Plataforma per a la simulació de xarxes de sensors

Memòria del projecte
d'Enginyeria en Informàtica
realitzat per
Marc Pujol González
i dirigit per
Meritxell Vinyals Salgado i
Josep Puyol Gruart
Bellaterra, 4 de febrer de 2008

El sotasignat, *Josep Puyol i Gruart*

Professor de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

Marc Pujol i González

I per tal que consti firma la present.

Josep Puyol i Gruart
Bellaterra, 4 de febrer de 2008

A la meva mare, del nen que fa temps va deixar de ser-ho.

Agraïments

Aquest projecte no hauria estat possible sense l'esforç, constància, recolzament, paciència i comprensió de la Meritxell, que ha cregut en mi quan ni jo mateix ho feia. Sense cap mena de dubte ha estat la millor directora que podria haver desitjat.

La més sincera gratitud al Bruno, un company inestimable de sol·lucions inesgotables i la més polida professionalitat a qui sens dubte trobaré a faltar en qualsevol projecte d'ara endavant.

Haig d'agraïr al Jar per els ànims i la serenitat que m'ha transmès en els moments difícils, l'experiència amb què ha guiat el projecte i la cordialitat amb que m'ha tractat des del primer dia.

El repte no hauria estat el mateix sense les idees i l'energia del Jesús. Reinventaria el món al seu costat (amb zebres, per suposat!).

Gràcies també a l'Institut d'Investigació en Intel·ligència Artifical, que m'ha proporcionat els recursos necessaris i un bon espai de treball. El millor que me n'emporto és la gent que hi he conegut.

Finalment, gràcies a totes aquelles persones que m'han fet costat durant el projecte i tota la carrera.

Índex

Índex de figures

Índex de taules

1 Introducció

Les xarxes de sensors són xarxes formades per dispositius amb diferents capacitats de còmput i percepció que estan distribuïts en una àrea i collaboren per produir una informació amb significat global a partir de les informacions sense processar obtingudes a nivell individual.

L'interès de la comunitat científica per a les xarxes de sensors com a domini d'aplicació i com a font de reptes ha crescut enormement en la última dècada a causa principalment de tres motius: (1) la presència en el mercat de sensors cada cop de menor tamany i amb un preu més reduït que n'han permès el seu desenvolupament (2) el conjunt d'avantatges que presenten en front d'altres tecnologies per a la monitorització i (3) el gran nombre d'aplicacions que, tot i ser una tecnologia emergent, ja s'han detectat per aquestes xarxes.

L'estudi de xarxes de sensors a partir de desplegaments reals resulta inviable per dos motius: (1) el desplegament complert d'una xarxa suposa una elevada despesa econòmica i (2) la inevitable variabilitat dels resultats obtinguts, que depenen de factors incontrolables, com per exemple, la meteorologia. Una alternativa molt més factible per a la realització d'aquests estudis consisteix en la utilització de simulacions per ordinador. La simulació permet el desenvolupament de nous algorismes sense els costos de desplegament i ofereix un entorn de treball controlat que facilita la rèplica, anàlisi i validació dels experiments. Aquest projecte es centra en proporcionar una plataforma de simulació per a xarxes de sensors des de la perspectiva dels sistemes multi-agents. La plataforma ha de facilitar la programació de diferents aplicacions concretes de xarxes de sensors i sorgeix davant de la necessitat d'una eina d'aquest tipus per a portar a terme experiments i demostracions al grup de recerca del projecte IEA (Institucions Electròniques Autònomes, TIN2006-15662-C02-0) del IIIA-CSIC.

En la secció 2 s'introdueixen els fonaments teòrics de les xarxes de sensors i la simulació per computador. Els requeriments que haurà de complir la plataforma presentada s'exposen a l'apartat 3. La secció 4 presenta un estudi de l'estat de l'art dels simuladors per a xarxes de sensors i defensa la necessitat d'una nova eina en aquest camp. A continuació es presenta una planificació temporal del treball (secció 5), el llenguatge de programació escollit i les eines utilitzades (6). Seguidament s'exposa l'arquitectura, disseny i implementació de la plataforma en el capítol 7 i dues simulacions d'exemple implementades (secció 8). Finalment, l'apartat 9 exposa les conclusions del projecte, que repassen els objectius assolits i presenten possibles línies de continuació.

2 Fonaments teòrics

Aquesta secció recull els fonaments teòrics necessaris per a la comprensió del problema. En l'apartat 2.1, amb [1] com a referència, s'introdueixen les xarxes de sensors, les seves característiques i els dominis d'aplicació per defensar la visió d'aquestes xarxes com a sistemes multi-agent. Tot seguit, en l'apartat 2.2.1 es realitza una breu introducció a la simulació per computador, les motivacions que n'impulsen la utilització i les diferents aproximacions que existeixen actualment.

2.1 Les xarxes de sensors

A diferència d'altres tecnologies per a la monitorització que utilitzen un o un nombre reduït de sensors amb altes capacitats, complexes i de preus elevats, aquestes xarxes emergents estan formades per gran nombre de sensors de baix cost, baix consum i capacitats més limitades. El primer avantatge enfront d'aquestes altres tecnologies és la capacitat (degut al gran nombre de sensors) de monitoritzar grans àrees mentre que les tecnologies tradicionals solen estar limitades a petits o mitjans entorns. Un segon avantatge és que degut a la seva estructura distribuïda aquestes xarxes es poden utilitzar per monitoritzar entorns remots i hostils, presentant robustesa a caigudes dels nodes. Finalment les xarxes de sensors són considerades una tecnologia no invasiva el que les fa idònies per certs dominis com la monitorització de l'hàbitat, la seguretat o la creació d'entorns intel·ligents.

Conseqüentment, les xarxes de sensors han estat identificades com una de les tecnologies més prometedores en un futur immediat [2][3]. A més, les xarxes de sensors formulen una sèrie de reptes per la comunitat científica que obren un conjunt de noves àrees d'interès tant en maquinari com en programari.

Les xarxes de sensors són un domini d'aplicació interessant que planteja nous reptes per a la recerca degut a que es caracteritzen com: *complexes* (no hi ha una manera simple de dissenyar manualment una xarxa de sensors que s'adapti i reaccioni a les condicions canviants de l'entorn), *inherentment distribuïdes* (els dispositius estan distribuïts en una àrea i amb un alt grau d'activitat local), *interconnectades* (les accions dels sensors presenten dependències entre aquests i han d'interactuar a l'hora de pendre decisions), *d'adaptació i reacció ràpida* (en general estan col·locades en entorns de condicions ràpidament canviats que els imposen importants restriccions de temps en la seva adaptació) i de *gran escala* (les xarxes estan formades per centenars, milers de nodes).

A continuació en la secció 2.1.1 es dóna una visió de la gran diversitat de tipus de xarxes de sensors existents segons les principals característiques dels seus components. Tot seguit, en la secció 2.1.2 s'enumeren els principals dominis d'aplicació identificats per aquestes xarxes i els reptes que plantegen

a la comunitat científica (secció 2.1.3). Finalment, en la secció 2.1.4, es defensa l'aplicació de les tècniques i algoritmes per a sistemes multiagents a l'hora de modelar i afrontar els problemes que plantegen aquestes xarxes.

2.1.1 Tipus i característiques de les xarxes de sensors

Encara que totes les xarxes de sensors presenten unes característiques comunes, els problemes i reptes que planteja cada xarxa poden variar substancialment depenent de les característiques dels sensors, de la xarxa i de l'entorn. Pel que fa a les propietats dels sensors s'ha de considerar:

- *La font d'alimentació:* La font d'alimentació és una característica molt important dels sensors que restringeix tota la xarxa. Típicament els sensors no estan connectats a la corrent i tenen una alimentació per bateria. Els canvis d'aquestes bateries en moltes aplicacions no són possibles degut al tipus de desenvolupament de la xarxa d'una manera ad-hoc i la seva situació en llocs de poc accés i grans àrees. Per tant el fet de que els sensors disposin d'una bateria limitada limita la vida de la xarxa.

- *La mobilitat:* El fet que els nodes tinguin mobilitat és un factor a considerar. Per una banda, la posició s'afegirà en la configuració del sensor per a la seva optimització. Per l'altra, un canvi de posició introdueix un canvi a tota una sèrie de punts a considerar: el conjunt de veïns, el cost de la comunicació i altres.

Pel que fa a les propietats de la xarxa destaquem quatre propietats:

- *Tipus de nodes:* Una xarxa pot estar composada per a sensors homogenis (pel que fa al tipus de fenòmens que observen, a les seves capacitats i possibles configuracions) o per a sensor heterogenis.

- *Desenvolupament (fixa /ad-hoc):* Pel que fa al tipus de xarxa típicament les xarxes de sensors són xarxes *ad-hoc* on a diferència de les xarxes fixes els sensors s'afegeixen/abandonen la xarxa d'una manera no controlada.

- *Comunicació:* El tipus de comunicació que utilitzen els nodes de la xarxa és un factor molt important alhora de considerar-ne restriccions. Típicament les xarxes de sensors estan formades per a nodes sense fils amb poc ample de banda i una comunicació inestable el que fa que estiguin caracteritzades per fortes restriccions en aquest sentit.

- *La propietat:* Si la xarxa està tota formada per a nodes de la nostra propietat tots els nodes estaran sota el nostre control mentre que si hi ha múltiples propietaris s'haurà de negociar i implementar estratègies per obtenir-ne les observacions i les configuracions desitjades.

Finalment, pel que fa a l'entorn que observa i on es situa la xarxa en destacarem *el dinamisme*, doncs depenent del dinamisme de l'entorn (velocitat en que es produeixen els canvis i evoluciona l'entorn) la xarxa i els seus sensors estaran subjectes a unes restriccions temporals més o menys fortes per actuar.

2.1.2 Aplicacions de les xarxes de sensors

Un dels aspectes més motivant de les xarxes de sensors és el seu gran nombre de dominis d'aplicació. Des que varen aparèixer les primeres xarxes, els científics han identificat un gran nombre d'aplicacions possibles i desplegat xarxes reals aplicades a dominis específics. Aquí es detallen els dominis considerats més rellevants per aquest tipus de xarxes:

- *Detecció de perills.* Fa referència a l'aplicació de les xarxes de sensors a la detecció d'atacs (biològics, químics, nuclears, etc.) o perills potencials (erupció de volcans, detecció de foc al bosc, etc.). En aquest context es consideren les xarxes de sensors perquè ofereixen una resposta ràpida i permeten la cobertura de grans àrees.

- *Monitorització de l'hàbitat i l'entorn.* Fa referència a l'aplicació de les xarxes de sensors en la monitorització de l'hàbitat d'animals i/o vegetació. Es refereix també a la determinació de paràmetres ambientals per a la predicció de canvis en ecosistemes o meteorològics.

- *Aplicacions biomèdiques.* Consisteix en introduir xarxes de sensors dins del sistema humà per monitoritzar estats de salut i controlar/-detectar certes malalties. En aquesta direcció ja s'han desenvolupat monitors del nivell de glucosa i pròtesis de retina.

- *Monitorització/Vigilància del trànsit.* La monitorització del tràfic es basa en distribuir sensors per detectar, comptar i estimar les velocitats dels objectes mòbils (cotxes, avions, vaixells) en una zona/xarxa determinada. Aquesta xarxa de sensors pot utilitzar la informació individual per generar informació més elaborada tal com detectar embussos o altres patrons de tràfic. Una segona aplicació de les xarxes de sensors al tràfic és la vigilància i el seguiment d'un vehicle específic a través de múltiples càmeres.

- *Creació d'espais intel·ligents (Smart Spaces).* Les xarxes de sensors permeten el desenvolupament d'entorns i edificis intel·ligents amb capacitats de percepció i còmput que ofereixen una millor interacció amb humans monitoritzant-ne les seves activitats i personalitzant-ne l'ambient segons les seves preferències. Aplicacions populars en aquest domini són els sistemes *HVAC* (calefacció, ventilació i aire condicionat) que controlen la temperatura , sistemes d'il·luminació; i sistemes per a monitoritzar persones de la tercera edat.

- *Robòtica distribuïda.* Els sensors que tenen mobilitat o fins i tot actuadors, a més a més dels elements de percepció, poden ser considerats com a micro-robots. En aquest context, les xarxes de sensors s'han aplicat a problemes típics de robòtica referents als equips de robots: creació de mapes de l'entorn col·laborativament o equips de robots que treballen per localitzar supervivents en una àrea catastròfica.

- *Materia intel·ligent.* S'anomena matèria intel·ligent a aquells materials que modifiquen les seves propietats adaptant-se a canvis de l'entorn. Per fer-ho aquests materials disposen d'una xarxa de sensors que els permet monitoritzar i adaptar-se a aquests canvis. Un exemple força popular és la monitorització d'estructures de la construcció tals com edificis, ponts i altres.

2.1.3 Reptes que plantegen les xarxes de sensors a la comunitat científica

Entre els principals reptes i noves àrees d'interès que les xarxes de sensors plantegen a la comunitat científica destacarem les següents:

- **Tècniques de localització.** Degut al tipus de desenvolupament *ad-hoc* de les xarxes de sensors en general els seus nodes no tenen coneixement de la seva pròpia posició. Així doncs les tècniques de localització per estimar les coordenades dels nodes és una línia oberta de recerca.

- **Algoritmes d'encaminament i propagació de dades.** Els algoritmes d'encaminament a les xarxes de sensors han de donar sol·lucions per un encaminament eficient en xarxes típicament caracteritzades com a *wireless*, *ad-hoc* i amb fortes restriccions d'energia.

- **Algoritmes per a la fusió/agregació i el processament descentralitzat de la informació.** En moltes aplicacions la informació captada individualment pels sensors s'ha de processar i fusionar amb informacions d'altres sensors. Això pot ser degut a que els sensors necessitin observacions d'altres sensors per actuar de forma òptima o a que la informació s'ha d'agregar i abstreure per a reportar només la informació més rellevant (degut a la gran quantitat de dades generades i a les limitacions d'energia i ample de banda de la xarxa).

- **Tècniques de comportament reactiu/adaptatiu i estrategies comunes de percepció.** En aquest punt fa referència a les tècniques que proporcionen als sensors una percepció activa. S'entén com a percepció activa com a la capacitat d'una xarxa de reconfigurar i coordinar els seus sensors per a maximitzar la informació percebuda. En moltes aplicacions aquests sensors han de realitzar una percepció conjunta i alhora de reconfigurar-se d'una manera òptima no només han

de tindre en compte les seves accions sinó també coordinar-se amb les accions dels seus veïns.

- **Tècniques de negociació i cooperació per a xarxes composades de sensors amb diversos objectius.** Quan els sensors d'una xarxa tenen diversos propietaris aquests també poden tindre diversos objectius. Alhora de modelar aquests entorns no cooperatius es fa necessari l'ús de tècniques de negociació i mecanismes de mercat.

2.1.4 Xarxes de sensors i sistemes multi-agent

Un *sistema multi-agent* es pot definir com un sistema computacional on un conjunt d'agents interactuen (competint o cooperant els uns amb els altres) per aconseguir realitzar una tasca individual i/o col·lectiva. Aquests agents poden cooperar (*sistemes cooperatius*) o competir pels recursos (*sistemes no-cooperatius*). Es classifiquen també en sistemes *tancats* (sempre hi ha els mateixos agents en el sistema) i *oberts* (la població d'agents canvia amb el temps). Finalment el sistema pot ser o no *distribuït* el que comporta que els diferents agents estan distribuïts en diferents nodes d'una xarxa.

Els sistemes multi-agents s'estan aplicant amb èxit a les xarxes de sensors degut a la seva capacitat per a modelar d'una manera natural un conjunt de sensors autònoms i físicament distribuïts i les seves interaccions [4].

Aquests sistemes són de tipus distribuït i, en general, cada sensor és modelat com un agent (encara que hi ha aproximacions en què un agent controla un conjunt de sensors). En la majoria de casos són modelats com a sistemes oberts degut a la naturalesa *ad-hoc* d'aquestes xarxes. Aquests sistemes tan poden ser dissenyats com a cooperatius o com a no-cooperatius depenent de la propietat dels sensors i l'enfoc del problema per part del dissenyador.

En la literatura, veiem que els sistemes multi-agents han contribuït en el desenvolupament i la investigació en xarxes de sensors amb noves metodologies de les quals destaquen: *la teoria computacional d'organitzacions*, *el disseny de mecanismes de mercat*, *algorismes de cooperació i coordinació* i *algorismes d'aprenentatge i distribució de la informació distribuïts.*

2.2 Simulació per computador

La simulació per computador representa un mètode econòmic d'estudiar tot tipus de sistemes. Els simuladors d'avui en dia han de resultar eficients i senzills d'utilitzar i disposar de la capacitat de tractar amb la creixent complexitat dels sistemes modelats. A més, és necessari un alt nivell d'extensibilitat per poder-se utilitzar en l'anàlisi del comportament d'un ampli ventall de sistemes.

La primera classificació a realitzar entre les diferents aproximacions a la simulació per computador es basa en distingir la naturalesa dels sistemes

que permeten modelar, separats en **continus** i **discrets**.

