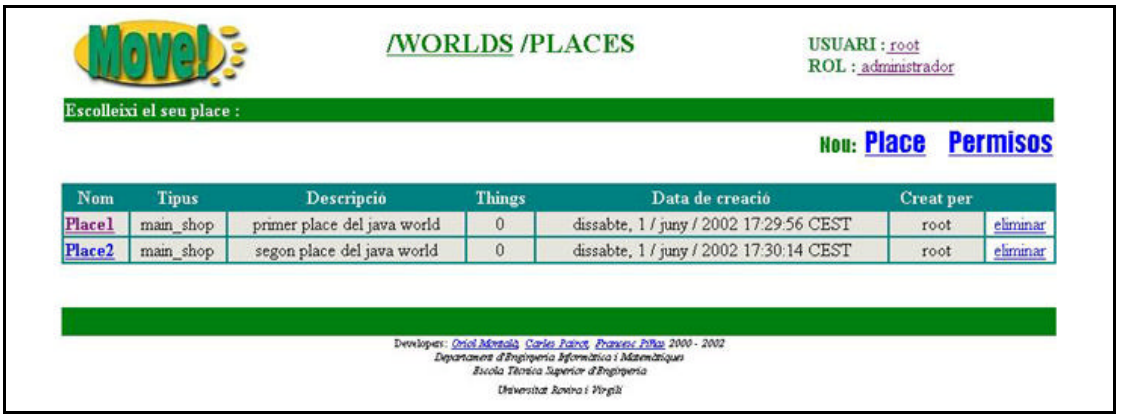
Un cop finalitzada la introducció de les dades, cal prémer el botó 

i el nou usuari passarà a formar part de la llista d'usuaris que hi ha registrats en el MOVE.

8.4 Definir permisos en un World

El primer que cal fer és dir que entenem per **Permisos**. Doncs bé, definirem *permisos que té cada rol en un World* com a les tools que poden utilitzar tots els usuaris que pertanyen a aquest rol dins un determinat World. Només es podran definir permisos per als rols, **Estudiant** i **Professor**. El rol **Administrador** tindrà accés a totes les tools en qualsevol dels Worlds. Un cop dit això ja estem en disposició de definir permisos en un World.

El primer que cal fer és entrar en un World.



Una vegada dins el World observarem que en la pantalla on apareixen tots els Places que conté aquest World, ha aparegut una nova opció, 

.Si cliquem sobre aquesta opció entrarem a la pantalla on podrem definir els permisos de cada rol sobre les Tools.

Nom	Descripció	Tipus de thing	Estat	Acció
Audio	Tool Audio	Audio.v01	activada	desactivar
Banner	Tool Banner	Banner.v01	desactivada	activar
Builder	Tool Builder	Builder.v01	desactivada	activar
Camera	Tool Camera	Camera.v02	desactivada	activar
Document	Tool Document	Doc.v01	activada	desactivar
Hook	Tool Hook	Hook.v01	desactivada	activar
NetMeeting	Tool NetMeeting	NetMeeting.v01	activada	desactivar
Simulació 3D	Tool Simulació 3D	S3d.v01	desactivada	activar
Transperències	Tool Transperències	Slides.v05	activada	desactivar
URL	Tool URL	Url.v01	activada	desactivar
Video	Tool Video	Video.v01	activada	desactivar
Stream de Video 2	Tool Stream de Video Versió 2	VideoStream.v02	activada	desactivar
Votació	Tool Votació	Votation.v01	activada	desactivar

Observarem que apareix una taula amb totes les Tools que hi ha disponibles en el MOVE, i a la part superior d'aquesta, els noms dels dos rols sobre els quals podem definir els permisos. El nom del rol que està en cursiva és al que pertany la informació actual de la taula.

Per a cada Tool tenim la següent informació:

- **Nom:** nom de la Tool.
- **Descripció:** descripció de la Tool.
- **Tipus de Thing.**
- **Estat:** estat en què es troba actualment la Tool per aquest rol.
- **Acció:** Ens permet activar o desactivar la Tool per tal de que pugui ser utilitzada o no per aquest rol.

9. Conclusions i vies futures

S'ha experimentat com les tecnologies existents poden crear nous entorns amb grans prestacions que reforçaran la interacció de l'usuari. Hem creat i integrat dues eines que poden ser molt útils de cara al futur de la creació de móns multiusuari. En primer lloc, amb el mòdul creador de móns *VRML* es poden crear habitacions senzilles a on col.locar diversos objectes de manera molt visual. Tot i les limitacions de *VRML* i *EAI* hem pogut crear una aplicació força consistent i que simplifica molt la tediosa feina d'escriure a ma el codi font *VRML*. Amb el mòdul multiusuari que integra els diferents models, aconseguim fer que aquests móns siguin més interactuables per part dels usuaris i alhora s'aconsegueix recrear un ambient òptim per a practicar-hi tècniques de tele-ensenyament. La part de l'aplicació web s'ha desenvolupat per crear una mena de *frontend* cap a l'aplicació tridimensional distribuïda.

Potser les seves aplicacions més interessants es trobin en els *Ambients Virtuals Col.laboratius*, els quals possibiliten que vàries persones localitzades en llocs diferents puguin portar a terme alguna forma de col.laboració.

