

Università degli Studi di Salerno
Facoltà di Scienze Matematiche Fisiche e Naturali

Tesi di Laurea di I livello in
Informatica

Documentazione assistita di ABMs (Agent-Based models)

Relatore
Prof. Vittorio Scarano

Candidato
Simone Romano

Anno Accademico 2013-2014

Alla mia famiglia che mi ha sostenuto,
a lei che ha colorato i miei obiettivi

Questa tesi è stata sviluppata in 

Indice

1 Introduzione **1**
- 1.1 Panoramica . . . 1
- 1.2 Documentazione ABMs . . . 1
- 1.3 MAD (Mason Assisted Documentation) . . . 2
- 1.4 Piano dell'opera . . . 2

2 Modelli di simulazione basati sugli agenti **4**
- 2.1 ABMs (Agent-Based Models) . . . 5
  - 2.1.1 Contesto di applicazione . . . 5
  - 2.1.2 Vantaggi nell'utilizzo di ABMs . . . 5
  - 2.1.3 ABMs in informatica . . . 6
- 2.2 MASON . . . 7
  - 2.2.1 Architettura . . . 7
  - 2.2.2 Simulazioni in MASON . . . 7

3 Doxygen **10**
- 3.1 Overview . . . 11
  - 3.1.1 Note per l'uso . . . 11
  - 3.1.2 Modules . . . 16
- 3.2 MASON e Doxygen: l'idea iniziale . . . 17

4 MAD(Mason Assisted Documentation)- prima versione **18**
- 4.1 Struttura ricorrente in MASON . . . 19
  - 4.1.1 SimState . . . 19
  - 4.1.2 Steppable . . . 19
- 4.2 Semantic analysis of code . . . 20
- 4.3 MAD (Mason Assisted Documentation) . . . 20
  - 4.3.1 Critiche . . . 21
  - 4.3.2 Lati positivi . . . 23

5 ODD protocol e MASON **25**
5.1 ODD protocol . . . 26
5.2 Elementi del protocollo ODD . . . 27
5.2.1 Purpose . . . 27
5.2.2 Entities, state variables, and scales . . . 27
5.2.3 Process overview and scheduling . . . 28
5.2.4 Design concepts . . . 28
5.2.5 Initialization . . . 29
5.2.6 Input data . . . 29
5.2.7 Submodels . . . 29
5.3 ODD e MASON . . . 30
5.4 ODD e Doxygen . . . 33

6 MAD- versione corrente **36**
6.1 Seconda versione . . . 37
6.1.1 Parsing del codice . . . 37
6.1.2 Visitor pattern . . . 37
6.1.3 Iterabilità . . . 37
6.1.4 Process overview and scheduling . . . 38
6.1.5 Color feature . . . 38
6.1.6 Files di configurazione . . . 39
6.2 Terza versione - Output type . . . 40

7 Implementazione **45**
7.1 Librerie utilizzate . . . 46
7.1.1 AST(Abstract syntax tree) . . . 46
7.1.2 iText . . . 46
7.2 Struttura . . . 47
7.2.1 Analizer . . . 48
7.2.2 Control . . . 52
7.2.3 Wizard . . . 52
7.2.4 ODD . . . 52
7.2.5 Visitor . . . 52
7.3 Sviluppi futuri . . . 54

8 Manuale d'uso **55**
8.1 Manuale utente MAD . . . 56
8.1.1 Download . . . 56
8.1.2 Utilizzo . . . 56

9 Conclusioni **78**

Bibliografia **79**

A Appendice **80**
A.1 Listati 80
A.1.1 L'oggetto ODD 80
A.1.2 La classe FieldInitializerVisitor 87
A.1.3 La classe JavaDocVisitor 89
A.1.4 CodeVisitor 90
A.1.5 StartStepMethodVisitor 93
A.1.6 RecoursiveCodeVisitor 97
A.2 Esempio di utilizzo di MAD 98

Capitolo 1

Introduzione

1.1 Panoramica

I sistemi di simulazione basati sugli agenti (ABMs - Agent-Based Models) rappresentano una tecnica abbastanza recente nel campo dell'informatica (primi studi negli anni '80; diffusione anni '90) per modellare sistemi complessi che abbracciano svariati campi di applicazione (economia, sociologia, chimica, fisica delle particelle ecc.). L'interazione tra l'agente e il modello è autonoma; l'agente è quindi dotato di una propria intelligenza. Scopo dei sistemi basati sugli agenti è determinare l'evoluzione di un sistema al fine di prevederne l'organizzazione risultante. Diverse sono gli strumenti utilizzabili per la creazione di ABMs. Durante il nostro lavoro ci siamo occupati della documentazione assistita di modelli di simulazione multi agente scritti in linguaggio java e che utilizzano la libreria *MASON*.

1.2 Documentazione ABMs

E' di particolare importanza produrre una valida documentazione di ABMs che talvolta risulta non rispondere alle informazioni ricercate. Quello che andremo informazioni relative al modello di simulazione descritto dalla applicazione. Per raggiungere tale scopo è risultato fondamentale capire effettivamente come un ABMs va documentato. Il protocollo ODD (Overview, Design concepts and details) [1] pubblicato nel 2006 si propone come riferimento per la descrizione di modelli basati sugli agenti. La problematica che questo protocollo si propone di affrontare è fornire uno standard per documentare ABMs.