2.2.1 Simuladors continus

Els simuladors continus es caracteritzen per a la utilització de fórmules matemàtiques que descriuen com els components de la simulació responen a diferents condicions. Per poder utilitzar un simulador d'aquestes característiques és necessari que el comportament dels components sigui conegut i expressable en equacions. El simulador aplicarà les fórmules segons les condicions definides per l'entorn i la connectivitat entre els components. S'obté així una sortida contínua que descriu acuradament com respondria el sistema modelat a les condicions exposades. Aquesta sortida reflexa normalment canvis de l'estat del sistema en referència al temps, tot i que pot demostrar també altres tipus de relacions. Desafortunadament, les equacions utilitzades per a modelar els sistemes de forma contínua són complexes, amb un cost computacional que s'incrementa exponencialment en relació al nombre d'elements que intervenen. Aquest cost s'acaba reflectint en lentitud i, per tant, aquests simuladors s'utilitzen només per a sistemes amb un nombre relativament petit de components descrits a molt baix nivell.

2.2.2 Simuladors discrets

En els sistemes discrets l'estat dels components canvia únicament en instants de temps determinats, responent a un succés o event. La simulació d'aquests sistemes consistirà doncs en identificar quins són aquests events i descriure els processos que desencadenen. Al contrari que per als sistemes continus, la descripció d'aquests processos no té perquè ser matemàtica: consistirà d'un conjunt de canvis en l'estat dels elements del sistema i/o el desencadenament de nous events. Existeixen dues aproximacions diferents per a la simulació per computador dels sistemes discrets: *event scheduling* i *activity scanning*.

L'*event scheduling* és el primer mètode que es va desenvolupar. En aquest mètode un event és qualsevol cosa que canvia l'estat del sistema, exceptuant-ne el pas del temps. Els events s'emmagatzemen en una cua que els manté ordenats segons el temps en què succeeixen. La idea bàsica és avançar el temps fins que succeeix un event i desencadenar-ne el procés corresponent. A partir d'aquí l'execució d'una simulació del sistema és trivial, seguint el següent algorisme:

1. Es configura el rellotge de la simulació a temps 0 i s'afegeixen els events inicials a la cua.
2. S'extreu el primer event de la cua i s'executa el procés associat.
3. S'afegeixen tots els nous events generats a la cua, mantenint l'ordre.
4. Mentre no s'acabi la simulació, tornar al pas 2.

L'*activity scanning* és una aproximació força similar, però amb una diferència significativa. Introdueix un nou tipus d'events que no succeeixen en un temps determinat sinó al complir-se certes premisses. Aquests events s'anomenen activitats o events condicionals. El simulador haurà de comprovar contínuament si les condicions d'aquestes activitats es compleixen per desencadenar-ne els processos associats, seguint el següent algorisme:

1. Es configura el rellotge de la simulació a temps 0 i s'afegeixen els events incondicionals inicials a la cua.

2. S'extreu el primer event incondicional de la cua i s'executa el procés associat.

3. S'afegeixen tots els nous events incondicionals generats a la cua, mantenint l'ordre.

4. Es comproven les premisses de tots els events condicionals, executant els processos associats dels que les satisfan.

5. Mentre no s'acabi la simulació, tornar al pas 2.

La sobrecàrrega introduïda per els events condicionals representa el major desavantatge d'aquest mètode, mentre que amb ell resulta molt més senzill pensar i formalitzar els sistemes. Actualment els inconvenients d'utilitzar aquest mètode sobrepassen els beneficis però no es descarta en certes àrees com la simulació de sistemes basats en normes.

3 Requeriments

La simulació per computador és l'opció més viable per estudiar les xarxes de sensors perquè permet el desenvolupament dels algorismes sense els costos de deplegament i ofereix un entorn de treball controlat que facilita la rèpica, anàlisi i validació dels experiments. No obstant, aquesta aproximació no està exempta de reptes als que haurem d'aportar una sol·lució. A continuació es presenten els punts de major relevància a considerar durant el disseny i la implementació del simulador.

3.1 Codi lliure

La plataforma ha de ser de domini públic perque pugui ser aprofitada per tota la comunitat científica. És necessari doncs que no depengui de cap programari propietari ni per al seu desenvolupament (llenguatge de programació, sistema operatiu) ni per a la seva execució (llibreries utilitzades).

3.2 Abstracció

Ha de ser un entorn de simulació de propòsit general, disposant del suficient nivell d'abstracció com per a permetre instanciar xarxes de sensors de diferents dominis i aplicacions. Per aconseguir-ho, haurà d'aportar una infraestructura que interrelacioni els diferents elements bàsics de qualsevol xarxa de sensors (exposats en la secció 2.1):

- Fenòmens observables
- Sensors
- Agents
- Mecanismes de comunicació

Aquests elements seran presents en pràcticament qualsevol xarxa de sensors, compartint certes caracterítiques generalistes que podem abstreure i modelar perquè no hagi de fer-ho l'usuari.

3.3 Extensibilitat

Igual que existeixen característiques comunes entre els diferents components d'un mateix tipus, també hi trobarem grans diferències degudes a l'alt grau d'especialització de que poden arribar a disposar.

El simulador que es presenti ha de facilitar en la mesura del possible la reutilització i extensió d'aquests components per a diferents simulacions, permetent així l'estudi dels algorismes sobre diferents aplicacions o dominis sense exigir un elevat cost de reconfiguració i/o reprogramació.

Seria també desitjable que aportés un sistema de classificació i compartició pública dels models creats per evitar la reimplementació de les mateixes funcionalitats per part de diversos usuaris, estalviant-los temps i millorant la qualitat dels models resultants.

3.4 Escalabilitat

El major potencial de les xarxes de sensors deriva directament de la cooperació entre gran quantitat de sensors, de manera que un requisit bàsic i indispensable per al simulador és la capacitat de gestionar-los.

Com que la memòria és un recurs finit i s'haurà de tractar amb gran nombre d'elements que durant el procés de simulació restaran emmagatzemats en memòria, es fa evident la necessitat de limitar al màxim l'espai utilitzat per a representar cadascun d'aquests components.

3.5 Eficiència

La capacitat de procés també és un recurs limitat del qual s'ha d'intentar treure el màxim rendiment possible. No obstant, cal notar que en el simulador es duran a terme dos tipus diferents de processos:

1. **Procés dels agents**
   Representa el temps de còmput utilitzat per els agents, és a dir, el còmput que realitzarien els diferents nodes de la xarxa si en realitzessim un desplegament físic. Lògicament aquest temps de procés només depèn dels algorismes escollits per l'usuari, no del marc de treball que s'ofereix, i per tant serà impossible reduïr-lo.

2. **Procés del marc de treball**
   Representa el cost computacional derivat de la infraestructura que interrelaciona els components i simula els diferents processos: comunicacions, consumició d'energia, mecanismes de control, interfície d'usuari i altres.

Podem definir com a eficiència de procés del simulador E, la interrelació entre el temps de procés dels agents P_A i el temps de procés total P_T:

$$E = P_A/P_T$$

Evidentment, el simulador haurà d'intentar maximitzar aquesta eficiència, però s'ha de tenir en compte que depèn del temps de procés dels agents P_A. Com que aquest temps varia en funció dels algorismes escollits per l'usuari la mesura d'eficiència descrita anteriorment és només orientativa i s'ha d'observar com a tal.

3.6 Repetibilitat

Els experiments realitzats amb el simulador només seran vàlids per la comunitat científica si altres investigadors els poden repetir. És imprescindible doncs que dues execucions de la mateixa simulació i amb els mateixos valors inicials produeixin exactament el mateix resultat.

3.7 Control temporal

Donat l'entorn dinàmic sobre el que treballen les xarxes de sensors, l'objectiu dels algorismes a simular no és tant sols assolir una configuració òptima dels sensors o la millor fusió d'informació sinó també reaccionar a una velocitat adequada al dinamisme de l'entorn. Per això un dels requisits del simulador serà proporcionar eines per a modelar amb exactitud la durada de les accions dels components i així les seves interaccions amb l'entorn.

3.8 Observació

L'execució d'una simulació genera una sèrie d'informació que s'haurà de recollir i presentar a l'usuari. Aquesta informació pot ser de dos tipus:

1. **General**
   Informació aplicable a qualsevol domini, que fa referència al model general de les xarxes de sensors, inclou:
   - Missatges enviats i rebuts
   - Temps de CPU utilitzat per els agents
   - Posició dels diferents elements
   - Energia consumida
2. **Específica del domini:**
   Serà tota aquella informació que sigui útil per a la comprensió i avaluació del sistema i depengui del domini que s'estigui simulant:
   - Estat del món, fenòmens i agents
   - Mesures captades per els sensors
   - Contingut dels missatges enviats i rebuts
   - Mesura d'error entre la realitat i la percepció oferta per la xarxa

El repte en aquesta àrea consisteix en integrar al simulador un conjunt d'eines que realitzin la recollida de la *informació general* de manera automàtica i permetin combinar-la amb la *informació específica*, així com un sistema de visualització i/o generació d'informes a partir d'aquestes.

4 Estat de l'art dels simuladors per a xarxes de sensors

Els primers entorns de simulació de xarxes de sensors són en realitat adaptacions de simuladors més generalistes, entre els quals podem destacar el *ns-2*[5] i el *J-Sim*[6], que exposarem en les següents seccions.

Posteriorment apareixen un seguit de simuladors nous, dissenyats des d'un principi pensant en la simulació de xarxes de sensors. De tots els estudiats, presentarem els dos que més s'aproximen als requeriments que impulsen aquest projecte: *Radsim*[7] i *SENSE*[8].

4.1 The Network Simulator - ns2

Ns2 és probablement el simulador més utilitzat en l'àrea de xarxes de computadors (té programats un gran nombre de protocols d'enrutament) i s'ha extès amb algunes funcionalitats bàsiques per a la simulació de xarxes de sensors (model d'energia i canal sensorial) incloses en la extensió *Sensor Sim*.

Proporciona un motor de simulació basat en events discrets implementat amb una arquitectura orientada a objectes, que li dóna certa extensibilitat. Per contra, presenta un alt grau d'acoblament entre els diferents mòduls, dificultant l'addició de mòduls nous o modificació dels existents fins al punt que només aquells que disposen d'un coneixement profund del seu funcionament poden fer-ho.

El nucli del simulador està implementat en *C++*, però la definició de les simulacions i la recollida de dades es realitzen en *OTcl*. Trobem doncs un altre inconvenient, ja que per realitzar simulacions de xarxes de sensors és necessari tant aportar nous mòduls com definir les simulacions, l'usuari estaria obligat a conèixer els dos llenguatges de programació.

Un altre problema que presenta és l'escalabilitat: cada node és un objecte i poden interaccionar tots entre ells, de manera que hi trobem algorismes amb complexitat $O(n^2)$. Per a xarxes amb un nombre relativament reduït això no és cap problema, però un simulador per a xarxes de sensors hauria de poder gestionar milers de nodes.

Com a últim motiu i més determinant pel que fa als objectius d'aquest projecte, cal notar que aquest simulador no disposa de capa d'aplicació i per tant ens seria impossible implementar-hi algorismes de fusió de dades i aprenentatge.

4.2 OMNeT++

OMNeT++ [9] és un simulador per a xarxes de sensors basat en components i de simulació basada en events que utilitza mòduls connectats en una jerarquia aniuada. Els mòduls bàsics de OMNet++ estan escrits en C++ mentre

que les estructures més complexes utilitzen un llenguatge propi, NED, com a llenguatge d'alt nivell. OMNet++ incorpora un model de xarxes de sensors, el *SenSim*. Aquesta extensió crea mòduls per als diferents components d'una xarxa de sensors: (1) sensors que són un mòdul composat d'altres mòduls (de capes de protocols, de hardware i de coordinadors de la comunicació) (2) objectes físics (3) canals sensorials i (4) canals de comunicació wireless.

OMNet++ ha demostrat ser significativament més ràpid que *ns-2*, oferint capa d'aplicació, modelització dels fenòmens físics, i més facilitat en la modificació dels components.

Tot i així, no és tan popular com *ns-2* i alhora de provar algoritmes d'encaminament els usuaris segueixen preferint el primer degut el gran nombre que en té implementats que en facilita les comparatives.

4.3 J-Sim i el seu Wireless Package

J-Sim és una llibreria, escrita totalment en Java, per a la simulació de processos discrets. Pensada principalment per a l'aplicació en xarxes de computadors, pot aplicar-se també a qualsevol sistema on els estats dels seus components canvien a temps discrets.

Amb l'experiència prèvia sobre la dificultat d'extensió i els problemes d'escalabilitat del *ns-2*, els creadors de *J-Sim* aposten per un disseny basat en components, on cadascun dels elements que intervenen en la simulació es modela com a un component. Cadascun d'aquests components és un sistema que ofereix una certa funcionalitat i es pot comunicar amb altres components, havent de complir els següents requisits:

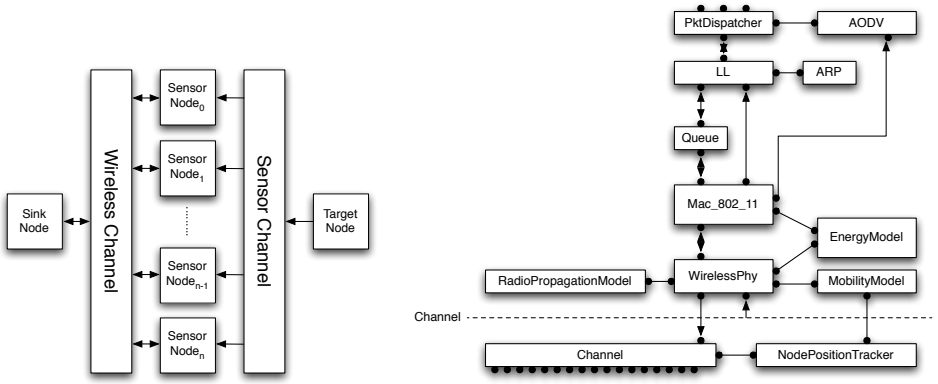
- Reutilitzable
- No depèn del context
- Combinable amb altres components
- Encapsulat (els altres components no poden inspeccionar el seu estat intern)

Wireless Package

Com l'*ns-2*, *J-Sim* inclou la implementació d'algorismes típics de xarxes de sensors tals com algorismes de localització, d'encaminament geogràfic [1] i l'algorisme de *difusió direccional* [10].

El *J-Sim Wireless Package*[11] és una extensió per al *J-Sim*, consistent en un conjunt de components creats específicament per a la simulació de xarxes de sensors. En la 1(a) es mostra l'esquema general del model utilitzat per representar i operar les xarxes de sensors, que inclou els següents

[1] algoritmes que realitzen l'encaminament no a partir d'una direcció de xarxa sinó a partir de la localització geogràfica dels nodes



(a) Model general (b) Model de sensor

Figura 1: Esquemes dels models utilitzats per *J-Sim Wireless Package*.

components: (1) *objectes observables* objectes que generen canvis en el món i que estan composats per tres mòduls: component de mobilitat i localització, el generador d'estímuls i el de comunicació amb la capa sensorial; (2) *sensors* per a modelar els diferents sensors que s'implementen a la xarxa composats pels següents components: CPU, radio, bateria, agent i mòduls encarregats de la interacció amb el canal sensorial; (3)*canal sensorial o físic* encarregat de transmetre els estímuls i composat del mòdul de localització de sensors i mòdul de propagació dels estímuls; i (4)*canal wireless* que emula una xarxa wireless utilitzant models de propagació de radio, models de targeta de xarxa i models de receptors. En la 1(b) trobem un esquema general dels tipus de components aportats per el *wireless package* i com aquests interaccionen entre ells.

Aquesta extensió ha estat dissenyada per provar algoritmes d'encaminament, disseminació de la informació i, en general, algoritmes destinats al traspàs d'informació en una xarxa del tipus wireless i ad-hoc amb restriccions de comunicació i energia com acostuma a succeir en una xarxa de sensors.

Com a conseqüència, el simulador intenta modelar d'una manera realista els components hardware de la xarxa realitzant un esforç per a modelar amb detall les propietats dels canals wireless, l'atenuació dels senyals en diferents models de terreny i els components dels sensors tals com la radio, la CPU o el consum de bateria. En contrast, encara que el simulador enfoca el problema des d'una perspectiva multi-agent, en la introducció del simulador es pressuposa que els nodes sensors envien sempre la informació a nodes d'emmagatzemament en detectar un estímul o sota demanda i no es comenta ni el tema de reconfiguració de sensors ni el de processament d'informació distribuïda.

4.4 Radsim

Radsim (*Radar Simulator*) [7] és un entorn de simulació basat en events discrets desenvolupat per a provar xarxes d'agents cooperatius, que compleix molts dels requeriments d'aquest projecte i en certa manera li serveix d'inspiració.

A més de la coincidència de perspectiva, els autors defensen que disposa d'una bona escalabilitat, però presenta dos grans inconvenients: no és una plataforma pública i no és un simulador de propòsit general per a xarxes de sensors (està implementat per a un problema específic de seguiment d'objectes mòbils mitjançant radars *Doppler*).

4.5 SENSE - Sensor Network Simulator and Emulator

SENSE [8] és un simulador força recent (primera aparició l'any 2004), que neix amb l'objectiu de convertir-se en un potent simulador de xarxes de sensors fàcil d'utilitzar. Pretén aconseguir aquest objectiu aplicant els últims avanços de la simulació basada en components a l'àrea de les xarxes de sensors i combinant diferents idees extretes dels altres simuladors presentats.