Les expectatives derivades d'aquest sistema continuen sent molt ambicioses, ja que els camps d'investigació són nombrosos i estan en contínua evolució. En els pròxims anys es veurà com aquests nous espais de treball en 3D reforçaran la cooperació i el treball en una gran varietat d'àrees i més en especial en el tele-ensenyament. A més, tenint en compte la nova creació de l'especificació de *X3D* que millora aspectes de la interacció entre Java i *VRML* de tal manera que es podrà tenir una major flexibilitat i potència per refinar encara més aquest tipus d'aplicacions.

Des del punt de vista educatiu, el projecte també planteja reptes importants. A continuació s'estudia el projecte des d'aquesta perspectiva. L'objectiu primordial era el de crear una aplicació que fos general i fàcil d'utilitzar. Ha de ser general perquè ha de ser capaç de soportar processos d'ensenyament i aprenentatge en diferents camps del saber. Ha de ser fàcil perquè la seva implantació en institucions educatives no ha de fer que es necessiti personal amb coneixements tècnics diferents a connectar un ordinador i executar una aplicació. Un cop dissenyada l'aplicació amb les condicions anteriors és necessari validar la seva utilitat des del punt de vista pedagògic.

És important assenyalar que el producte busca convertir-se ne una eina de suport per a l'ensenyament. En cap moment es planteja com una alternativa millor als processos d'ensenyament en els quals el professor es trobi físicament a lloc. De moment, però, apreciem els següents avantatges respecte a l'ambient educatiu tradicional (sense suport informàtic):

- Se superen les barreres de l'espai físic presents en els mètodes d'ensenyament tradicionals. Aquells alumnes que es troben, per qualsevol motiu, impossibilitats per accedir a les instal.lacions universitàries podran seguir la classe des del seu ordinador.
- Es poden aplicar diversos enfocaments metodològics.
- L'estudiant es motiva més ja que està utilitzant una eina novedosa.

A més dels importants reptes tècnics que queden per superar, potser l'aport més important, pedagògicament parlant, per a aconseguir l'objectiu d'un ús massiu d'aquestes aplicacions en el nostre país, sigui trobar els models pedagògics que millor funcionin amb aquest tipus de tecnologies.

No obstant, des del punt de vista pedagògic aquest tipus d'entorns són encara un terreny sense explorar. Moltes són les preguntes que encara no tenen resposta i que poden donar origen a posteriors projectes d'investigació:

Quines són les pedagogies més apropiades per a aquest tipus d'aplicacions?

Quines àrees de coneixement són les més apropiades perquè s'ensenyin utilitzant aquest tipus d'eines?

Com avaluar els processos d'ensenyament i aprenentatge en els quals s'utilitzen aquestes eines?

9.1 Vies futures

Degut a la gran envergadura del projecte, han quedat sense explorar moltes línies d'investigació de gran interès relacionades amb el sistema. Queda obert així, el camí a noves investigacions a partir de les ja realitzades que, per a facilitar el camí als següents treballs, s'enumeren a continuació:

- **Integració de veu:** Seria possible i molt avantatjós introduir un mitjà de comunicació entre els participants de l'escena basat en la veu. Ja existeixen línies d'investigació sobre aquest tema. Gràcies a això la comunicació milloraria molt i aproparia més el sistema a la realitat.

- **Implementació de bots intel.ligents:** Tal i com hem anat comentant al llarg d'aquesta documentació, es podrien implementar *bots* intel.ligents que poguessin

10. Bibliografia

[1] Web3D Consortium, URL http://www.vrml.org/vrb

[2] VRML 2.0: Information technology – The Virtual Reality Modeling Language International Standard ISO/IEC 14772-1:1999, URL http://www.web3d.org/Specifications/VRML97

[3] Roehl, Couch: Late Night VRML 2.0 with Java, ZD Press, ISBN 1-56276-504-3.

[4] PLATINUM technology, EAI, URL http://www.cosmosoftware.com/products/player/developer/eai

[5] Sun Microsystems, The Source for Java Technology, Java3D 1999, URL http://java.sun.com/products/java-media/3D/

[6] The Mbone Network, URL : http://www.mbone.com/

[7] PLATINUM technology, INTERVISTA, URL http://www.intervista.com/vrml/

[8] LIVING WORLDS, Making VRML 97 Applications, 9 Juny 1998, URL http://www.vrml.org/WorkingGroups/living-worlds/

[9] Grup de Treball *Living Worlds*, Web3D Consortium http://www.web3d.org/vrb/living-worlds.html

[10] The Java3D and VRML Working Group, 15 Juliol 1999, URL http://www.vrml.org/WorkingGroups/vrml-java3d/

[11] Elvin4 Notification Service, URL http://elvin.dstc.edu.au/

[12] VRML Works, Bob Crispen, URL http://home/hiwaay.net/~crispen/vrmlworks

[13] Rational Software Corporation "Scope of the UML", URL http://mirac.ee.ncku.edu.tw/~canseco/tech/uml/summary/summary4.html

[14] Pablo Figueroa's Site - UML 1.997, URL http://agamenon.uniandes.edu.co/~pfiguero/soo/uml/

[15] VRML 1.0: VRML 1.0c Specification, URL, http://www.vrml.org/Specifications/VRML1.0

[16] Sun Microsystems, Inc 1995-2000. URL http://java.sun.com/

[17] EAI: External Authoring Interface Working Group, URL http://www.vrml.org/WorkingGroups/vrml-eai

[18] Ambients Virtuals Col.laboratius, Universitat EAFIT, URL http://sigma.eafit.edu.co/~virtualc/intro_esp.htm

[19] EAI: External Authoring Interface, URL http://www.cosmosoftware.com/products/player/developer/eai/

[20] The VRML Repository, URL http://www.web3d.org/vrml/vrml.htm

[21] VRMLSite Aereal, Inc.1997, URL http://www.vrmlsite.com/

[22] About.com, Inc. 1999, CMP Media, Inc. 1999, URL http://web3d.about.com/compute/web3d/

[23] Booch, Grady; Rumbaugh, James; Jacobson, Ivar: El Lenguaje Unificado de Modelado, Addison Wesley, ISBN 84-7829-028-1.