1.3 MAD (Mason Assisted Documentation)

I programmi analizzati durante il nostro lavoro (scritti in linguaggio *java* con libreria *MASON* [3]) sono la concretizzazione (proposta da esperti informatici) di problemi studiati ed impostati da esperti in scienze sociali. L'esperto non informatico trova spesso difficoltà nel ricavare le informazioni legate al modello all'interno di un contesto che non gli appartiene(programma informatico). D'altra parte l'informatico deve essere in grado di fornire informazioni che interessano all'esperto in scienze sociali. Per creare un ponte tra queste due figure e supportare quindi lo scambio di informazioni, il nostro lavoro propone MAD (Mason Assisted Documentation), un plug-in per la piattaforma *Eclipse* che offre un supporto alla creazione di documentazione per programmi che utilizzano la libreria *MASON*. Una volta realizzato il programma di simulazione, l'utente può lanciare la procedura assistita che mira a creare una documentazione strutturata secondo lo standard ODD [1]. Molte delle informazioni richieste sono generate in maniera automatica; l'utente può interagire aggiungendo inoformazioni. Il plugin offre diverse modalità di output. In particolare la modalità *Doxygen* [2] che collabora con il tool *Doxygen* per generare html e latex. L'utilizzo di *Doxygen* aggiunge informazioni aggiuntive alla documentazione.

1.4 Piano dell'opera

Per iniziare verranno introdotti i modelli di simulazione basati sugli agenti, evidenziando il ruolo dell'informatica. In particolare verrà descritta la libreria MASON che consente di creare simulazioni multi-agente in java. Nel capitolo 3 è introdotto Doxygen, un tool che consente di generare documentazione automatica del codice ed evidenzia le motivazioni che ci hanno spinto al suo utilizzo per il raggiungimento del nostro obiettivo. Il capitolo 4 introduce la prima versione di MAD(Mason Assisted Documentation), il prodotto realizzato durante questo lavoro di tesi. Verranno analizzati i lati positivi e le critiche a cui la prima versione si prestava. In particolare verrà discusso il problema della struttura della documentazion generata da MAD nella versione iniziale. Analizzeremo quindi nel capitolo 5 uno standard per la documentazione di ABMs, il protocollo ODD [1]. Per ognuna delle sezioni del protocollo ODD sono state individuate le relazioni con la libreria MASON. A questo punto il capitolo 6 presenta la versione attuale di MAD, descrivendone i punti di forza. Il capitolo 7 fornisce dettagli implementativi, descrivendo le librerie utilizzate e analizzando ciascuno dei package dell'ap-

plicazione. Infine il capitolo 8 fornisce informazioni per il download e l'utilizzo del plugin, mostrando un esempio di utilizzo sul programma *AntsForage* fornito con la libreria MASON. In A.2 è mostrato l'output dell'esempio.

Capitolo 2

Modelli di simulazione basati sugli agenti

Introduzione

Per poter intraprendere un discorso relativo alla documentazione assistita di modelli di simulazione basati sugli agenti si rende necessario una introduzione a tali modelli. E' scopo di questo capitolo illustrare quelle che sono le caratteristiche degli ABMs. Vedremo in quali contesti tali modelli risultano particolarmente efficienti e quali sono i vantaggi che offrono. Sarà illustrato il rapporto che tali modelli hanno con l'informatica ed, in particolare, sarà descritta in la libreria *MASON* [3]. Vedremo quindi come tale libreria supporta la fase di creazione di ABMs in java.

2.1 ABMs (Agent-Based Models)

Il problema di come modellare le complessità presenti in molti sistemi reali è stato un problema su cui esperti sociali si sono imbattuti per diverso tempo. Uno degli approcci per far fronte a tale problema è utilizzare modelli basati sugli agenti. I sistemi multi-agente costituiscono un'interessante tipologia di modellazione di società, ed hanno a questo riguardo vasti campi d'applicazione. Un sistema multi-agente o (sistema ad agenti multipli) è un insieme di agenti situati in un certo ambiente ed interagenti tra loro mediante una opportuna organizzazione. Un agente è cioè un'entità caratterizzata dal fatto di essere, almeno parzialmente, autonoma, sia essa un programma informatico, un essere umano, e così via. I sistemi multi-agente vengono utilizzati per simulare le interazioni esistenti tra agenti autonomi. Si cerca di determinare l'evoluzione del sistema al fine di prevederne l'organizzazione risultante. La simulazione viene utilizzata per lo svolgimento di esperimenti su un sistema reale sarebbe impossibile o impraticabile: ad esempio, a causa del costo elevato di prototipazione e test, o perché la fragilità del sistema non supporterà numerose prove, o per la durata dell'esperimento in tempo reale è impraticabile.