Entre els avantatges que aporta el disseny basat en components d'aquest simulador hi trobem: (1) possibilitat de paral·lelitzar l'execució de la simulació millorant-ne l'escalabilitat, (2) altes possibilitats de reutilització dels components, (3) bona extensibilitat degut al baix acoblament entre els components base.

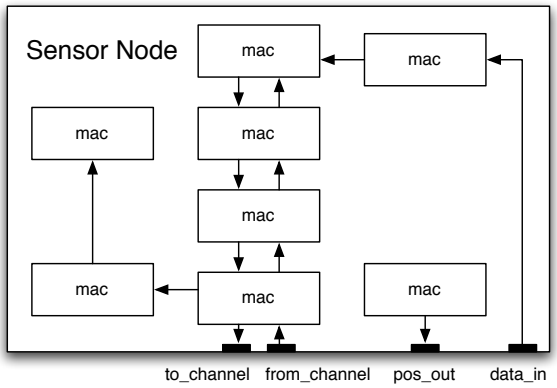


Figura 2: Esquema d'un sensor en *SENSE*

Està implementat en *C++* per intentar assolir el màxim rendiment possible, i els seus desenvolupadors desacrediten obertament la utilització de *Java* en aplicacions que requereixen alta eficiència, una opinió controvertida amb gairebé tants adeptes com oponents. A la figura 2 es mostra l'esquema de components que defineixen un node de la xarxa.

4.6 Repast

Repast[12] és un entorn de simulació basada en *steps* per a sistemes multi-agent, de codi lliure i molt extès en la comunitat. Algunes de les seves característiques més destacables són: (1) permet definir els agents en multitud de llenguatges, (2) ofereix una extensa col·lecció d'algorismes adaptatius llestos per a ésser utilitzats, (3) incorpora diverses utilitats de registre de les simulacions i anàlisi de resultats i (4) està disponible per a pràcticament qualsevol arquitectura i sistema operatiu actuals.

Malgrat *Repast* permet especificar la durada de l'acció d'un agent, s'ha d'especificar abans de realitzar-la. Les accions es realitzen llavors en un fil d'execució separat i es segueixen entregant events en paral·lel. L'entorn *multithreading* introdueix una important penalització en temps d'execució, elimina la repetibilitat de les simulacions i impedeix controlar la ocupació de l'agent ja que pot estar processant diferents tasques en fils paral·lels sense acumular-ne els temps d'ocupació.

4.7 Comparativa

A la taula 1 es presenta un resum de les diferències més notables entre els diferents simuladors presentats anteriorment:

4.8 Conclusions

Contrastant els requeriments exposats en la secció 3 amb les característiques dels simuladors estudiats s'evidencia que cap d'aquests satisfà les necessitats que impulsen aquest projecte. *Ns-2* queda descartat automàticament per la manca de model d'aplicació. *OMNeT++*, *J-Sim* i *SENSE* admeten aquest capa, però ofereixen un enfoc de massa baix nivell que introdueix càrrega i complexitat innecessàries per al modelat de les xarxes com a sistemes multi-agent. *Radsim* no és una plataforma pública ni de propòsit general i *Repast* a l'utilitzar una simulació basada en passos (*steps*) no ofereix suficient control temporal per modelar les comunicacions, la ocupació dels nodes i el consum d'energia.

Simulador	Llenguate	Disseny	Avantatges	Desventatges
ns-2 (+ extensió SensorSim)	C++	OO	nombre d'usuaris, algorismes enrutament implementats	extensibilitat, escalabilitat, manca de model d'aplicació
OMNeT ++ (+ extensió SenSim)	C++	BC	escalabilitat i extensabilitat	poca comunitat i algoritmes implementats
J-SIM WP	Java	BC	extensibilitat	*MAC 802.11* obligat, baix nivell
Radsim	Java	OO	enfoc multi-agent	propietari, específic per a una aplicació
Sense	C++	BC	extensibilitat, reusabilitat, escalabilitat	baix nivell, c++
Repast	Java	OO	enfoc multi-agent, nombre d'usuaris, eines d'anàlisi, algoritmes implementats	Manca de control temporal, Irrepetibilitat

Taula 1: Taula comparativa de simuladors actuals.

5 Planificació del projecte

En la taula següent es presenta una planificació detallada de les diferents tasques que composen l'execució d'aquest projecte, una breu descripció i el nombre d'hores aproximat que s'hi haurien de dedicar. Finalment s'inclou el diagrama de Gantt (Fig. 3) referent a la planificació d'aquestes tasques.

5.1 Tasques de documentació i anàlisi

- **Reunions de seguiment (20h):** Reunions amb el grup i el director per al realitzar un seguiment del projecte.
- **Introducció a les xarxes de sensors (15h):** Introducció al concepte de xarxes de sensors, literatura existent, sol·lucions que aporten i reptes que plantegen.
- **Introducció a la simulació per computador (15h):** Introducció a la teoria de la simulació per computador, aproximacions i algorismes existents, casos d'aplicació.
- **Estat de l'art dels simuladors per a xarxes de sensors (20h):** Investigació sobre l'estat de l'art dels simuladors per a xarxes de sensors: altres simuladors existents, funcionalitat que ofereixen, etc.
- **Requisits del projecte (20h):** Recollida de requisits i objectius que hauria de complir el simulador presentat.

5.2 Tasques de disseny i implementació

Totes les tasques d'implementació de noves funcionalitats inclouen la generació de testos i la seva documentació.

- **Disseny general (20h):**
  - **Arquitectura del simulador (12h):** Establir l'arquitectura que regirà el disseny del simulador, identificar-ne els components bàsics i decidir amb quin llenguatge de programació s'implementarà.
  - **Entorn de programació (8h):** Escollir, instal·lar i configurar l'entorn de desenvolupament, el sistema de control de versions i la plataforma de proves automatitzades.
- **Nucli del simulador (100h):**
  - **Disseny del nucli del simulador (40h):** Disseny dels components bàsics que intervindran en les simulacions, les seves interfícies i com interactuaran.

  - ***Scheduler* (8h):** Implementació del component encarregat de la planificació d'events futurs .
  - ***Field* (4h):** Implementació del component que representa el "món", on es donen tots els fenòmens.
  - ***Phenomenon* (8h):** Implementació del component que representa els fenòmens, és a dir, tot allò que és susceptible de ser observat.
  - ***Agents* (12h):** Implementació del component base de tots els agents, que representa la part amb capacitats computacionals dels sensors.
  - ***Sensors* (12h):** Implementació de les funcionalitats bàsiques dels sensors, components que permeten percebre el que està passant al món (*field*).
  - ***Actuators* (8h):** Implementació de les funcionalitats bàsiques dels actuadors, components que permeten realitzar accions que modifiquen l'estat del món i/o els seus elements.
  - ***ComponentFactories* (8h):** Implementació de les factories de components, és a dir, objectes encarregats d'introduir nous components a la simulació durant l'avanç d'aquesta.

- **Exemples de simulació (40h):**
  - ***MiniSim* (16h):** Implementació d'un petit exemple de simulació que utilitzi totes les funcionalitats i es vagi desenvolupant conjuntament amb el programa per utilitzar-la com a plataforma de proves d'integració.
  - ***CoverageProblem* (24h):** Implementació d'una simulació específica per aportar un exemple real d'utilització als futurs usuaris i realitzar demostracions.

- **Registre de la simulació (10h):** Disseny, implementació i proves del mòdul encarregat d'enregistrar la informació de tot el que succeeix en la simulació.

- **Model de comunicació (20h):**
  - **Disseny (8h):** Segona iteració del disseny de la plataforma, en la que es definirà el model que permeti comunicar-se als agents.
  - **Implementació (12h):** Implementació de les modificacions introduïdes als components afectats per el redisseny i dels components de comunicació.

- **Model de consum d'energia (30h):**

  - **Disseny (8h):** Tercera iteració del disseny de la plataforma, en la que es definirà el model model de consumició d'energia.
  - **Implementació (12h):** Implementació de les modificacions introduïdes als components afectats per el redisseny i el component bàsic *Bateria* amb algun exemple concret.

- **Interfície gràfica (40h):**
  - **Model de la interfície gràfica (12h):** Disseny i implementació del model de la interfície gràfica, que representarà l'estat de la simulació en l'instant de temps que s'estigui mostrant a la interfície gràfica.
  - **Visualitzador d'events (6h):** Disseny i implementació d'un panell que mostri tots els events que s'han donat en la simulació.
  - **Inspector del model (8h):** Disseny i implementació del panell que mostrarà tots els elements que existeixen en la simulació i el seu estat (propietats, variables internes, etc.).
  - **Visualització gràfica de la simulació (14h):** Disseny i implementació del panell que mostrarà una representació geogràfica de la simulació i la seva evolució.

5.3 Tasques de redacció

- **Informe previ (15h):** Preparació de l'informe previ imprescindible per poder presentar el projecte.
- **Redacció de la memòria (80h):** Recopilació de tota la informació generada i redacció de la memòria del projecte.

5.4 Cost del projecte

Tractar-se d'un projecte de desenvolupament d'una aplicació *software*, els costos que porta associats són principalment en l'apartat de recursos humans. Podem extreure un cost econòmic aproximat d'un programa d'aquest tipus a partir de la planificació temporal presentada en l'apartat anterior, agrupant les tasques segons el perfil de personal que ha de realitzar-les i aplicant-hi una dotació econòmica en €/hora (reflectits en la Taula 2).

A aquest cost hem d'afegir-li les despeses corrents (electricitat, aigua, etc.) i les de maquinari (plataformes de desenvolupament i prova) utilitzats per dur-lo a terme i obtindrem el cost total aproximat de la seva execució (reflexats en la Taula 3). Cal fer notar que no s'afegeix cap import en concepte de llicències, doncs un dels requeriments és precisament la utilització de programari lliure tant per realitzar el desenvolupament com en requisits de l'aplicatiu final.

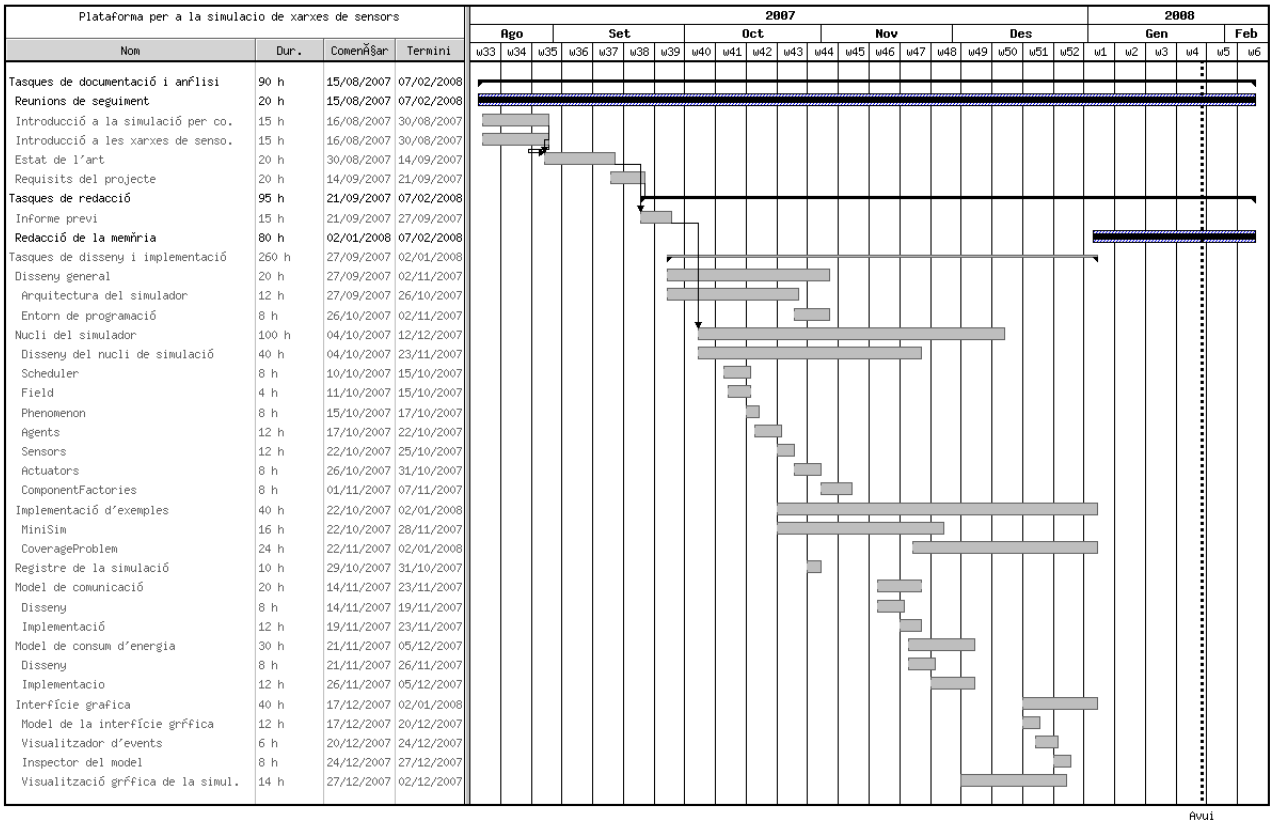


Figura 3: Diagrama de Gantt

Perfil	**Hores**	**Dotació**	**Total**
Programador	335h	30 €/h	10.050 €
Analista	100h	50 €/h	5.000 €
Director	40h	70 €/h	2.800 €
Cost en recursos humans			17.850 €

Taula 2: Estimació del cost en recursos humans

Concepte	**Dotació**
Recursos humans	17.850 €
Maquinari	750 €
Despeses corrents	300 €
Cost total	18.900 €

Taula 3: Estimació del cost total del projecte

6 Entorn i llenguatge de programació

6.1 Llenguatge de programació

L'arquitectura basada en components pràcticament ens obliga a utilitzar un llenguatge de programació orientat a objectes. Si s'utilitzés un llenguatge procedimental o funcional la plataforma hauria de contemplar les facilitats d'encapsulació, modularitat i reutilització inherents dels llenguatges orientats a objectes, augmentant enormement la complexitat del seu disseny i implementació.

Per a realitzar simulacions serà imprescindible conèixer o aprendre el llenguatge de programació en el que s'implementi la plataforma, de manera que la popularitat d'aquest serà important per a obtenir una bona acceptació dels usuaris.

Amb aquest nou requisit el ventall de llenguatges possibles queda reduït a tres: *Java*, *C++* i *C#*. Aquest últim però queda descartat pel requisit que la plataforma ha d'utilitzar programari lliure i gratuït ja que encara que el llenguatge en sí és lliure (*Standard ECMA-334* [13]), la llibreria de classes estàndard (*Microsoft .NET Framework*) i la principal eina de desenvolupament (*Microsoft Visual C#*) no ho són.

Tant *Java* com *C++* podrien servir per a la implementació del simulador ja que ambdós s'adeqüen als requisits del projecte. Algun dels simuladors presentats en l'estat de l'art es decideixen per a *C++* per la seva velocitat i bona gestió en la utilització de memòria. No obstant, aquest argument perd pes quan són varis els estudis [14] que atorguen molta més importància als algorismes utilitzats i quan existeixen ja comparatives d'on *Java* en surt vencedor. El llenguatge escollit per a la plataforma de simulació ha estat *Java* que, gràcies a la seva llibreria de classes i a la gestió automàtica de la memòria, millora la productivitat [15] dels programadors. A més a més facilita la documentació, introdueix menys errors i ens facilitarà les tasques de test.

6.2 Entorn de programació

Existeix un ampli ventall d'utilitats per al desenvolupament d'aplicacions en *Java*, així com per totes les tasques de control i gestió del projecte i els seus artefactes. Tractant-se d'una plataforma que s'alliberarà al públic, s'ha evitat vincular el simulador amb qualsevol eina de desenvolupament específica, permetent que cada usuari pugui escollir aquelles eines amb les que es trobi més còmode. La única excepció és l'eina de construcció del projecte ja que per a les tasques d'automatització dels processos de compilació s'ha d'utilitzar l'estàndard (*de-facto*) per a *Java*: *Apache Ant* [16].

Pel que fa a les eines utilitzades pel desenvolupament del propi projecte en destaco les següents:

- **CVS (Concurrent Versioning System)**. Un sistema de control de versions que és programari lliure, disposa de clients per a totes les plataformes i és el més suportat per els entorns integrats de desenvolupament (*IDE's*).

- **l'*IDE* de Netbeans**. Per a les tasques de disseny s'ha utilitzat l'*IDE* netbeans[17] que, des de la versió 5.5, aporta un versàtil entorn de modelat UML

- **Eclipse** Com a entorn de programació s'ha utilitzat *Eclipse*, que ofereix un millor rendiment per les tasques d'edició de codi.

7 Arquitectura del simulador

Les xarxes de sensors són sistemes dinàmics complexes, que combinen processos continus, discrets i basats en events. La integració d'aquests processos serà la clau del simulador i dependrà en gran mesura del paradigma de programació escollit per implementar-lo.

Entre aquests paradigmes hi trobem (1) la programació procedimental -el més comú-, (2) la programació basada en autòmats finits que descriu els programes com a màquines d'estat finites, (3) la programació orientada a objectes on es modela el programa mitjançant objectes i les seves interaccions i (4) l'arquitectura basada en components.