[24] Orion Application Server Website, URL http://www.orionserver.com

[25] VRML 97 Specification, URL http://www.vrml.org/Specifications/VRML97/index.html

[26] X3D Specification, ISO/IEC 14772:200x, URL http://www.web3d.org/TaskGroups/x3d/specification/

[27] Vacca, John, VRML Clearly Explained, 2nd Edition, AP Professional, ISBN 0-12-710008-3

[28] ActiveWorlds Website, URL http://www.activeworlds.com/

[29] ActiveWorlds Report, URL http://www.bartlett.ucl.ac.uk/ve/Work9900/michiel/ActiveWorlds.htm

[30] blaxxun Interactive Website, URL http://www.blaxxun.com/

[31] García, Pedro; Gómez Skarmeta, Antonio; Montalà, Oriol; Pairot, Carles; Rallo, Robert; MOVE: Component Groupware Foundations for Collaborative Virtual Environments. Proceedings del congrés CVE2002 que se celebrarà del 30/9 al 2/10 de 2002 a Bonn, Alemanya.

[32] Montalà, Oriol; Pairot, Carles; World Builder, Multiuser World Applet. Projecte de Final de Carrera, DEIM, ETSE, URV. Setembre 2000.

[33] DIVE Homepage. http://www.sics.se/dive.

[34] Garcia P., Skarmeta A., Rallo R., ANTS: A new Collaborative Learning Framework, European Conference on Computer Supported Collaborative Learning. 2001.

[35] Greenhalgh, C., Purbrick, J and Snowdon, D. Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring. Proceedings of the CVE 2000 Symposium, ACM, 2000, 119-127

[36] H-Anim Humanoid Working Group. http://www.h-anim.org

[37] Hofte, G. Henri ter. Working Apart Together: Foundations for Component Groupware. Telematica Instituut, Enschede, The Netherlands, 1997, 288.

[38] Marsic, I. DISCIPLE: A Framework for Multimodal Collaboration in Heterogeneous Environments. ACM Comp. Surveys, vol 31, no. 2es, 1999.

[39] Mitsubishi Electric Research Laboratories. Scalable Platform for Large Interactive Networked Environments (SPLINE). http://www.merl.com/projects/spline/.

[40] NPSNET Research Group. http://www.movesinstitute.org/~npsnet/index.html.

[41] Oliveira, M., Crowcroft, J. and Slater, M. Component Framework Infrastructure for Virtual Environments. Proceedings of the CVE 2000 Symposium. ACM, 2000, 139-146.

[42] Prinz, W. NESSIE: An Awareness Environment for Cooperative Settings. Proceedings of 6th European Conference on Computer Supported Cooperative Work. Kluwer Academic Publishers, 1999, 391-410.

[43] Roseman, M. and Greenberg, S. Building Real-time Groupware with Groupkit, a Groupware Toolkit. ACM Transactions on Computer-Human Interaction, 3(1):66—106, March 1996.

[44] Tower Project. http://tower.gmd.de.

[45] U.S. Department of Defense (DOD). Distributed Interactive Simulation, IEEE Standard 1278. 1993.

[46] Sun Microsystems, Java (TM) 2 Platform, Enterprise Edition, URL http://java.sun.com/j2ee

11. Annex A: Article MOVE per al congrés CVE2002

MOVE: Component Groupware Foundations for Collaborative Virtual Environments

Pedro García, Oriol Montalà, Carles Pairot and Robert Rallo
Department of Computer Science,
University Rovira i Virgili
Email: pgarcia@etse.urv.es, {omp.ei, cpg.ei}@estudiants.urv.es

Antonio Gómez Skarmeta
Department of Computer Engineering,
University of Murcia
Email: skarmeta@fcu.um.es

ABSTRACT

The design of a Virtual Environment (VE) is a distributed problem of multi-user access to shared resources. Such problem requires careful design decisions in order to provide a seamless system infrastructure capable of supporting flexible interactions in the shared scenarios.

The complexity of this domain has led to intrincate software systems that provide ad-hoc solutions to specific problems. Furthermore, many of them have gone to a dead end, due to their non-extensible design and their lack of code and module reuse.

This paper presents a VE that is constructed on top of a component groupware framework. Our major aim is to provide an extensible infrastructure offering a set of collaborative services in a seamless way. At the conceptual level, it provides essential collaborative services: shared sessions, support for synchronous and asynchronous components, security, coordination, and a server-side awareness infrastructure. At the architectural level, the framework is constructed on top of a middleware integration platform and uses high performant publish/subscribe notification services. Finally, we present the advantages and limitations of this approach.

Keywords
Virtual Environments, Frameworks, Component Groupware, Distributed Systems.

INTRODUCTION

In the last years we have been experiencing advances in networking technologies and protocols as well as database, computer graphics and display technology. As a consequence, a lot of Collaborative Virtual Environments have emerged thanks to the increasing computation capabilities of desktop computers as well as the enormous growth in network bandwidth and the ubiquity of the Internet.

In this line, many research efforts have tried to solve the distributed problems inherent to the design of VEs. Relevant examples of these VEs are DIVE [32], MASSIVE [34], NPSNET [39] or SPLINE [38]. Many of them have successfully addressed problems like world structuring and zone partitioning, state propagation, coordination and consistency, and interest management and awareness.

Nevertheless, the complexity of the domain has led to huge intrincate software systems that are difficult to extend and reuse, and thus precluding system interoperability and flexibility. To cope with these problems, the trend is to augment modularization in the design of such infrastructures.

JADE [40] contributes to augment design modularization proposing a generic component framework that outlines system interoperability. However, this proposal specifies a minimal kernel focused on inter-communication capabilities that does not offer collaborative services like session and zone management, state propagation, coordination and consistency, and awareness.

Another important problem in the design of VEs is the lack of awareness services enabling the extraction of information from the running infrastructure in order to better understand the group interactions by means of indicators or data-analysis tools. These awareness services can improve the shared experience and enable situational reactions to specific events in the multi-user environment.

In this line, the TOWER [43] project is developing an infrastructure to provide awareness by analysing events from groupware systems and visualizing them using Blaxxun technologies. On the other hand, MASSIVE-3 [34] includes Record and Replay mechanisms that permit to recover and reconstruct collaboration flows in the shared scenarios. Our approach goes further, we consider it is very important to design VEs considering those awareness services that enable situational reactions to events in the environment.

In order to solve the problems mentioned above, we construct our MOVE environment on top of a

component groupware framework called ANTS. This framework offers key collaborative services to components like session management, state propagation, security, coordination, and a server-side awareness infrastructure. Furthermore, the framework is constructed on top of a middleware integration platform and uses high performant publish/subscribe notification services. Software facades permit us to change the underlying middleware services without affecting the upper layers.

The paper is structured as follows: Section 2 overviews the MOVE environment and design objectives stated for this system. Section 3 describes the ANTS framework and presents the collaborative services offered to components. Section 4 explains in detail the MOVE architecture specifying how it benefits from the underlying framework. To finish with, in section 5 we draw conclusions of our current work and present the advantages and limitations of this approach.

MOVE OVERVIEW

MOVE is a 3D collaborative environment where users (Avatars) can interact with other users or with shared artifacts like voting tools, presenters, 3D simulations, documents or multimedia contents.

MOVE is one of the key applications being tested in the Catalonian Internet2 project and it is mainly deemed for educational purposes in different scenarios like virtual classroom simulations, archaeological or medical applications.

Its architecture benefits from ANTS Computer Supported Cooperative Work (CSCW) Framework. In addition, MOVE has been developed using open technologies such as VRML or the H-Anim v1.1 [35] avatar standard specification. The development language chosen has been Java and the bridge used to communicate this language with VRML has been the External Authoring Interface (EAI).

Problems in Existing 3D CVEs

Existing 3D Collaborative Virtual Environments (CVE) present certain problems that from our point of view, have not yet been completely resolved:

- The first problem is the high complexity in developing such a system of this kind. This type of environments must be easily scalable and should support as many concurrent users as possible degrading as least as possible system's performance. Existing 3D CVEs have dedicated many efforts in developing the environment itself rather than making it more scalable or making it component oriented so as new components may be added in the future. This is what we call component reusability.
- Another problem is data extraction. These CVEs were not thought for retrieving data in an easy way. Sometimes it can be interesting to grab data about what a specific user has been doing in our environment for data-mining, logging or capture and replaying purposes.

The solutions that MOVE adopts to solve the problems mentioned above are:

- Define a component model based on the standard JavaBeans specification that hides complexity to users providing transparently remote persistence, distributed events and component descriptors in XML and packaging. By adopting this component model we can create new tools that can be easily inserted into our system by means of "importing" them.
- Define a dedicated awareness and event monitoring service together with an agent system which reacts to the events triggered in the bus. By adopting this method, any kind of information can be obtained, such as logs, application of data-mining algorithms or usage of capture and replay techniques.

To achieve all these objectives, MOVE uses ANTS CSCW component framework. This infrastructure facilitates development of collaborative components, and thus achieving a smooth transition from local to distributed applications.

ANTS FRAMEWORK

The ANTS system aims to provide a generic multi-user collaborative framework. Being the overall problem a complex distributed challenge, it is necessary to solid distributed technologies as a basis as well as to provide generic collaborative services to the user and developer communities.

As we can see in Figure 1, the collaboration bus is a key piece of the overall architecture.

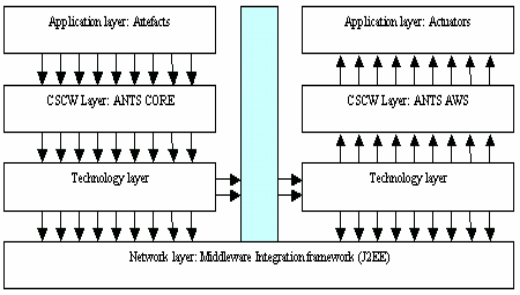


Figure 1. ANTS Architecture.

On the one hand, all components trigger events to this distributed information bus, and on the other hand, awareness components listen to the bus for information about what is happening in the system.

We can distinguish three main layers in the overall architecture: a technology layer, a CSCW service layer, and an application layer. Upper layers represent high level abstractions that hide complexity to users from the layers below.