En l'intent d'extreure el millor de cadascuna d'elles, s'ha optat per a la combinació de l'arquitectura basada en objectes i la basada en components. La plataforma de simulació es divideix en dues "capes" relacionades mitjançant l'herència.

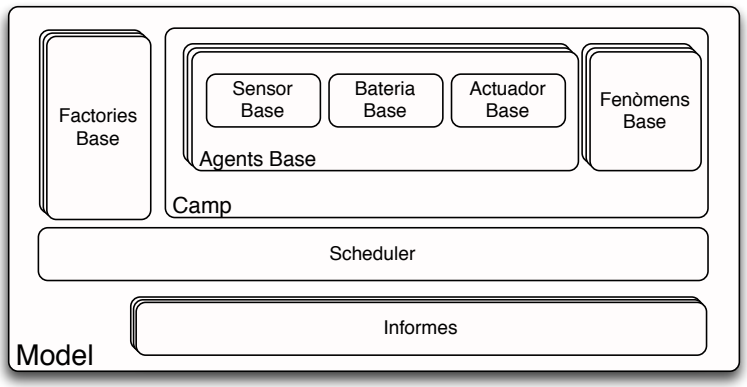


Figura 4: Arquitectura del simulador. Capa inferior.

La capa inferior utilitza una arquitectura basada en objectes, representada en la figura 4. Proporciona l'estructura bàsica per la simulació aportant (1) tots els objectes necessaris per a realitzar una simulació basada en events i (2) classes abstractes del tipus d'objectes que formen qualsevol xarxa de sensors (agent, bateria, sensor, etc..). A aquestes classes últims els anomenarem *components base* ja que proporcionen al usuari l'estructura bàsica per la seva simulació proporcionant les següents funcionalitats: (1) controlar els events generats i l'execució dels seus processos, (2) modelar les relacions jeràrquiques i de control entre diferents components i (3) regular-ne la comunicació. Per cadascun d'aquests components base s'ha definit també una interfície que actua de contracte. És possible doncs integrar objectes d'altres llibreries o plataformes fent que el nou objecte implementi la interfície corresponent.

En la figura 5 es mostra la capa superior, el nivell en que treballen els

usuaris de la plataforma. Està formada per instàncies dels *components base* i extensions d'aquests definides per l'usuari. Entre ells s'utilitzen una arquitectura basada en components, on cada cadascun exposa la seva funcionalitat desencadenant certs processos en resposta als events enviats per els altres. La llibertat oferta en aquest nivell de la plataforma disminueix l'acoblament entre els components (millorant-ne la reusabilitat) i abstreu l'usuari dels detalls de funcionament de la plataforma (facilitant l'aprenentatge).

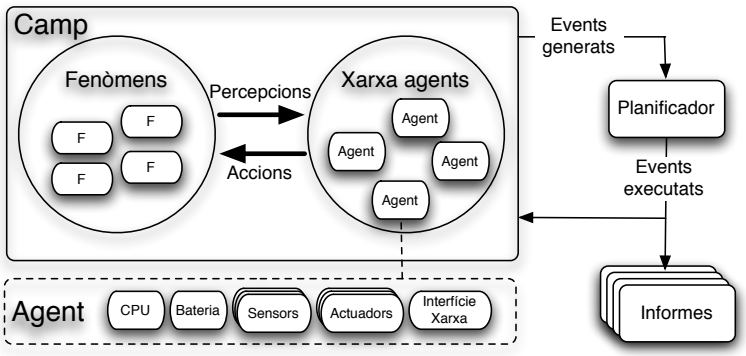


Figura 5: Arquitectura del simulador. Capa d'usuari.

7.1 *Model*

El model és l'objecte que *encapsula* tots els elements referents a una simulació. Aporta cohesió entre els elements que hi participen i actua de *façana* per a tot el nucli de simulació. A més, s'encarrega de la construcció i inicialització de tots aquests elements.

Els components de la simulació implementen dos mètodes exposats en la interfície del model (diagrama 6) especificant les accions per a la seva construcció i inicialització: *setup()* i *init()*. El mètode *setup()* implementa les tasques de *construcció* del component en qüestió mentre que el mètode *init()* implementa les tasques d'inicialització. El model és l'encarregat d'executar aquests mètodes en l'ordre adequat (primer es construeix tot el sistema i després s'inicialitza). L'algorisme utilitzat per les dues és el mateix que consisteix en anar cridant els mètodes *setup(long)* i *setup()* respectius de cada component.

El model també és l'encarregat de generar el paràmetre de tipus *long* per la fase de construcció que passa al mètode *setup()* . Aquest paràmetre és un nombre aleatori que serveix de llavor perquè cada objecte inicialitzi els seus propis generadors de nombres, en cas de necessitar-los. D'aquesta manera dues execucions de la mateixa simulació produiran exactament el mateix resultat (sempre que preparem el generador de nombres aleatoris del model amb la mateixa llavor), satisfent el requeriment de repetibilitat. La

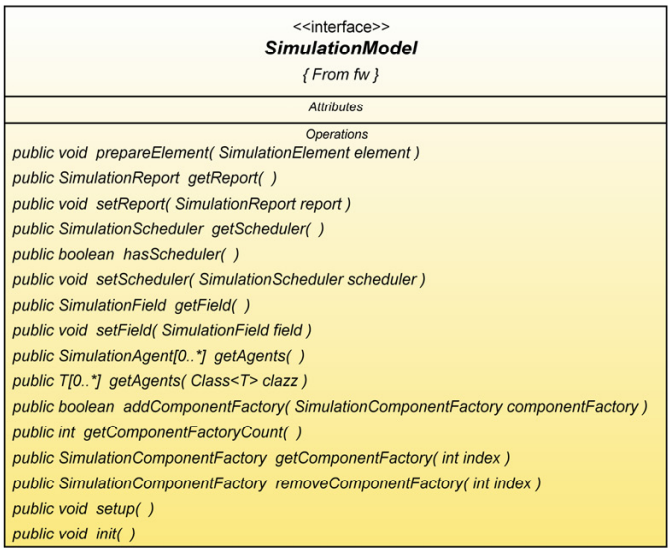


Figura 6: Interfície del model de simulació.

plataforma de simulació ofereix una classe abstracta, el *AbstractSimulationModel*, d'un model amb totes les funcionalitats bàsiques. Els usuaris de la plataforma han d'estendre aquest model per afegir-li els objectes de la present simulació i personalitzar-lo si és el cas.

7.2 Events

Com que realitza una simulació basada en events discrets, la simulació és una seqüència d'events que succeeixen en temps de simulació determinants i que produeixen canvis d'estat en el sistema. En el simulador la classe *event* és la que codifica aquests successos. Com podem veure en la figura 7, porten associat l'instant de temps de simulació en el que tenen lloc i l'objecte que els ha originat.

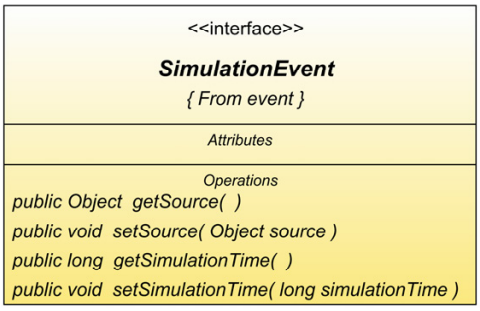


Figura 7: Interfície event.

La creació de nous events es porta a terme estenent i creant una jerarquia

d' aquests elements bàsics (creant-ne subclasses). La creació de noves classes permet als receptors d'events identificar l'event per la seva classe i així poder generar diferents reaccions per a diferents events.

A més a més dels camps bàsics, els events poden contenir informació addicional necessària per a poder processar l'event i per a que pugui utilitzar-se com a mecanisme entre components. Així per exemple podríem tenir un event generat per un component sensor que informes d'un objecte detectat on a més a més dels camps bàsics l'event hauria de contenir la posició del objecte: *EventObjecteDetectat(t=10)(x=10,y=20)*.

Part de la feina de disseny del nucli del simulador serà doncs definir tots els tipus d'events o successos necessaris per a la gestió i control de la simulació. En el diagrama 8 veiem el primer esglaó de la jerarquia d'events referents al nucli de simulació, tres interfícies que s'utilitzen per a informar de l'addició, modificació i eliminació dels elements de la simulació.

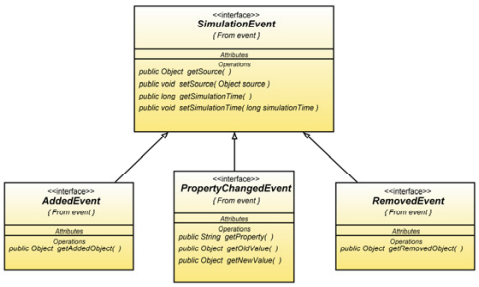


Figura 8: Diagrama d'events bàsics de canvi d'estat.

Aquests events són de tipus informatiu de successos ocorreguts durant l'execució de la simulació. Aquest tipus d'events, com es pot veure en els diagrames 7 i 8, només disposen de la informació referent a l'objecte en què s'originen i no tenen receptor. En la majoria de casos però especificar només l'origen no és suficient i és necessari especificar un receptor. Per exemple, l'event *EventApagarSensor(t=12)* originat en un component *Agent* no té sentit si no en coneixem el destinatari.

Aquesta situació es dóna amb tots els events que s'utilitzen com a mecanisme de comunicació entre components, donant lloc a un altre dels tipus bàsics d'event, el **Missatge**. Com que la informació addicional referent al destinatari només serà utilitzada pel planificador d'events, que serà el que realitzarà l'entrega del event, aquesta classe s'ha dissenyat mitjançant el patró de disseny *decorator*, tal com es mostra en el diagrama 9. A més a més de contenir destinatari, els missatges tenen la particularitat de poder-se planificar en un punt de la simulació i oirginar-se en un temps posterior. Per tant en els missatges es pot especificar un temps de emissió i un temps de simulació. De cara al simulador, el moment en què s'ha emès no és rellevant (no afecta la seva planificació d'entrega), però per a l'usuari que inspeccioni

una situació serà una dada molt útil així que s'ha decidit afegir-la (*getSendTime()*).

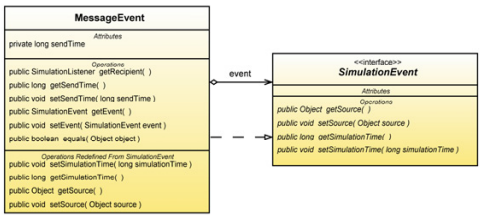


Figura 9: Diagrama *UML* dels Missatges.

Disposem doncs dels events com a mecanisme de comunicació entre components, però ens falta definir la interfície que han de complir aquests components per a declarar-se'n receptors. Tots els receptors d'events han d'implementar la interfície *SimulationListener* amb mètode *simulationChanged(SimulationEvent)*, que s'executarà per informar al objecte de la recepció d'un event, passant aquest per paràmetre (veure el diagrama 10).

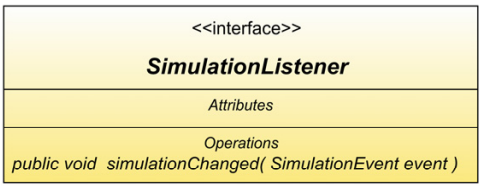


Figura 10: Interfície per la recepció d'events.

El problema és que si deixem als usuaris la implementació d'aquest mètode, com que aquest no distingeix entre els diferents tipus d'events aquest haurien de passar-lo per una cadena de crides del mètode *instaceof* per descobrir-ne el tipus i processar-lo adequadament.

Per a abstraure l'usuari d'aquests detalls, el mètode *simulationChanged(SimulationEvent)* ja el proporciona la plataforma mentre que el que han d'implementar els usuaris per a processar un event és la sobrecàrrega del mètode *process(SimulationEvent)* on *SimulationEvent* és del tipus del event en concret. Per exemple, per definir el codi que gestionarà la recepció de dades en un component s'hi afegirà un mètode *process(DataReceivedEvent)* i per definir-ne la gestió de les mesures rebudes dels sensors *process(MeasureEvent)*.

D'aquesta manera els desenvolupadors de components en la plataforma poden definir els seus propis events i aquests seran integrats al sistema de forma transparent. Si un component coneix el tipus específic d'event el tractarà com a tal, i si no és així realitzarà l'acció que tingui predefinida per al seu supertipus, escalant en la jerarquia de tipus d'events fins arribar,

com a màxim, a l'acció predefinida per un *Event* desconegut (normalment enregistrar un avís a l'arxiu de registre).

El problema és que la sobrecàrrega no es pot realitzar directament perquè en Java aquesta es realitza només *en temps de compilació*. Per exemple, en Java la sortida d'executar el codi del llistat 1 no és *DadesRebudes* sinó *EventDesconegut*.

Això és degut a que l'enllaç entre la crida *process* de la línia 14 i el mètode el realitza el compilador, per al qual *e* és del tipus *Event*, i només *Event*. D'aquesta manera, la crida queda enllaçada amb el mètode *process(SimulationEvent)* de la classe *ComponentBase*.

```
1  class ComponentBase implements SimulationEventListener {
2      public void process(SimulationEvent event) {
3          System.out.println (“ Event Desconegut”);
4      }
5  }
6
7  class Component extends ComponentBase {
8      public void process(DataReceivedEvent event) {
9          System.out.println (“ Dades Rebudes”);
10     }
11      public static void main(String []) {
12          ComponentBase c = new Component();
13          Event e = new DataReceivedEvent();
14          c.process(e);
15      }
16 }
17
```

Listing 1: Exemple de sobrecàrrega de mètodes

Encara que no sigui nadiu, és possible implementar aquest comportament en *java* mitjançant l'ús de *Reflection*. Aquesta *API* de la llibreria estàndard que permet inspeccionar els mètodes i propietats de les classes, instanciar-ne objectes i invocar mètodes en temps d'execució a partir dels seus identificadors. Així el mètode *simulationChanged*(*SimulationEvent*), que ja es dóna implementat per a tots els components susceptibles de ser estesos pels usuaris de la plataforma, és el que implementa aquest *Reflection*.

7.3 *Scheduler*

L'*Scheduler* o planificador d'events és el component encarregat d'integrar tots els events generats per la resta de components del simulador, decidir en quin ordre s'entreguen i controlar el temps de simulació.

La primera funcionalitat que ofereix és el mètode *advance()*, encarregada d'avançar un pas de simulació. Un avanç en la simulació fa referència al procés d'un event. L'event a processar en cada pas de simulació, és l'event amb menor temps de simulació encara no processat. Si l'event no té

destinatari l'avanç de la simulació significarà simplement informar als components de recollida de dades (*Report*, que veurem més endavant) del event en qüestió. Per als events amb destinatari (events del tipus *Message*) la gestió és més complexa ja que abans s'ha de notificar al component destinatari del missatge (sempre que aquest no estigui ocupat) i esperar a que aquest processi l'event. Com que es tracta del component que controla l'avanç de la simulació, l'*scheduler* s'encarrega també de mantenir el rellotge de la simulació, que es pot consultar mitjançant el mètode *getSimulationTime()*.

Com veiem en el diagrama 11, els components disposen de tres mètodes per a generar events: (1) *addEvent(SimulationEvent)* que genera un event en l'actual instant de simulació, (2) *addEvent(SimulationEvent, delay)* que genera un event per a executar-se al cap de *delay* nanosegons de simulació i (3) *addMessage(SimulationEvent, SimulationListener, delay)* que genera un event per un component en concret (amb destinatari) per executar-se al cap de *delay* nanosegons de simulació.

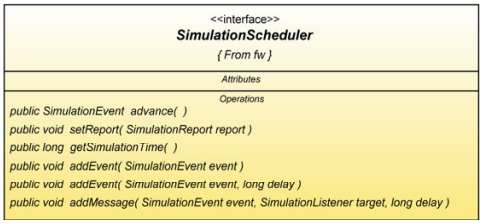


Figura 11: Interfície del planificador de tasques.

Aquest planificador utilitza una estructura *heap*, que manté la llista d'events futurs ordenada segons dos factors: el temps de simulació en què ocorren i, pels events simultanis, l'ordre en el que han arribat al planificador (important aquest detall per al correcte funcionament dels processos d'inicialització). L'avantatge d'utilitzar aquest *heap* enfront altres estructures de dades és que presenta una complexitat $O(log(n))$ tant per a l'addició de nous events com per l'extracció del següent a executar.

7.4 *Field*

Per realitzar simulacions necessitarem també un model de l'espai on interaccionen els diferents elements *físics* que hi participen. Aquests diferents elements *físics* són *phenomena*. El camp de simulació o *field* és l'objecte encarregat de realitzar aquesta funció, mitjançant la interfície mostrada en el diagrama 12.