In this line, the application layer is aimed to end users that extend the framework hotspots to create collaborative components and applications or monitoring agents. The CSCW layer comprises the set of collaborative services that the framework offers to application level components. Furthermore, these CSCW services are built on top of distributed services

offered by the underlying technology layer. Let us show their main functionalities:

- At the application layer, we have created a component model which abstracts access to a remote property-based persistent component, and to distributed events produced in the current session context.
- The CSCW layer comprises the set of collaborative services that the ANTS framework offers to the application layer in a coherent way. It thus provides session support, component's life cycle, communication abstraction, coordination support, and security. It also includes a seamless server-side platform for awareness actuators.
- At the technology layer, we have created software components that transparently facade advanced middleware services and network components. Our approach permits us to select from a variety of middleware vendors in database systems, application servers and notification services.

Application Layer

At the application layer, the framework provides two extension hooks: development of new collaborative components and awareness actuators reacting to information events produced in the framework. We, however, study awareness actuators in the CSCW layer as part of the overall awareness model.

Our major aim is to facilitate development of collaborative components, and thus achieving a smooth transition from local to distributed applications. Obviously, our approach is to be based on existing specifications already adopted in the developer community.

Like many other groupware toolkits [7], we have decided to use the Java language and the JavaBean component specification as our preferred development technologies. In this line, our approach aims to sustain JavaBean components in a container that hides complexity in distributed programming and networking skills. The general idea is to provide a property-based persistence system, and to construct a Java distributed event service. Components are also packaged in JAR files, and they use XML to provide component descriptor.

Component Model

The component model comprises persistence, events, introspection, packaging, and component deployment in the ANTS component repository:

Properties are shared data structures stored in the ANTS server. Each JavaBean component can use three types of properties: string properties, string indexed properties, and Serializable object properties. These properties are persistently stored in a database by the remote property-based component and thus assure component persistence. We believe that string properties and indexed properties are a strong base for storing textual or XML strings containing persistent state. Yet, we supply serializable object properties enabling storing of more complex java objects.

Following the JavaBean bound properties approach, each property change will trigger an event (in this case, to the distributed event service) allowing a later subscription to PropertyChangeEvents. Completely coherent with the JavaBean approach, we also provide addPropertyChangeListener methods to receive property change events in bound properties. In our case, and transparently to developers, property change events are received from the distributed notification service and restricted to events in the current session and component.

Another key issue of the ANTS framework is to support coordination in a generic way allowing coordination rules for access control, concurrency control, and floor control. Our approach is also inspired in the JavaBean model, and particularly in the constrained properties.

Each method can trigger a PropertyVetoException like in the constrained property convention. We, however, differ in the source of the VetoException; in the JavaBean model, vetoes are thrown by registered VetoableChangeListeners; in our model, vetoes are thrown by registered coordination managers. Coordination managers interpose in calls to a component to assure validity of the requested action

Concerning customization and introspection of components in the ANTS framework, we provide an XML descriptor file. In general, a descriptor includes properties and event information. Event information will be the key for the awareness infrastructure in order to know which events are triggered by each resource in the shared environment.

Finally, component packaging uses standard JAR files like in the JavaBean component model. The package must contain the XML descriptor, component classes, and any local resource –images, files– used in the component.

Packaged components can be deployed in the ANTS server following an automated process. In this process, components are registered in the component repository using the XML descriptor, and unpacked resources are copied to the Web container.

The repository is an important element of the overall framework that permits to get information about registered components, and allows insertion and removal of artifacts in shared session contexts.

CSCW Layer

The CSCW layer comprises the set of collaborative services that the ANTS container offers to the application layer. We will, however, distinguish two main modules, namely container runtime (ANTS.CORE) and awareness services (ANTS.AWS), seamlessly interconnected by the collaboration bus.

ANTS.CORE comprises a generic and extensible application layer for collaborative tools, providing

services like shared sessions, coordination control, and integration for both synchronous and asynchronous applications. ANTS.AWS is a generic awareness service providing appropriate actuators for events received from the collaboration bus.

ANTS.CORE

ANTS.CORE is an essential module in the ANTS framework that includes explicit support for Sessions, shared artifacts, coordination control, security, and a seamless security model.

We have been strongly influenced by Multi-user Object Oriented (MOO) architectures because of their extensible and generic architecture. In a MOO system, "place" objects, which represent discrete locations interconnected by portal objects, define the topology of the space. These portal objects can establish any graph-directed relationship between "places". In a MOO architecture every static or dynamic entity is an object (Thing), every object has a set of properties and a set of verbs, and properties and verbs can be added or removed in runtime.

Like a MOO system, our shared session is represented by the Place object, interconnected by portal objects (Links). The shared artifacts are represented by JavaBean components dynamically loaded by the ANTS container. Object properties are represented in our model by the remote persistent property-based component, and object verbs are JavaBean classes.

The Place represents the shared session containing users, components, and links to other places. It provides methods in order to send or subscribe to events in the current context. It also supplies methods to get connected users and available links to other places. Furthermore, it permits dynamic load of JavaBean components to the shared context. Let us study the component's life cycle (Figure 2):

```
Place place = PlaceFactory.createPlace("test");
place.loadModels();
ArtifactModel bean = (ArtifactModel) place.getModel("aboutNote");
String desc = bean.getDescription();
place.leave();
```

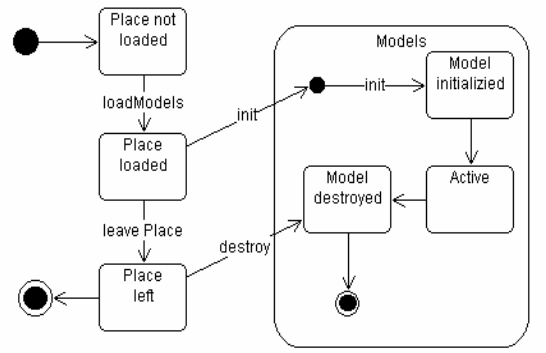


Figure 2. Component's life cycle.