Els mètode *addPhenomenon* serveix per afegir elements al camp i així puguin ser observats mentre el *removePhenomenon* s'utiliza per eliminar-los quan desapareixen. A més de portar el control dels fenòmens actius, també serà tasca del camp informar de la seva aparició i desaparició mitjançant els

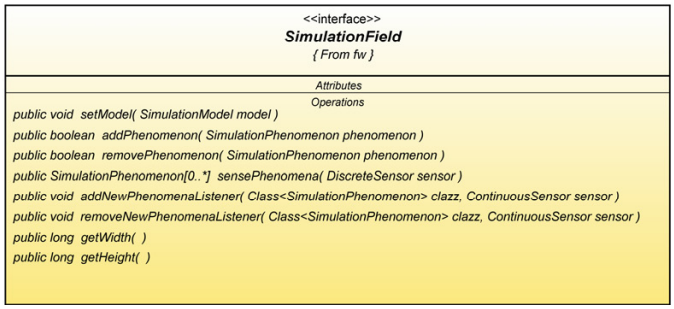


Figura 12: Interfície del camp de simulació.

events *PhenomenonAddedEvent* i l'anàleg *PhenomenonRemovedEvent*.

De *Phenomena* en podem distingir de dos tipus: els fenòmens que són agents i els que no. Els agents són modelats també com a fenòmens perquè tenen una presència física al món i també han de ser definits com objectes físics del camp (també poden ser detectats). Els agents són un fenomen particular ja que: poden contenir sensors que els permeten rebre observacions d'altres fenòmens del camp i poden contenir actuadors que els permet introduir nous fenòmens o modificar fenòmens del camp.

Com que es tracta de l'únic element del sistema que coneix tots els fenòmens actius, el camp ha d'actuar com a enllaç i filtre perquè els sensors dels agents puguin detectar aquests fenòmens. Degut a l'existència de dos tipus de sensors de diferent naturalesa (exposats en la secció 7.6.1), existeixen dues maneres força diferents de consultar aquesta informació:

1. Mitjançant el mètode *sensePhenomena()*, que retorna una llista de tots els fenòmens actius que és capaç de detectar el sensor.

2. A través dels mètodes *addNewPhenomenaListener()* i *removeNewPhenomenaListener()*, que permeten subscriure un sensor perquè sigui avisat cada vegada que s'afegeixi un fenomen d'un tipus determinat al camp i des-subscriure'l perquè deixi de rebre aquestes notificacions.

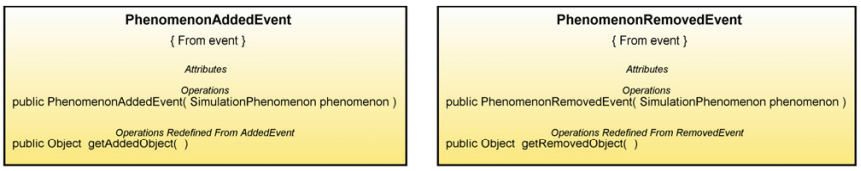


Figura 13: Events de notificació d'aparació/desaparició de fenòmens.

7.5 Fenomen

Els fenòmens són tots aquells successos susceptibles de ser observats dins la simulació, incloent als propis agents. Són doncs els components amb major dependència del domini que s'estigui simulant, ja que poden representar des d'un objecte mòbil fins a la temperatura en una habitació. La interfície del component *Phenomenon* (diagrama 14 és doncs molt bàsica i cobreix només dues funcionalitats: (1) la capacitat de desaparèixer de la simulació perquè el succés ha deixat de ser actiu, representada amb el mètode *die()* i (2) la possibilitat de decidir si un sensor pot detectar-lo o no.

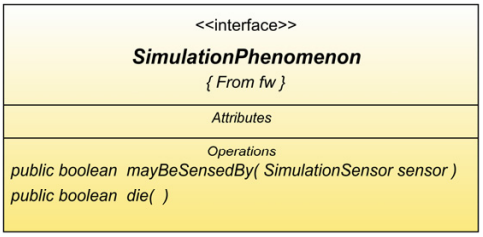


Figura 14: Diagrama de la interfície *Phenomenon*.

7.6 Agent

Els sistemes multi-agent modelen els nodes d'una xarxa de sensors com a agents autònoms. Per tant i com que l'objectiu de la plataforma és proporcionar un simulador de sistemes multi-agents per a xarxes de sensor s'haurà de definir el component agent (component que modelarà cada node de la xarxa). L'agent és un component d'alt nivell format alhora per altres components: la *CPU*, la *font d'energia*, *sensors*, *actuadors* i interfícies de comunicació.

En concret l'objecte agent el que implementa és la component *CPU* i implementa tots els processos de procés d'informació i presa de decisions del agent. La major part de la interfície exposada per aquest component (diagrama 15) són mètodes que permeten gestionar els diversos components que composen l'agent. Igual que feia el *field* amb els fenòmens, l'agent haurà d'informar la resta de la simulació dels elements que se li afegeixin o eliminin mitjançant els events mostrats en la figura 16.

L'agent disposa també de les funcionalitats propies d'un fenomen donat que representa una entitat física amb la qual la resta d'elements del sistema podria interactuar. Com a fenòmens als agents se'ls obliga a definir la seva ubicació dins el camp de simulació (una propietat que no ha de ser especificada per a tots els fenòmens) que es pot consultar mitjançant el mètode *getLocation()*.

L'*scheduler* processa els events seqüencialment esperant per a processar el següent event a que el destinatari acabi de processar l'event actual. Si no

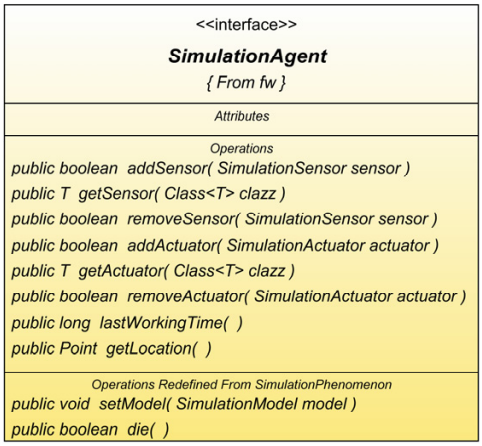


Figura 15: Diagrama de la interfície *Agent*.

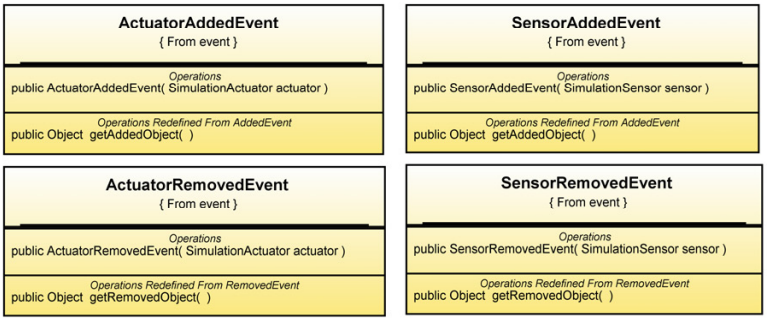


Figura 16: Events de notificació d'addició/eliminació de sensors i actuadors.

es porta un control del temps que cada agent dedica al procés d'un event, la percepció serà que els càlculs realitzats pels agents són instantanis (comencen i acaben en el mateix instant de simulació de l'event). Per evitar-ho, l'agent retorna el temps de procés cada event un cop aquest ha estat processat. En la implementació bàsica d'agent aportada pel simulador aquest temps es calcula com el temps *CPU* real. El mètode de la interfície *lastWorkingTime()* és l'encarregat de retirar a quin instant de simulació s'ha estès l'últim event processat per l'agent. D'aquesta manera s'ofereix la funcionalitat més comú però es permet modificar el comportament en qualsevol agent específic simplement sobreescribint el mètode *lastWorkingTime()*. Per exemple, fent que sempre retornés sempre 0 es simularia la capacitat de realitzar qualsevol càlcul de forma instantània. D'aquesta manera, l'*scheduler* pot consultar la disponibilitat d'un agent abans d'entregar-li un event i retrassar-ne l'entrega si el troba ocupat. A continuació, el següent pseudoalgorisme mostra el càlcul del event rebut:

```
1   public void simulationChanged(Event e) {
```

```
2        long startTime = SystemTimeInNanoseconds();
3        this.process(e);
4        long endTime = SystemTimeInNanoseconds();
5        this.lastWorkingTime = (endTime − startTime) ∗ scaleFactor;
6    }
7    public long lastWorkingTime() {
8        return this.lastWorkingTime;
9    }
```

7.6.1 Sensor

Com ja s'ha comentat, els sensors són els components que permeten als agents percebre senyals o estímuls del món (representats en aquesta plataforma com a fenòmens). El primer que es necessita doncs és una manera de definir les capacitats receptives de cadascun d'aquests sensors, definició que formalitzarem amb un *Filtre*. La tasca del filtre és decidir, donat un fenomen determinat, si el sensor al que representa en pot obtenir observacions o no.

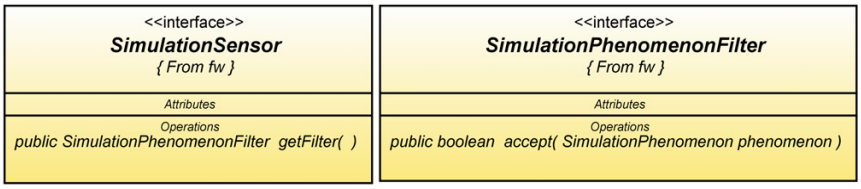


Figura 17: Interfícies Sensor i Filtre.

Pel que fa a l'acció de percebre s'ha de distingir entre dos tipus diferents de sensors (diagrama 18):

1. **Discrets** (deriven del component base *AbstractDiscreteSensor*): Aquest tipus de sensors prenen mesures cada cert interval de temps o quan reben l'ordre explícita de fer-ho. Es modelen utilitzant un event dirigit al sensor (SenseEvent()) que fa de *disparador*. El mètode que processa aquest event ve implementat per la plataforma en el component base (*AbstractDiscreteSensor*) i executa un mètode del *field* (*Field::sensePhenomena(DiscreteSensor)*) que retorna tots els fenòmens actius observables pel sensor (els no observables són filtrats pel *field* utilitzant el filtre del sensor). Finalment, el mètode procés del event crida al mètode *sense(Phenomena)* del mateix sensor passant-li els fenòmens d'interès.

2. **Continus** (deriven del component base *AbstractContinuousSensor*): L'altre tipus de sensors són aquells que reben lectures de forma contínua

o a intervals extremadament curts, per als quals no podem programar un event que faci de disparador ja que hauríem d'estar programant events contínuament sobre temps continu. Per això el que fa el component base dels sensors continu (*AbstractContinuousSensor*) és subscriure's al camp de simulació per observar un classe de fenòmens (*SimulationPhenomenaClass*) mitjançant la crida al mètode *Field::addNewPhenomenaListener*. A partir d'aquell moment, cada vegada que s'afegeixi un nou fenomen d'aquesta classe a la simulació, el camp comprovarà si aquest sensor l'hauria de percebre (mitjançant el filtre del sensor) i el notificarà cridant-li el mètode *phenomenonAdded(Phenomenon)* en cas afirmatiu.

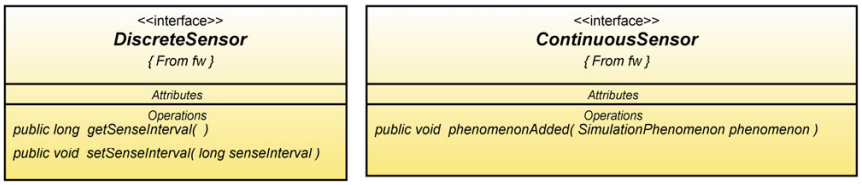


Figura 18: Tipus de sensors.

Un cop realitzada la detecció, els sensors disposen d'accés a tota la informació encapsulada en els mètodes públics dels fenòmens capturats. El mètode *Sense(Phenomena)*, en sensors discrets, i el mètode *phenomenonAdded(Phenomenon)*, en continus, són els encarregats d'implementar quines observacions el sensor obté a partir dels fenòmens segons les característiques del sensor que s'està simulant i remetre aquestes mesures al agent. Com que aquest procés és totalment depenent de l'aplicació que s'estigui simulant la plataforma deixa la implementació d'aquests dos mètodes al usuaris.

Per exemple, un radar detector d'objectes (sensor discret): (1) obtindria els diferents fenòmens que passin el seu filtre cada *senseInterval* temps, (2) els hi demanaria la posició, (3) calcularia els valors de freqüència i amplitud (afegint-hi un error) que hauria rebut realment el radar i, finalment, (4) transmetria aquests valors a l'agent utilitzant un event de mesura (diagrama 19).

7.6.2 Actuador

L'altre tipus d'elements de què disposen els nodes de xarxa són els actuadors, dispositius capaços d'introduir nous fenòmens a la simulació o afectar els existents. L'operativa d'aquest component bàsic és més senzilla que la dels sensors, ja que tots ells funcionen de forma discreta, és a dir, només actuen de forma periòdica o com a conseqüència d'una comanda de l'agent que els controla. L'ordre d'actuar es dóna mitjançant un event del tipus *ActuateEvent*, que enviarà el propi actuador (si és periòdic) o l'agent que

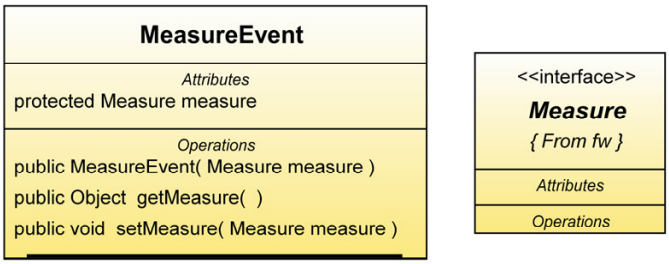


Figura 19: Tractament de les mesures.

el controli. L'event, a més a més, pot contenir qualsevol altra informació convenient per precisar l'acció.

Per facilitar la implementació, el component base *AbstractActuator* implementa el mètode que processa els events de tipus *ActuateEvent* gestionant l'addició dels nous fenòmens al camp. L'únic que ha d'implementar l'usuari és el mètode *actuate()* que retorna la llista de fenòmens a introduïr al camp per efecte de l'acció del sensor.

7.6.3 *ComponentFactory*

En moltes simulacions serà necessari disposar d'algun component que s'encarregui de generar nous elements durant el seu curs. Habitualment es tractarà de fenòmens o agents. Per exemple, el desplegament típic en xarxes que es troben en hàbitats inhòspits o inacessibles es realitza generalment en diverses fases ja que hi ha una part dels nodes de la xarxa que després d'un cert temps esdevindran inactius (per bateria esgotada, reparació o simplement desapareixeran per acció d'algun fenomen del entorn). S'ha preparat el component base *ComponentFactory* (diagrama 20) destinat a oferir aquesta funcionalitat, però no exposa cap mètode ja que tota la seva operativa depèn del domini que s'estigui simulant.

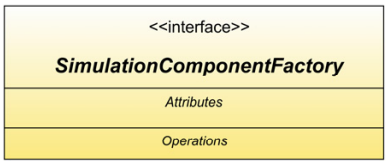


Figura 20: Factoria de components.

7.6.4 *Report*

Perquè les simulacions siguin d'alguna utilitat, la plataforma ha de permetre recollir-ne els resultats i analitzar-los per extreure'n conclusions. Aprofitant que tota la informació del que ha succeït en la simulació passa per l'*scheduler*

en forma d'events, en tindrem prou amb transmetre'n una copia a un component que enregistri, analitzi o mostri aquesta informació: el *report* o informe de la simulació. En el diagrama 21 es mostra la interfície exposada per aquest component i la implementació bàsica *AbstractReport* oferta per la plataforma. Aquesta implementació inclou la utilització d'un *thread* propi per evitar bloquejar el simulador amb les tasques d'anàlisi.

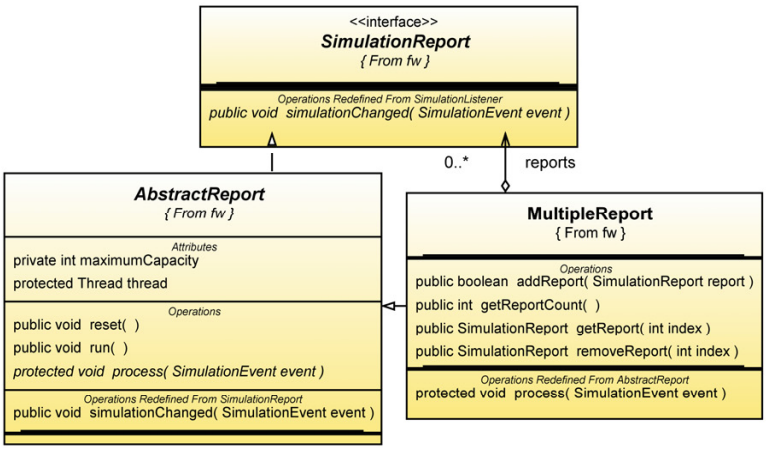


Figura 21: Informes de la simulació.

En certes situacions resultarà interessant repartir aquesta feina d'anàlisi entre diferents informes per facilitar-ne el desenvolupament, augmentar la reusabilitat, activar o desactivar informes independentment uns dels altres, etc. Per no complicar el planificador s'entrega també amb la plataforma un informe *MultipleReport* que aporta aquesta funcionalitat utilitzant un patró *composite*: manté una llista de diversos informes i entrega una rèplica de tots els events rebuts a cadascun d'ells.