When the *loadModels* method is invoked in a Place, all JavaBean components are activated (calling the *init* method) and classes are dynamically loaded by the ANTSClassLoader. The ANTSClassLoader retrieves classes from the public web directory using the HTTP protocol. Finally, when the user leaves the shared session, the *destroy* method is invoked in all JavaBean components currently active in the container.

ANTS.AWS

ANTS.AWS is a server-side awareness infrastructure enabling the triggering of a set of actuators in response to events produced in the collaboration stream.

ANTS.AWS is consistently integrated with the component model through XML descriptors included in packaged components. As such, every descriptor contains the events that this component triggers to the bus. Because this information is stored in the repository during the deployment phase, external monitoring application or AWS actuators can be aware of the information produced by every component in the platform.

This service creates the base for sophisticated monitoring applications, intelligent agents, and persistent actuators activated by events received from the Notification system. Based on existing CSCW Awareness systems [41], AWS includes an event repository, a notification service, and an infrastructure to filter or get information from events in the distributed bus.

AWS follows a Mediator design pattern with three important entities: Sensors, the Mediator, and actuators. Sensors represent any physical or software component producing events to the Notification system. Actuators perform special tasks on response to sensors; and the Mediator binds a sensor with one or more actuators, and it is responsible for launching actuators in response to events.

In relation to sensors, we consider two distinct types: event sensors and time sensors. Event sensors are activated in response to application events produced in the information bus. An event sensor is created using a subscription following the constraint filter language available in the Notification system.

On the other hand, time sensors are activated by the ANTS time scheduler; we can create a time sensor for a specified date or even set repetitive dates. The time scheduler notifies the awareness service when a time sensor has been activated.

In a possible scenario, we could create a subscription telling that we are interested in events produced by the user called "pedro" (event sensor), and that we want to make them persistent (actuator). This would log each event coming from user "pedro" occured in the environment. We could also create our own bots, which would react to events specified by our subscription.

Technology Layer

First of all, such a problem requires a solid infrastructure providing security, scalability, persistence, transactions and performance. As we explained before, many existing systems rely on non-distributed, single-threaded environments that limit their applicability to more complex problems.

For this reason, we have chosen the Java 2 Enterprise Edition (J2EE) standard as our preferred technology infrastructure. It saves us from implementing infrastructure and system-specific code and permits us to base on open specifications and components. This technology makes our system vendor-independent: we can choose any J2EE-compliant application server or relational database.

Concerning message oriented middleware, we needed a publish-subscribe notification service for our framework. Being JMS (Java Message Service) a standard alternative, we were also strongly interested in the new Elvin notification service. This service has a very good performance and has been used for CSCW purposes. As a result, we implemented a facade API that permits us to choose between any JMS-compliant messaging solution and Elvin.

In order to connect to a specific messaging middleware, developers only care about registered bus aliases in the framework. We provide several sample bus aliases to different middleware services. Depending on the selected middleware, we can choose different transport layers, such as, raw multicast, reliable multicast, encrypted Secure Socket Layer (SSL) streams, ordered streams, according to our specific requirements.

Non-expert programmers can simply select *SSL* as the communication channel (bus alias) and thus use an encrypted communication channel. Applications requiring scalability for a high number of users could then select the *RMCAST* alias.

MOVE ARCHITECTURE

In this section we will deal with MOVE's collaborative architecture. We will focus on the essential collaborative services provided, like session and zone management, component model and state propagation, coordination and consistency, and awareness.

Session and Zone Management

In order to determine the environment structure, the place concept is used. A place is a piece of virtual environment where users can interact with each other and use the different shared artifacts available. ANTS.CORE defines a session as a group of objects associated with some common communications pattern and supporting full-duplex multipoint communication among an arbitrary number of connected application entities. This is essentially what a place means in MOVE and any other MOO environments. By using this philosophy, we identify session with place.

In our case, the concept of "Places" is similar to the concept of "Locales" in MASSIVE-3 [34]: places are the fundamental unit of world composition, where each place represents a clearly separated region of the virtual world, such as a room or an open space, and can contain many virtual shared artifacts as well as avatars and portal objects.

As far as portal objects is concerned, we can say they are used for place linking purposes, thus providing a mechanism of transport between different places. By using this zone partitioning approach, we can define a session hierarchy where places can contain subplaces.

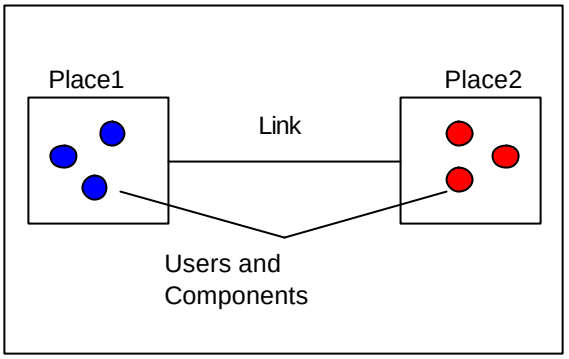


Figure 3. Zone Partitioning in MOVE.

Shared Artifacts and State Propagation

Another essential concept is the explicit support for **shared artifacts**. Normally, users engage in a collaborative interaction with other users in the shared context, or with accessible collaborative components. These collaborative components – usually called artifacts – range from synchronous to asynchronous, depending on the interaction time frame. Explicit support for creation and insertion of new artifacts represent a differentiating factor between extensible and non-extensible CSCW toolkits.

MOVE is based on a modular architecture for every component. There are different types of components available, such as voting, presenter, video, document or multimedia tools. All of these components follow the Model-View-Controller (MVC) paradigm. Every client owns a local copy of the controller, which is the tool used to modify the state of the component, and the view, which is the representation of the component that is shown on the screen.

Our main aim is to design the system so that it is easily extensible and reusable. To do so, every component has an XML descriptor that contains all the necessary properties and information related to events that can be thrown, besides the component's classes and any local resources. This new component is packaged using the standard JAR format.

Packaged components are deployed in the ANTS server following an automated process where these

components are registered in the component repository, using the XML descriptor, and unpacked resources are copied to the Web container. Using this mechanism makes integration of new components particularly easy because it is like "importing" the new component (Tool) into the system before it is ready to use.

One of the most difficult problems to solve when programming distributed systems is maintaining the state of the components among all other participants. In our case, this problem is minimized and transparent to us by using ANTS CSCW framework because it guarantees that the state will be propagated to every each user in the same place that we are in. This way, when developing new components for our system, this problem is solved and we can concentrate our development efforts on other aspects.

Another fact concerning state propagation is that persistence in our system is maintained, as the natural behaviour for every change in any property of the system is stored immediately to the database by means of propagating the event to the server. Other distributed virtual environments are not persistent in the sense that they are transient due to the fact that information is stored among all the clients. This may be an advantage sometimes but if all the clients go down at the same time the information may be lost.

We have developed a set of shared artifacts which include a voting tool, a presenter tool (both can be seen on Figure 4), a bars tool, a document tool, a jukebox tool and a video tool, as well as two tools for grabbing user's attention (the hook and camera tools). In these components, state propagation mechanisms provided by ANTS CSCW framework have been used.

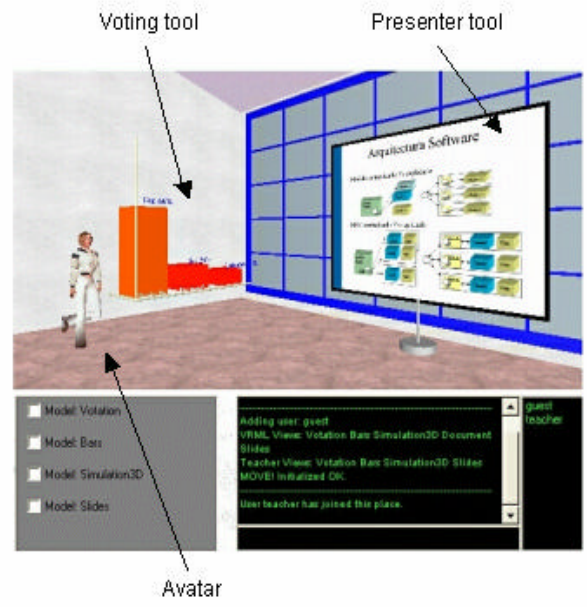


Figure 4. MOVE snapshot showing the presenter and voter artifacts and an avatar.

However, there are certain components that cannot use the state propagation mechanisms that the framework provides us with. One of the examples we have found is the avatar movement component. It is clear that it is impossible to transmit every positional event generated by the movement of a certain user to other users in the same place. This would generate a huge amount of event overhead over the network that even clients would be unable to deal with. In this case, we have had to optimize this component by implementing proximity event filters. As it is known, one of the weak points of real-time 3D environments is the enormous processing time required to render the 3D scene. This is why when there are lots of users in the same place the cost of rendering them is unacceptable for the client. To solve this problem we have implemented a proximity event filtering algorithm so as clients only receive the events of the avatars that are in a certain threshold distance near them, as seen in Figure 5.

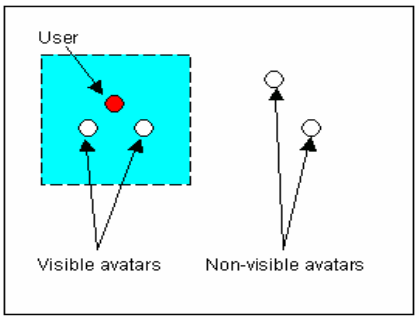


Figure 5. Only the avatars in user's threshold

Finally, note that a personal profile management configuration option has been added so that MOVE users can choose their own virtual representation in our environment (avatar), as well as many other properties that could be used in the future by an agent system.

Coordination and Consistency

Another key issue in any CSCW framework is the explicit software support for **coordination policies**. These policies can be set programmatically or declaratively, and can be categorized in roles for access control and concurrency control. The platform should provide extension hooks to insert custom coordination components that would then transparently interpose in normal calls to a collaborative component.

In our case, and considering that MOVE was initially designed as an educational collaborative virtual environment our access control policies have been established by means of defining basically two main roles: *teacher* and *student*. By using this approach, the teacher role members have total access to manipulating the state of the shared artifacts, whereas student role members can only view the result of these state changes made by teachers.

We are working in the development of a concurrency control mechanism by using a token-based lock component, to avoid more than one user manipulating any shared artifact. This mechanism will also allow users to transport artifacts from one place to another, thus assuring data consistency among sessions.

Awareness

Awareness is an important feature of collaborative environments. Awareness can be defined as an understanding of the activities of others, which provides a context of your own activity. From our perspective, an awareness platform should provide data acquisition from the running environment. In this line we emphasize the importance of a well-defined awareness model.