7.7 Model d'energia

En les xarxes de sensors el control del consum d'energia dels seus components es presenta com un punt clau i, per tant, s'ha tindre en compte en el seu modelat. En la majoria de les xarxes de sensors, els nodes disposen d'una font d'energia limitada (una pila o bateria). Per tant l'objectiu d'una xarxa de sensors no és tan sols realitzar la seva tasca sinó fer-ho minimitzant l'energia consumida o maximitzant el temps de vida de la xarxa.

Per a incorporar el model d'energia en el simulador s'ha proporcionat un nou component per l'agent (la bateria), s'ha modificat la resta de components per modelar el seu consum energètic i s'han modificat el planificador i el camp perquè abans de fer una crida a algun component facin la comprovació de sí a aquest encara li resta encara energia.

7.7.1 Bateria

El primer pas en aquesta direcció passa per la introducció d'un nou component base: la bateria. Aquest component s'encarrega de modelar la font d'energia que utilitza el node, portant control sobre l'energia disponible i entregant-la quan els components ho requereixin. La interfície exposada per complir aquests requeriments es mostra en el diagrama 22 i s'explica per si mateixa.

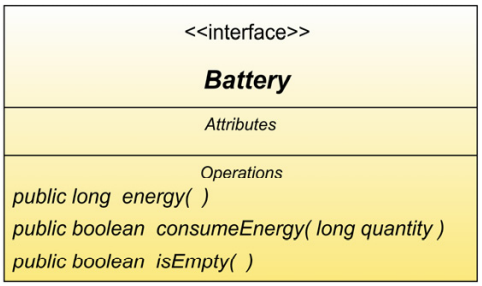


Figura 22: Interfície bateria.

Amb la plataforma de simulació s'aporten dos models de bateria llestos per ser utilitzats:

1. *DefaultBattery*: una bateria senzilla amb certa càrrega inicial que es decrementa en proporció directa al consum dels elements que alimenta.

2. *InfiniteBattery*: una "bateria" infinita que s'utilitzarà per modelar nodes de la xarxa que estiguin connectats a algun tipus de font d'alimentació externa inesgotable.

Actualment, aquests són els dos únics models de bateria que es proporcionen en la plataforma. En un futur n'està planificada la implementació d'altres més realistes i variats on la descàrrega varïi segons segons el temps de vida, la càrrega actual, les condicions ambientals, etc. o amb possibilitat de ser recarregades (bateries solars).

7.8 Modelat del consum energètic dels components

Els components als quals hem de proporcionar mètodes per a modelar el seu consum energètic són els que formen el component agent: cpu, sensors i actuadors. La necessitat de disposar d'energia per poder realitzar la seva tasca s'expressa formalment implementant la interfície *EnergyConsumer* (diagrama 25). El mètode *hasEnergy()* d'aquesta interfície serveix per a comprovar si el component disposa o no d'energia. La simulació del consum energètic és la part més complexa de la gestió d'energia ja que en molts casos és un procés continu i a més a més depèn del tipus de component:

- Per als actuadors el model és força senzill donat que només necessiten energia en el moment d'actuar i sempre ho fan en temps discrets. L'únic que han de fer és consumir certa quantitat d'energia (normalment fixada en la inicialització però també podria variar amb el temps) just abans d'actuar, o evitar fer-ho si la bateria no els ha pogut facilitar l'energia necessària.

- Els sensors presenten un major problema, ja que consumeixen energia pel sol fet d'estar actius però no necessiten generar cap event mentre no detectin cap fenomen. Com que la bateria és compartida, s'ha d'anar restant el consum dels sensors durant l'avanç de la simulació ,no tan sols quan aquests prenen mesures, ja que sinó es permetria als altres components lligats a la mateixa bateria consumir energia quan aquesta ja hauria quedat exhaurida pel sensor. Per solucionar aquest dilema es discretitza el consum dels sensors, és a dir, consumeixen energia cada cert interval de temps en que estan operatius.

- El consum dels agents varia en funció de la utilització que facin del seu processador. Es tracta doncs de dividir el consum de l'agent en intervals d'espera (*standBy*) i activitat(*working*), aplicant el consum corresponent a l'interval en canviar entre aquests estats. Com que el control d'aquests canvis d'estat ja esta implementat per determinar la ocupació de l'agent (explicada en la secció 7.6), n'hi ha prou amb consumir certa quantitat d'energia proporcional al temps de l'últim interval just abans de fer el canvi. En el diagrama 23 veiem la funcionalitat destinada a aquest tractament que s'ha afegit al component base *agent* de la plataforma.

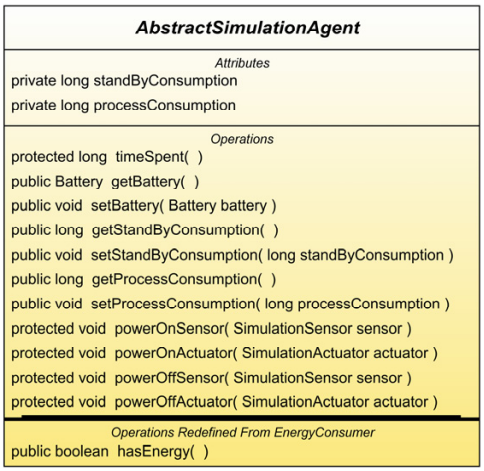


Figura 23: Mètodes de control del consum per als agents.

Com que els sensors i actuadors consumeixen energia del node, l'agent controlador de cada node ha de poder apagar-los, és a dir, desconnectar-los de la bateria perquè deixin de consumir quan no són necessaris. Entre els mètodes del component base *agent* s'hi troben les utilitats *powerOnSensor*, *powerOffSensor*, *powerOnActuator* i *powerOff* que serveixen a aquest objectiu encenent o apagant el sensor o actuador indicat. En realitat, el que fan aquests mètodes és enviar uns events especials (figura 24) a la bateria que reacciona connectant-se o desconnectant-se d'ells mitjançant la funció *setBattery()* del sensor o actuador corresponent. Un cop desconnectats de la bateria l'*scheduler* descartarà tots els events destinats a aquests components ja que la comprovació d'energia d'aquests (*hasEnergy()*) indicarà que estan inoperatius.

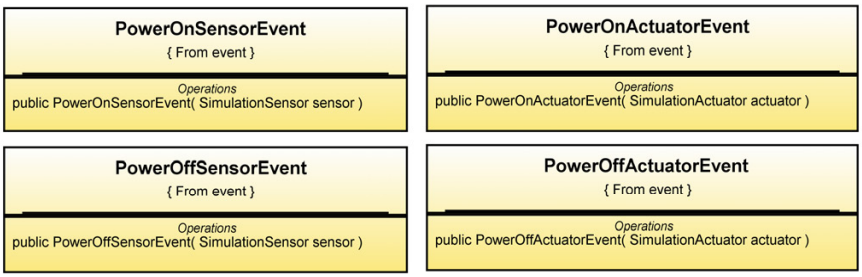


Figura 24: Events d'encès i apagat.

Finalment, s'han de modificar l'*scheduler* i el camp perquè no realitzin cap crida als components que no disposin de bateria.

El *scheduler* comprovarà abans de notificar l'entrega d'un event a un component si aquest té o no energia per a processar-lo (mitjançant una crida al mètode *hasEnergy()* del component). Si el component no té energia aquest event es descartarà.

```
1   if ((!source instanceof EnergyConsumer) || source.hasEnergy()) {
2       source.simulationChanged(event);
3   }
```

El mateix procés és realitzat pel camp quan ha de notificar als sensors continus que s'ha afegit un nou fenomen.

7.9 Model de comunicacions

El model de comunicacions és la part de la plataforma encarregada de simular i gestionar la comunicació entre els agents. La comunicació entre agents consisteix en la **transmissió** (acte d'enviar) de **paquets** (unitat d'encapsulament) de **dades** (informació). No s'ha de confondre la transmissió entre

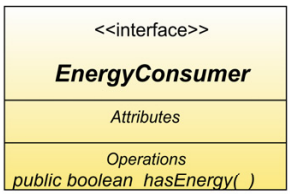


Figura 25: Interfície pels components que necessiten energia.

agents amb l'enviament de missatges d'events entre components explicat en la secció 7.2.

S'introdueix així la interfície *Data* (diagrama 26) que implementaran totes les classes que contenen informació transmissible i la classe *DataPacket*, que encapsula aquesta informació afegint-li dades de control. El fet que els paquets de dades siguin considerats dades en si mateixos ens permet crear diferents capes de xarxa.

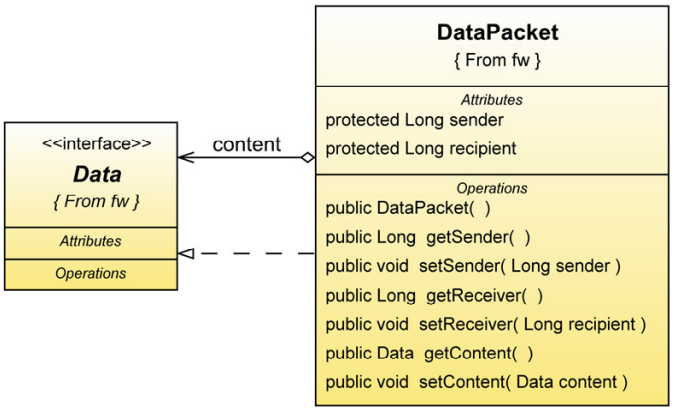


Figura 26: Interfície *Data* i classe *DataPacket*.

El model de comunicació s'ha implementat utilitzant els components base definits en el simulador, sense haver-lo hagut de modificar. Els agents es comuniquen generant un tipus especial de fenòmens mitjançant actuadors i percebent-los utilitzant sensors.

Com que no existeix cap protocol de xarxa dominant per a les xarxes de sensors, el nucli de la plataforma manté el màxim nivell d'abstracció possible definint tant sols les tres interfícies bàsiques mostrades en el diagrama 27:

1. *Transmitter* és una especialització d'actuador que disposa de la capacitat de transmetre dades (objectes de tipus *Data*). Aquesta transmissió es realitzarà generant fenòmens d'un tipus específic que es correspongui amb el medi físic utilitzat com per exemple *ElectricalImpulse* o *RadioWave*.

2. *Receiver* són els sensors destinats a rebre aquestes transmissions, que sempre seran continus degut a que els fenòmens que transporten les dades es propaguen a una velocitat molt superior a la resta d'elements que participen en la simulació.

3. *NetworkInterface* representa una interfície de xarxa en sí, normalment composta per un receptor i un transmissor als quals fa de *façana*. La seva funció és alliberar a l'agent de les tasques de gestió d'aquests transmissors i receptors, amb un benefici addicional: la possibilitat de canviar el tipus d'interfície utilitzada sense modificar una sola línea del codi de l'agent.

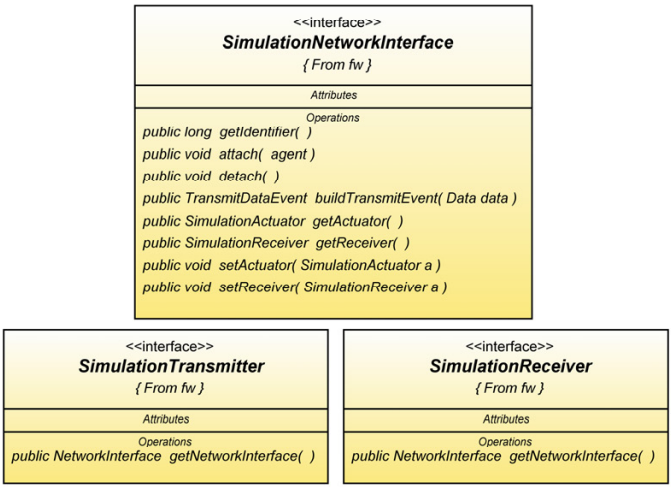


Figura 27: Interfícies per a la comunicació.

7.9.1 Wireless package

Com que la majoria d'aquestes xarxes utilitzen ràdio-transmissors per a comunicar-se, la plataforma proporciona un paquet (*package*) destinat al modelat d'aquest tipus de comunicacions.

Existeixen multitud de models per simular les comunicacions per radio, cadascun millor que els altres en certes situacions específiques. El *wireless package* defineix primer els elements d'aquests models en la seva forma més senzilla:

- *WirelessWave*
  És un fenomen que representa les ones de ràdio portadores dels paquets de dades i defineix les propietats bàsiques de que constarà qualsevol ona: la ubicació i l'instant de temps en què s'ha emès i les dades que transporta.

- *WirelessTransmitter*
  És qualsevol transmissor capaç de generar ones de ràdio. Per facilitar-ne la tasca implementa el mètode gestor de l'event d'actuar i delega només la creació de l'ona concreta (*generateWave(ActuateEvent)*) al transmissor específic.

- *WirelessReceiver*
  Representa els sensors destinats a rebre les ones de ràdio. Defineix un filtre que accepta els fenòmens *WirelessWave* sempre que arribin amb major potència que el llindar del receptor.

Partint d'aquesta base és possible implementar qualsevol model de comunicació per ràdio. En el mateix *package* s'inclouen els dos models que s'exposen a continuació.

Model radial El radial és el model més simple dels dos. Suposa que les ones s'expandeixen circularment a partir del seu punt d'origen sense pertorbacions que les afectin. L'abast de la ona és doncs una circumferència centrada en el punt d'emissió i limitada per la distància màxima a recórrer: el radi. Qualsevol receptor que es trobi a menys d'aquesta distància serà capaç de captar la ona i obtenir les dades que transporta.

Model de propagació en l'espai lliure El model de propagació en l'espai lliure és una aproximació més realista ja que té en compte les característiques del transmissor i el receptor. És vàlid aplicar-lo per modelar les comunicacions de ràdio on el l'emissor i el receptor disposen de línia de visió directa.

En l'espai lliure, la potència irradiada per una antena isotròpica ideal s'expandeix uniformement i sense pèrdua per la superfície d'una esfera envoltant l'antena. L'àrea d'una esfera de radi r és $4 \cdot \pi \cdot r^2$. La potència per unitat d'àrea w (en w/m^2) a distancia d d'un transmissor amb potència P_t i guany G_t és doncs:

$$w = \frac{P_t \cdot G_t}{4 \cdot \pi \cdot d^2}$$

La potència rebuda P_r per una antena receptora amb guany G_r és:

$$P_r = \frac{P_t \cdot G_t}{4 \cdot \pi \cdot d^2} \cdot A = (\frac{\lambda}{4 \cdot \pi \cdot d^2})^2 \cdot G_r \cdot P_t \cdot G_t$$

on A és l'àrea efectiva o obertura de l'antena receptora i $\lambda = c/f_p$ la longitud d'ona, que és igual a la velocitat de la llum c entre la freqüència de la senyal portadora (f_p).

7.10 Interfície gràfica

Junt amb la plataforma de simulació s'ofereix un *report* (secció 7.6.4) gràfic i interactiu que permet l'agregació i classificació de les dades resultants de les simulacions. S'implementa com a informe enlloc d'integrar-lo en el motor de simulació perquè d'aquesta manera resta totalment desacoblat, convertint-se en un component opcional, amb cicle de desenvolupament propi i que es pot activar o desactivar fàcilment. Permet així que l'usuari utilitzi aquesta interfície per analitzar profundament la simulació o realitzar demostracions i la desactivi quan vulgui obtenir els resultats més ràpidament o vulgui executar diverses simulacions en *batch*.

7.10.1 Visor de la simulació

En la imatge 28 es mostra una captura del visor de la simulació, consisteix en una finestra dividida en tres seccions: (1) una capçalera on es situen els botons de control i s'informa de l'instant de simulació mostrat, (2) un espai lateral per facilitar l'accés als elements de la simulació i mostrar-ne les seves propietats i (3) l'espai central on es mostrarà la informació més detallada.

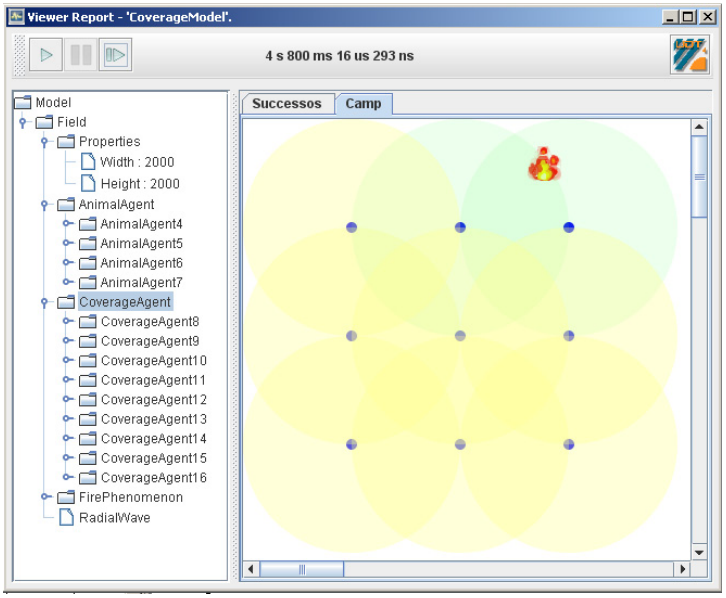


Figura 28: Visor de la simulació.

A la figura 29 s'exposa el diagrama de classes general d'aquest visor. Com tots els informes de simulació, el visor incorpora un *thread* propi on executar les seves tasques sense bloquejar el motor de simulació. Manipulant l'execució d'aquest fil es crea l'efecte d'estar controlant l'execució de la simulació tot i que en realitat només s'està afectant a la presentació dels

resultats. Les accions *Play*, *Pause* i *Step* (utilitzades per els botons de la capçalera) s'aprofiten d'aquest fet per oferir aquest control a l'usuari.

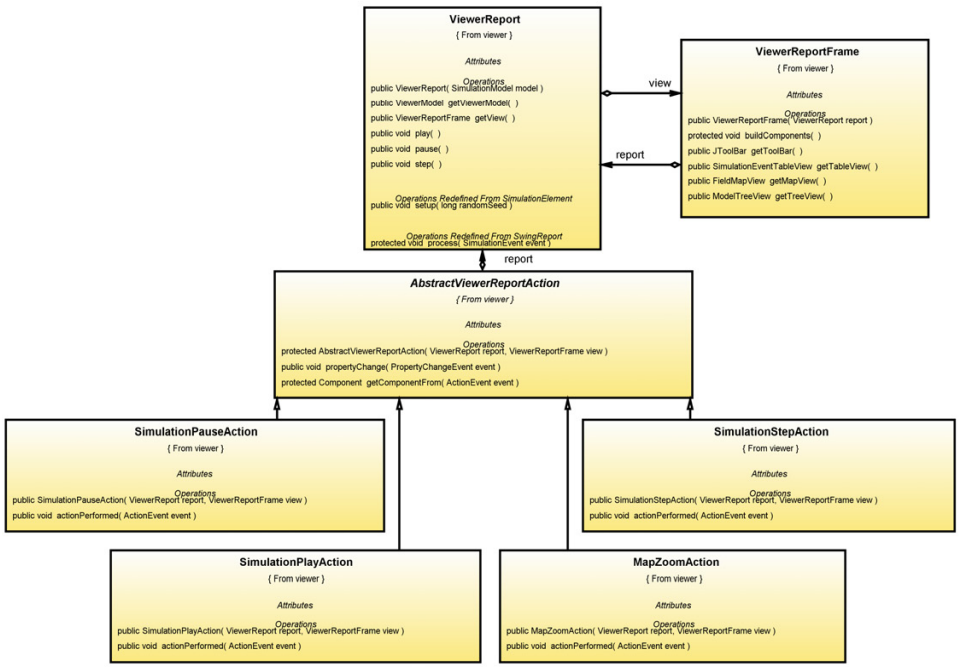


Figura 29: Diagrama de classes del visor.

Aquesta infraestructura es suportarà utilitzant una arquitectura de Model-Vista-Controlador (*MVC*), que es basa en la separació clara del model que conté la informació i les diferents formes de presentar-la.

7.10.2 Model del visor

El model del visor mostrat en el diagrama 30 és una petita estructura de classes dissenyada per mantenir una rèplica de la informació (i només de la informació) continguda a la simulació en un instant determinat. Es construeix utilitzant únicament els events que arriben al visor i per tant no manté cap lligam amb el motor de simulació.

Al contrari que el model del nucli de simulació, el del visor no està pensat perquè l'usuari l'estengui. Com que únicament ha d'emmagatzemar dades resulta molt més senzill ja que només ha de modelar les relacions entre els components base i utilitzar taules indexades clau-valor (*Hashtables*) per emmagatzemar les propietats o l'estat del component.

7.10.3 Vistes

Existeixen infinitat de maneres de mostrar les dades del model, cadascuna d'elles millor que les altres per a algun propòsit específic. Les vistes repre-

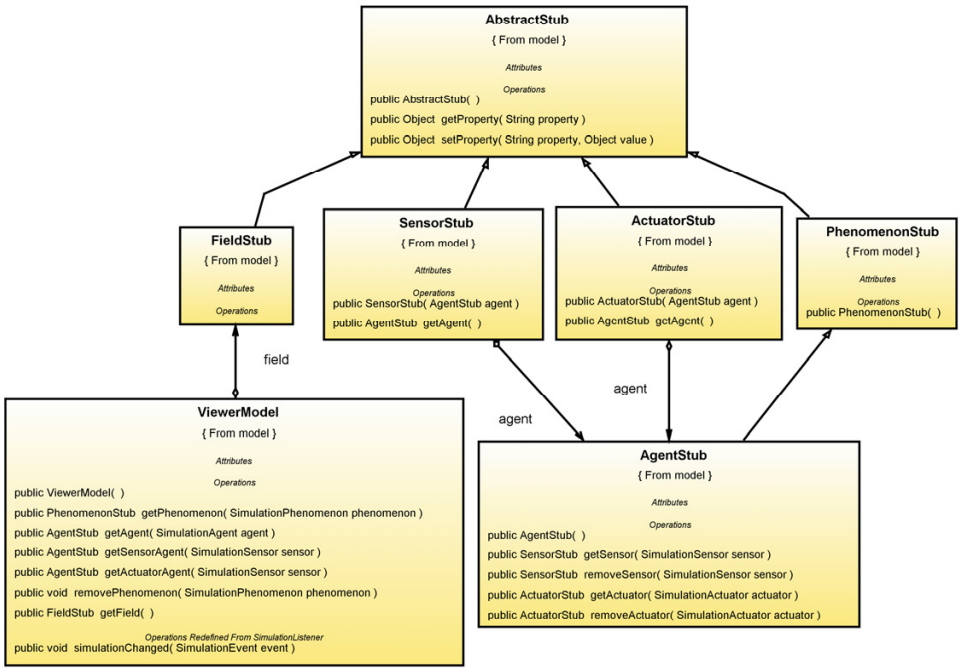


Figura 30: Diagrama de classes del model del visor.

senten aquestes agrupacions i classificacions que ofereixen a l'usuari diferents perspectives de la simulació per a diferents necessitats d'anàlisi.

Vista d'arbre La primera de les vistes que ofereix el visor (diagrama 31) presenta un arbre expansible format de tots els components que intervenen en la simulació, junt amb les seves propietats. S'utilitza un arbre (*Jtree*) enlloc d'una llista perquè permet agrupar els components segons el seu tipus i representar acuradament la relació jeràrquica existent entre ells ($Field \rightarrow Agent \rightarrow Sensor$).

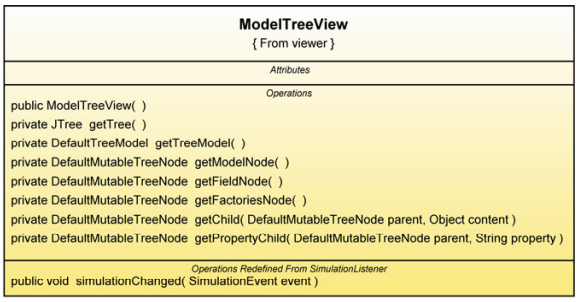
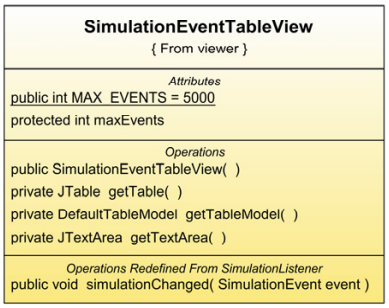


Figura 31: Vista d'arbre.

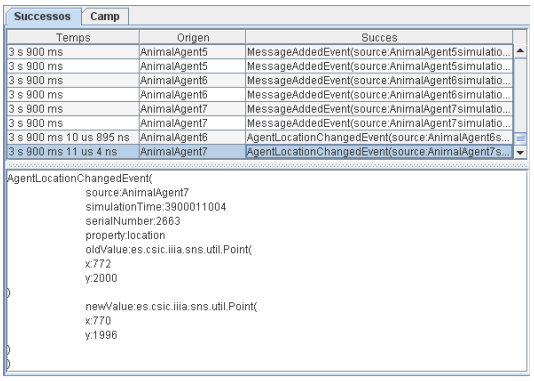
El major potencial d'aquesta vista resideix en la facilitat de localitzar un

element concret entre tots els de la simulació i visualitzar-ne l'estat actual o monitoritzar la seva evolució.

Vista d'events La vista d'events divideix el seu espai de visualització en dues àrees o sub-vistes. La primera d'elles mostra una taula que conté els darrers *maxEvents* (veure diagrama 32(a)) events produïts pel motor de simulació. De cadascun d'ells en mostra l'instant de temps en què s'ha donat, el component on s'ha generat i una representació en forma de cadena. En seleccionar un event de la taula, la segona sub-vista mostrarà la descripció complerta de l'event en qüestió.



(a) *SimulationEventTableView*.



(b) Screenshot de la vista.

Actualment aquesta vista resulta de gran utilitat per determinar l'ordre en el que es generen els events i els seus detalls. En un futur està prevista la possibilitat de filtrar els events per mostrar només aquells que afecten a un element concret, ajudant a determinar el seguit de successos que l'han portat a l'estat actual mostrat en la vista d'arbre.

Vista del camp de simulació La vista del camp de simulació es representa l'espai físic on succeeix la simulació, la ubicació dels diferents elements i algunes de les seves propietats en forma gràfica.

Per representar gràficament una simulació concreta és necessari que l'usuari especifiqui com s'han de dibuixar els components i les seves propietats més significatives. Aquesta descripció es fa mitjançant una factoria *MapComponentFactory* que crea un objecte de tipus *MapComponent* per a cada element de la simulació. En el diagrama 32 es pot veure que aquests objectes han d'implementar el mètode *paintComponent(Graphics2d)*, l'encarregat final de dibuixar el component. Combinant els diferents dibuixos de tots ells es formarà la vista del camp de la simulació.

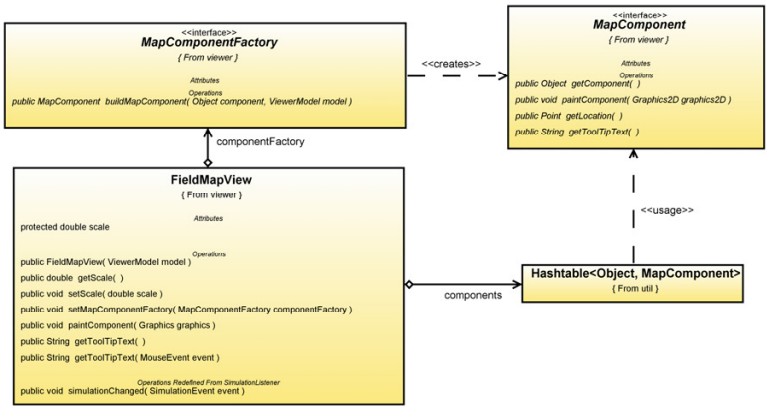


Figura 32: Diagrama de classes de la vista del camp.

8 Instàncies i exemples implementats

És imprescindible disposar d'algun exemple de simulació per a validar el funcionament de la plataforma, realitzar-ne demostracions i oferir exemples d'utilització als usuaris.

8.1 MiniSim

El primer dels exemples implementats ha servit com a eina de validació durant el desenvolupament de l'entorn de simulació. Utilitza tots els components bàsics que ofereix la plataforma però utilitzant-ne el mínim nombre possible per facilitar la comprovació dels resultats.

Presenta un objecte mòbil que avança a velocitat constant en una trajectòria rectilínia i dos agents amb radars d'abast limitat per intenten detectar-lo. En aquesta prova l'objecte només passarà prou aprop d'un dels dos radars com per ser detectat i l'agent corresponent notificarà a l'altre en fer-ho. En la captura de la figura 33 els agents es representen amb un punt blau i l'objecte mòbil amb un de vermell.

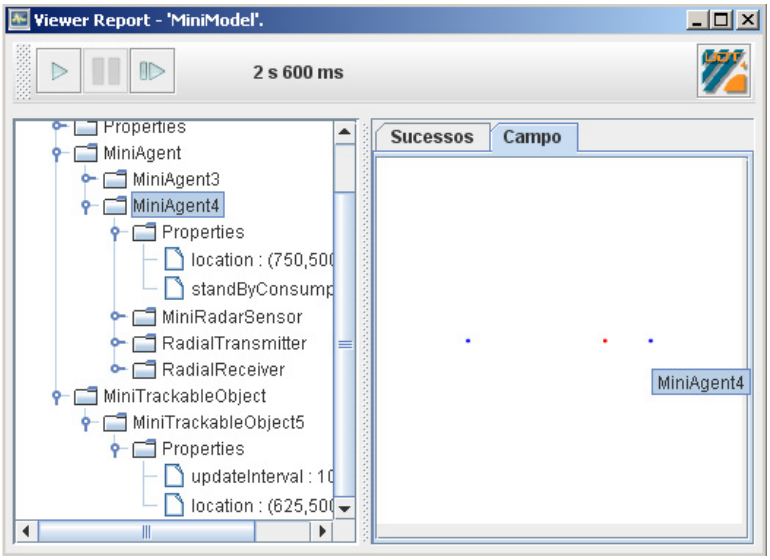


Figura 33: Simulació d'exemple *MiniSim*.

Aquesta simulació s'utilitza també com a cas de prova (*test case*) d'integració, aprofitant que qualsevol modificació de la plataforma no n'hauria d'alterar la seva execució. Es comprova el correcte funcionament del simulador comparant el resultat d'executar aquest exemple amb un registre complert i validat de la sortida esperada. Automatitzant aquest procés obtenim un cas de prova que fallarà si existeix alguna diferència de resultats, indicant que alguna modificació ha alterat el comportament del simulador.

Implementació

El primer dels components requerit per qualsevol simulació és el model. Estenent la classe *AbstractSimulationModel* se n'obté un que incorpora accions predefinides per a totes les funcionalitats, excepte dos mètodes que ens obliga a implementar: (1) el *buildAgents()* crearà els dos agents d'aquest exemple i (2) el *buildComponentFactories()* que crearà les factories de components.

El següent pas és crear una nova classe estenent-la *DefaultSimulationAgent* per modelar els agents. En el mètode *setup(long)* (construcció de l'agent) se li afegeixen el radar i la interfície de xarxa que utilitzarà, en aquest cas la *RadialNetworkInterface* explicada en la secció 7.9.1. Sobreescribint el mètode *init()* es configura l'estat inicial, el consum d'energia i s'encenen tots els sensors/actuadors perquè per defecte estan apagats i no podrien intervenir. Finalment s'ha de determinar a quins events han de reaccionar i implementar els mètodes *process* corresponents:

- *process(MeasureEvent)*, enviat per el radar de l'agent en detectar un objecte. L'agent extreurà les dades de la mesura i les enviarà en *broadcast* a través de la seva interfície de xarxa.
- *process(DataReceivedEvent)* que es cridarà en rebre la notificació de detecció enviada per l'altre agent i mostrarà un missatge a la consola.

El radar és un sensor discret i per tant s'utilitza la classe *DefaultDiscreteSensor* com a base per modelar-lo. Aquesta classe obliga a implementar el mètode *sense(List)* que rep per paràmetre una llista dels objectes detectats i en aquest cas n'extreurà la posició i la remetrà a l'agent en forma de mesura. La capacitat sensitiva es defineix creant una nova classe *SensorFilter* amb el mètode *accept(Phenomenon)* que només retornarà *true* si el fenomen en qüestió és un objecte mòbil i cau dins el cercle de detecció determinat per la propietat *sensingRadius*.

Es podria afegir l'objecte mòbil directament des del model però com que la finalitat d'aquesta simulació és provar tots els components s'implementarà una factoria per fer-ho. Estenent la classe *DefaultComponentFactory* només hem de sobreescriure el mètode *init()* perquè crei l'objecte i l'afegeixi a la simulació. Aquesta factoria es crearà i activarà des del mètode *buildComponentFactories* del model.

L'últim dels components que d'aquesta simulació és l'objecte mòbil, el seu únic fenomen. S'implementarà a partir de la classe *DefaultPhenomenon*, afegint-li un model de mobilitat. Es prepara un nou event *UpdateObjectEvent* perquè es produeixi cada *updateInterval* nanosegons i s'implementa el mètode *process(UpdateObjectEvent)* perquè actualitzi la posició de l'objecte.

Amb això es disposa ja d'un model totalment funcional que es pot executar i visualitzar-ne la simulació inclús amb l'entorn gràfic, excepte per un petit detall: l'objecte mòbil no apareix en el camp de simulació. Els agents

apareixen directament perquè sempre tenen posició dins el camp però els fenòmens no i per tant la plataforma no pot assumir que siguin dibuixables. La sol·lució passa per crear una factoria de components, estenent la classe *DefaultMapComponentFactory* i sobreescrivint-ne el mètode *buildMapComponent* perquè retorni un nou objecte *DefaultPhenomenonMapComponent* quan el component a pintar sigui un objecte mòbil. En la figura 34 es mostra el diagrama complert de totes les classes que intervenen en aquesta simulació d'exemple i com es relacionen entre elles.