MOVE benefits from ANTS CSCW awareness and monitoring services. In fact, ANTS.AWS is the base for the MOVE agent infrastructure. In MOVE it is possible to program bots that perform special tasks and that react to events produced in the environment. As an example, we provide a guide bot that shows the whole place to new-comer users and interacts with the shared artifacts when a certain user moves besides it. More advanced agents can be developed and in fact we are using them pervasively to test our environment to simulate high user loads.

As seen in Figure 6, the bus is used for state propagation and for awareness purposes. On the one hand, each time the user moves it transmits its movements to the bus and they are received by the rest of the clients in the shared context (1 and 2). On the other hand, the monitoring service (*Mediator*) listens to the bus (3) and launches specified actuators in response to filtered events. In this case, the Mediator activates a guide bot that starts a sightseeing tour over the place.

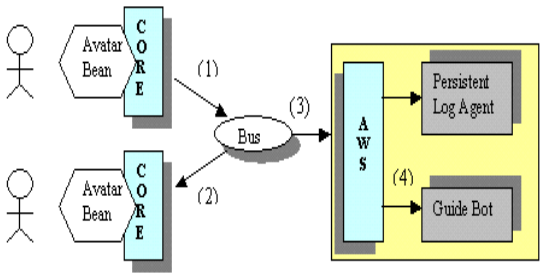


Figure 6. MOVE components and bots.

Right now we have simple bots or agents running, yet in the future these could be enhanced so as we could have advanced guiding bots, help bots or even bots that could collaborate and interact with each other by using agent communication protocols.

As all the events are recorded on the system, it would not be difficult to build up a Capture & Replay system to make bots for instance, to replay all the actions that a user has committed in a certain session. It is also possible to create log files of these events or even create subscription-specific log files for components or roles.

Data mining techniques could also be applied to extract relevant data from these log files.

CONCLUSIONS

MOVE is a Collaborative Virtual Environment constructed on top of a component groupware framework. The VE fully benefits from the underlying collaborative services provided by the ANTS infrastructure.

While the tendency for modularity is rippling through VE systems, we foresee interesting research in the development of component frameworks offering collaborative services in a seamless and extensible fashion to VE developers.

Our work aims to provide consistent groupware foundations to the development of VEs and thus enabling further reuse and interoperability in this complex domain.

One of the key aspects of this work is to provide extensibility at all levels of the framework. It is thus possible to create new components or artifacts, to develop new coordination mechanisms, to interface with different middleware services and propagation channels, and to develop new actuators for the awareness service. This creates an interesting Playfield for research in VEs by opening the system to third-party extensions willing to solve specific problems in this wide domain. Furthermore ANTS and MOVE are open source projects that welcome contributions and extensions by commercial, educational or research groups.

We are now developing new artifacts required by the educational communities in archaeological, medical and pedagogical applications involved in the Internet2 Catalonian project. The use of MOVE in a real setting will give us interesting feedback to create new awareness actuators and bots and to further improve interaction capabilities in the MOVE environment

As future works, we plan to further improve the coordination and consistency control mechanisms, to test different propagation channels and to apply data-analysis techniques to better understand collaboration flows in the running environment.

We also foresee interesting research in the awareness extensions to the MOVE Virtual environment. The provision of advanced awareness indicators and the extension of the agent system can lead to new interesting scenarios in VEs.

ACKNOWLEDGEMENTS

This project has been partially funded by the 'Generalitat de Catalunya' Internet2 Advanced Telecommunications project.

REFERENCES

[1] Broll, W. Populating the Internet: Supporting Multiple Users and Shared Applications with VRML. ACM SIGGRAPH:

Proceedings of the VRML'97 Symposium. New York, 1997, 87-94.

[2] DIVE Homepage. http://www.sics.se/dive.

[3] Garcia P., Skarmeta A., Rallo R., ANTS: A new Collaborative Learning Framework, European Conference on Computer Supported Collaborative Learning. 2001.

[4] Greenhalgh, C., Purbrick, J and Snowdon, D. Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring. Proceedings of the CVE 2000 Symposium, ACM, 2000, 119-127

[5] H-Anim Humanoid Working Group. http://www.h-anim.org

[6] Hofte, G. Henri ter. Working Apart Together: Foundations for Component Groupware. Telematica Instituut, Enschede, The Netherlands, 1997, 288.

[7] Marsic, I. DISCIPLE: A Framework for Multimodal Collaboration in Heterogeneous Environments. ACM Comp. Surveys, vol 31, no. 2es, 1999.

[8] Mitsubishi Electric Research Laboratories. Scalable Platform for Large Interactive Networked Environments (SPLINE). http://www.merl.com/projects/spline/.

[9] NPSNET Research Group. http://www.movesinstitute.org/~npsnet/index.html.

[10] Oliveira, M., Crowcroft, J. and Slater, M. Component Framework Infrastructure for Virtual Environments. Proceedings of the CVE 2000 Symposium. ACM, 2000, 139-146.

[11] Prinz, W. NESSIE: An Awareness Environment for Cooperative Settings. Proceedings of 6th European Conference on Computer Supported Cooperative Work. Kluwer Academic Publishers, 1999, 391-410.

[12] Roseman, M. and Greenberg, S. Building Real-time Groupware with Groupkit, a Groupware Toolkit. ACM Transactions on Computer-Human Interaction, 3(1):66—106, March 1996.

[13] Tower Project. http://tower.gmd.de.

[14] U.S. Department of Defense (DOD). Distributed Interactive Simulation, IEEE Standard 1278. 1993