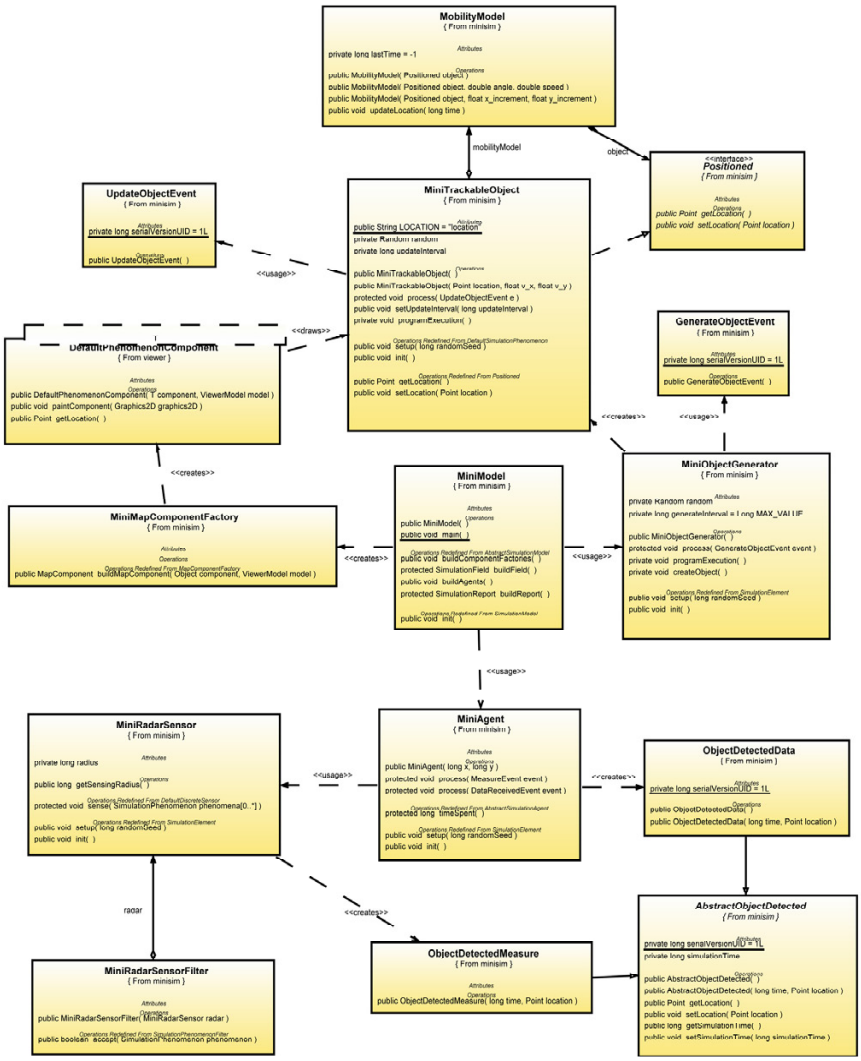


Figura 34: Diagrama de classes de l'exemple *MiniSim*.

8.2 Coverage Problem

El segon exemple s'enfoca a l'estudi d'un problema ben definit en la literatura: la cobertura d'una àrea considerant el consum d'energia [18], aplicat al problema de detecció de focs. S'han modelat tots els elements que intervenen en aquest domini, les seves funcionalitats i el dinamisme tant en la xarxa de sensors com en el camp. Els algorismes dels agents per optimitzar el consum d'energia o maximitzar el temps de vida de la xarxa queden fora de l'abast d'aquest projecte i per tant no s'han implementat. En la figura 35 es presenta una captura de pantalla de la simulació resultant.

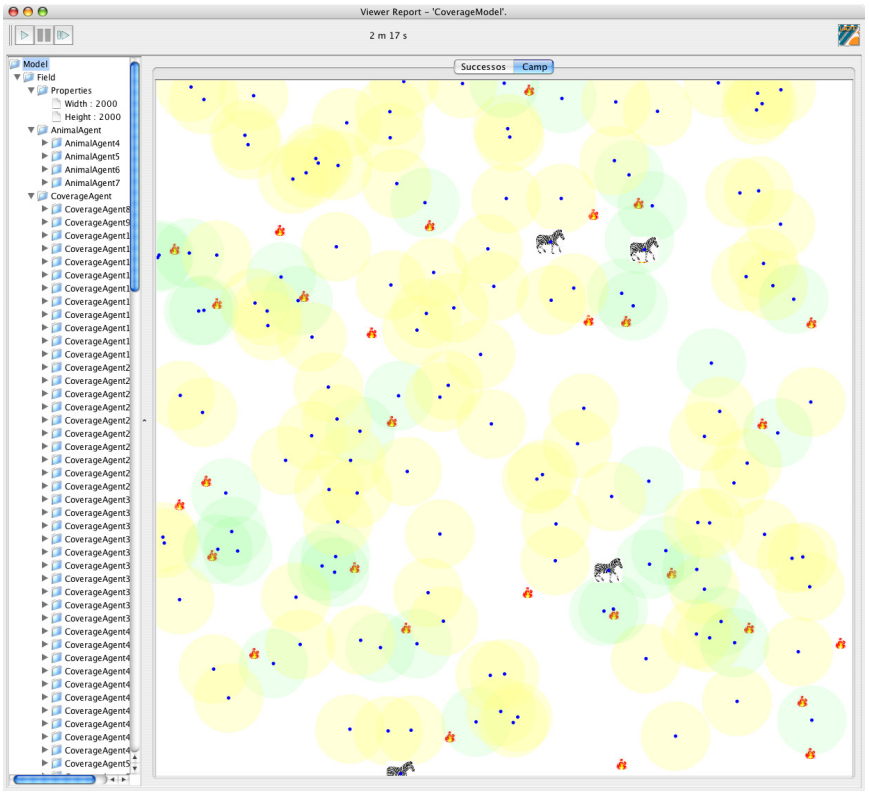


Figura 35: Simulació d'exemple *CoverageProblem*.

Els passos seguits per a la implementació d'aquest exemple són molt similars als de l'exemple anterior. En aquesta ocasió però s'han complicat alguns dels components utilitzats. A continuació es presenta una relació de quins són, la funció que realitzen i com s'han implementat.

Model. Estenent del model base, s'implementen els mètodes de creació dels agents inicials (*buildAgents*) i les factories de components (*BuildComponentFactories*). El problema indica que la posició inicial dels agents ha de ser aleatoria. Aquest requisit es satisfà utilitzant un dels distribuidors físics que integra la plataforma: *RandomDistribution*. La classe rep com

a paràmetres l'àrea en la que ha de distribuir un conjunt de fenòmens o agents i el tipus d'aquests, procedint a repartir-los aleatoriament. És millor fer servir una distribució enlloc de posicionar cada agent al.leatòriament ja que d'aquesta manera si canviés el requisit només s'hauria de modificar la distribució utilitzada.

Fenomen foc. Els incendis es representen mitjançant fenòmens foc, que no necessiten cap funció especial excepte la seva extinció. En la inicialització, el fenomen s'envia un event *FireExtinctionEvent* a si mateix amb un retard d'entre 1 i 5 segons. En rebre'l, ell mateix s'elimina de la simulació.

Factoria de focs. Com que els incendis són temporals, necessitem un actor que s'encarregui de crear-ne de nous durant la simulació. La *FirePhenomenonFactory* realitza aquesta tasca, afegint n focs cada *interval* segons.

Agent de cobertura. Els agents de cobertura representen els nodes de la xarxa de sensors. S'encarreguen de decidir la configuració adient per al seu sensor de cobertura i de la comunicació amb els altres agents. L'algorisme per optimitzar aquestes decisions i configuració no està implementat, però sí la comunicació de les deteccions als agents propers utilitzant una interfície

RadialNetworkInterface (explicada en la secció 7.9.1). Els agents utilizen una bateria infinita però s'especifica el consum dels components perquè es pugui controlar el seu consum energètic.

Sensor de cobertura. El sensor de cobertura és un dispositiu capaç de detectar incendis en un perímetre circular. Pren mesures cada cert interval de temps i per tant s'implementa estenent el *DefaultDiscreteSensor*. En detectar algun incendi envia una mesura a l'agent que especifica l'instant de temps i la ubicació de la detecció.

Agents animal. En un desplegament real realitzat al bosc hi hauria una sèrie d'elements en l'entorn capaços de destruïr els sensors: fenòmens meteorològics, animals que els trepitgen o se'ls mengen i altres. Per representar-los s'introdueix aquest nou tipus d'agent. Es mou de forma aleatoria per el camp i destrueix qualsevol agent de cobertura que es troba. Es modela com a agent perquè disposi d'un sensor que detecti els agents de cobertura. En rebre una notificació de detecció (*AgentsDetectectedEvent*) procedeix a la seva destrucció mitjançant el mètode *die()* que aquests exposen.

Sensors d'agents. Són sensors discrets que actuen igual que els sensors de cobertura però detectant fenòmens de tipus *CoverageAgent* enlloc de focs. En detectar un agent de cobertura notifiquen el seu *agent animal* perquè aquest pugui destruir-los.

MapComponents. Per millorar la visualització de la simulació s'han implementat també tres *map components*: (1) els sensors de cobertura es dibuixen amb una circumferència que mostra la seva zona de percepció, (2) els agents animal es representen mitjançant la imatge d'una zebra i (3) els fenomens foc escenificats amb la imatge d'un foc.

9 Resultats i conclusions

El resultat d'aquest projecte és una plataforma de simulació basada en events per a xarxes de sensors. En la secció 4 es realitza un estudi dels simuladors existents i al presentar els requeriments (secció 3) queda palesa la necessitat d'un nou simulador de propòsit general per a estudiar les xarxes de sensors com a sistemes multi-agent, podent-se utilitzar per simular un gran nombre de dominis d'aplicació.

La plataforma proporciona una estructura de simulació i un conjunt de components (agents, sensors, actuadors, fenòmens, etc) que faciliten el modelat de les xarxes de sensors com a sistemes multi-agent. Aquests components proporcionen la funcionalitat comuna en totes les xarxes de sensors però mantenen un elevat nivell d'abstracció per poder ser utilitzats en qualsevol domini d'aplicació.

El control del temps de simulació es realitza seguint una política de planificació d'events (*event scheduling*) lleugerament modificada. Les modificacions introduïdes permeten satisfer tres dels requeriments llistats en la secció 3: (1) controlar el temps de procés dels components, (2) modelar el consum d'energia de cada element i (3) assegurar la repetibilitat de les simulacions.

Tant els algorismes com les estructures de dades utilitzades al nucli de simulació han estat estudiades i escollides per maximitzar el rendiment del simulador. No s'han realitzat proves comparatives amb altres simuladors, però amb els exemples implementats es demostra capaç de gestionar xarxes de més d'un miler de nodes amb un temps real inferior al temps de simulació.

Pel que fa a la visualització i obtenció de resultats el sistema proporciona una interfície gràfica que permet l'anàlisi i visualització de la simulació durant una execució. La implementació d'altres reports tals com per a generar estadístiques sobre el consum d'energia o el nombre de missatges generats no s'ha arribat a implementar i es deixa com una línia futura per a la plataforma.

La plataforma s'ha alliberat sota la llicència *GPL* perquè tota la comunitat científica se'n pugui beneficiar. Per introduir els usuaris a l'ús del simulador s'aporten dos exemples de simulació en diferents dominis que cobreixen les característiques principals de qualsevol model de xarxa de sensors. La present memòria serveix també de referència per comprendre l'arquitectura utilitzada. A més, el codi ha estat degudament documentat en la seva totalitat i s'acompanya d'un conjunt de testos automatitzats (test cases).

9.1 Línies futures

A continuació es detallen les extensions que identifico més rellevants per a realizar en el simulador:

- **Protocols d'encaminament.** Les comunicacions juguen un paper

clau en l'estudi de les xarxes de sensors. La plataforma ofereix una gran flexibilitat que permet implementar qualsevol protocol possible, però no ofereix la implementació de cap dels protocols estàndard. Desenvolupar els protocols més utilitzats en xarxes de sensors facilitaria la utilització del simulador i per tant milloraria enormement la seva acceptació.

- **Parametrització del model i execució de bateries de tests.** Un altre aspecte millorable és la configuració de les simulacions. Actualment els paràmetres d'una execució de simulació es defineixen directament en el codi, obligant a recompilar cada vegada que es volen canviar. Un sistema de definició de paràmetres facilitaria, a més, l'execució de bateries de proves, facilitant enormement l'anàlisi de l'impacte dels paràmetres sobre els resultats.

- **Extensió de les funcionalitats i disseny de la interfície gràfica** L'objectiu d'aquest projecte era proporcionar la primera versió d'un simulador per a xarxes de sensors en sistemes multi-agent i no s'ha centrat en la interfície gràfica d'aquest. Per això la interfície gràfica proporcionada presenta només les funcionalitats bàsiques i té un disseny millorable: es redibuixa la pantalla més del necessari, la vista dels events no permet cap tipus de filtratge.... A més, l'eficiència en general és *molt* inferior a la del nucli del simulador, on s'ha estudiat la naturalesa de cada algorisme utilitzat per maximitzar el rendiment.

- **Paral·lelització del simulador** La naturalesa distribuïda de les xarxes de sensors convida a pensar en una possible paral·lelització del simulador. Els primers passos en aquest sentit estan donats: els components no comparteixen estat (poden residir fins i tot en màquines diferents) i tots els events són serializables (poden ser transmesos per xarxa). El major repte que queda pendent en aquesta àrea es refereix al requisit de repetibilitat. En introduir diverses màquines i elements de xarxa, s'afegeix també una font de successos irrepetibles (retards en les transmissions, diferència de temps de procés i altres).

- **Suport per l'usuari: Manuals, tutorials i exemples.** Per a millorar l'acollida a la comunitat seria molt desitjable disposar també de més material destinat als potencials usuaris: un manual complert, tutorials, més exemples de simulacions i un repositori de components ja predefinits.

Referències

[1] M. Vinyals, J.A. Rodriguez-Aguilar, and J. Cerquides. A survey on sensor networks from a multi-agent perspective, 2008.

[2] C.Y.E.E. CHONG and S.P. KUMAR. Sensor Networks: Evolution, Opportunities, and Challenges. *PROCEEDINGS OF THE IEEE*, 91(8):1247, 2003.

[3] D. CRULLER, D. ESTRIN, and M. SRIVASTAVA. Overview of sensor networks. *Computer(Long Beach, CA)*, 37(8):41–49, 2004.

[4] Katia P. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998.

[5] DARPA. The network simulator - ns2.

[6] Jaroslav Kačer. J-Sim – a Java-based tool for discrete simulations. Master's thesis, University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering, Univerzitní 22, 30614 Plzeň, Czech Republic, May 2001.

[7] *Distributed Sensor Networks: A Multiagent Perspective*, chapter The Radsim Simulator. Defense Advanced Reserarch Projects (DARPA), 2003.

[8] M. J. Pflug L. Zhu G. Chen, J. Branch and B. Szymanski. Sense: A sensor network simulator. in advances in pervasive computing and networking, Springer, 2004.

[9] V. Kunchakarra SS. Iyengar R.Kannan C.Mallanda, A.Sun and A.Durresi. Simulating wireless sensor networks with omnet++.

[10] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking*, pages 56–67, 2000.

[11] J.C.; Lu-Chuan Kung; Ning Li; Honghai Zhang; Wei-Peng Chen; Hung-Ying Tyan; Hyuk Lim Sobeih, A.; Hou. J-sim: a simulation and emulation environment for wireless sensor networks. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 13(4):104–119, Aug. 2006.

[12] Repast. Repast organization for architecture and development. http://repast.sourceforge.net, 2004.

[13] ECMA. *ECMA-334: C# Language Specification*. ECMA (European Association for Standardizing Information and Communication Systems), Geneva, Switzerland, second edition, December 2002.

[14] Lutz Prechelt. An empirical comparison of c, c++, java, perl, python, rexx, and tcl for a search/string-processing program.

[15] Geoffrey Phipps. Comparing observed bug and productivity rates for java and c++. *Softw. Pract. Exper.*, 29(4):345–358, 1999.

[16] Apache Ant. Apache ant.
http://ant.apache.org, 2007.

[17] Sun Microsystems. Netbeans, 2007.

[18] M. Cardei and J. Wu. Energy-efficient coverage problems in wireless ad-hoc sensor networks. *Computer Communications*, 29(4):413–420, 2006.

Marc Pujol i González
4 de febrer de 2008

Aquest projecte descriu una plataforma de simulació per a xarxes de sensors des de la perspectiva dels sistemes multi-agents. La plataforma s'ha dissenyat per facilitar la simulació de diferents aplicacions concretes de xarxes de sensors. A més, s'ha entregat com a artefacte del projecte IEA (Institucions Electròniques Autònomes, TIN2006-15662-C02-0) de l'IIIA-CSIC. Dins l'entorn de l'IEA, aquesta és l'eina que aporta les capacitats de simulació per donar suport al disseny d'algorismes adaptatius per a xarxes de sensors.

Este proyecto describe una plataforma de simulación para redes de sensores des de la perspectiva de los sistemas multi-agentes. La plataforma se ha diseñado para facilitar la simulación de diferentes aplicaciones concretas de redes de sensores. Además, se ha entregado como artefacto del proyecto IEA (Institucions Electròniques Autònomes, TIN2006-15662-C02-0) del IIIA-CSIC. Dentro del entorno del IEA, esta es la herramienta que aporta las capacidades de simulación para dar soporte al diseño de algoritmos adaptativos para redes de sensores.

This project describes a multi-agent based simulation framework for sensor networks. It has been conceived to ease the simulation of a wide range of sensor network applications. Furthermore, the framework has been delivered as a result of the ongoing IEA (Institucions Electròniques Autònomes, TIN2006-15662-C02-0) project at IIIA-CSIC. In the realm of IEA, this is the tool that provides simulation facilities to support the design of self-organising algorithms for sensor networks.