2.1.1 Contesto di applicazione

La modellazione con sistemi basati sugli agenti si rende particolarmente adottabile in presenza delle seguenti condizioni:

- Il problema che si desidera esplorare ha una rappresentazione naturale costituito da agenti interagenti.
- Il problema può essere descritto in termini di singoli comportamenti dei partecipanti
- Gli agenti sono in relazione tra di loro o con l'ambiente in cui sono agiscono e sono dotati di intelligenza propria

2.1.2 Vantaggi nell'utilizzo di ABMs

Utilizzare simulazioni basate sugli agenti invece di esperimenti fisici porta una serie di vantaggi. Di seguito ne riportiamo alcuni:

- Tempo: alcuni esperimenti possono richiedere tempi di risposta lunghi se applicati in contesti reali. Gli stessi esperimenti spesso offrono risultati in pochi minuti se simulati

- Valori ottenuti: modelli di simulazione possono essere utilizzati in ambito economico. Ovviamente poter individuare ad esempio quali investimenti siano fruttuosi per un'azienda è un grande vantaggio. Applicare direttamente alcune strategie di marketing nella vita reale potrebbe portare ad un fallimento qualora le strategie si dimostrassero sbagliate. Con la simulazione tale problema non sussiste.

- Ripetibilità: in alcuni contesti risulta fondamentale ripetere alcuni esperimenti variando ad esempio alcuni parametri. Con le simulazioni questo risulta poco faticoso. Inoltre per esperimenti particolarmente lunghi potrebbe diventare molto costoso (in termini temporali) ripetere più volte l'esperimento.

- Versatilità: la simulazione ci permette di risolvere problemi in qualsiasi settore: produzione, logistica, finanza, salute, e molti altri. In ogni caso, il modello simula la vita reale e consente un'ampia gamma di esperimenti senza impatto a oggetti reali.

2.1.3 ABMs in informatica

Nel contesto informatico i modelli di simulazione basati sugli agenti rappresentano una branca relativamente nuova. I primi studi risalgono agli anni '80 mentre la diffusione di tali modelli risale alla metà degli anni '90. Diversi sono gli strumenti informatici che consentono di creare simulazioni. Di seguito ne citiamo alcuni:

- *Anylogic*: software rilasciato nel 2000 che consente di sviluppare diversi paradigmi di simulazione (da qui *Anylogic*). Questo software permette di creare simulazioni agevolmente tramite una interfaccia grafica. Inoltre può essere aggiunto codice java per estendere il modello.

- *Netlogo*: NetLogo è un ambiente di modellazione programmabile per simulare fenomeni naturali e sociali. E 'stato scritto da Uri Wilensky nel 1999 ed è in continuo sviluppo.

- MASON: Mason è una libreria java che supporta lo sviluppo di simulazioni basate sugli agenti.

2.2 MASON

Fondamentale per il nostro lavoro è stato comprendere le caratteristiche della libreria MASON. MASON è una libreria java che supporta la creazione di simulazioni multi-agente in java. E' progettata per supportare un gran numero di agenti in maniera efficiente su una sola macchina.

2.2.1 Architettura

L'architettura della libreria MASON può essere suddivisa fondamentalmente in 2 categorie:

- **model**
- **visualization**

Model

Il modello di simulazione in MASON è interamente incapsulato in un oggetto chiamato *SimState*. Tale oggetto rappresenta quindi l'ambiente in cui la simulazione si svolge. Il concetto del tempo è concretizzato con un *discrete event schedule* su cui vengono aggiunti i vari agenti da schedulare ad ogni step della simulazione. Solitamente l'oggetto istanza della classe *SimState* contiene quelli che sono chiamati *fields*, ovvero delle strutture dati (matrici bi-dimensionali/tri-dimensionali) su cui gli agenti si spostano. Per motivi di efficienza la libreria offre un generatore di numeri casuali (utilizza l'algoritmo *Marsenne Twister*).

Visualization

MASON garantisce una netta separazione tra il layer *model* ed il layer di visualizzazione (vedi figura 2.1). Questo consente di creare simulazioni con o senza interfaccia grafica; quest'ultima può essere pensata anche dopo aver completato la applicazione. Possiamo creare interfacce 2D o 3D.

2.2.2 Simulazioni in MASON

Una tipica simulazione che utilizza la libreria *MASON* è composta da 3 classi:

- Agente: classe che implementa l'interfaccia *Steppable* che espone il metodo *step()*. In questo metodo verranno inserite le operazioni che l'agente dovrà svolgere an ogni singolo step della simulazione.

- Ambiente: classe che estende la classe *sim.engine.SimState*. Tale classe conterrà il metodo *start()* che principalmente si occuperà di istruire lo *schedule* su quali agenti scehdulare ad ogni passo.
- GUI: una console grafica, che estende *GUIState* e fornisce tutti gli strumenti utili per creare una interfaccia grafica di supporto all'esecuzione della simulazione.

Gli agenti condividono variabili d'istanza pubbliche con l'ambiente della simulazione. Questa condivisione è concretizzata nel metodo *step()* di ciascun agente. Tale metodo ha infatti come parametro un oggetto che estende la classe *SimState*.

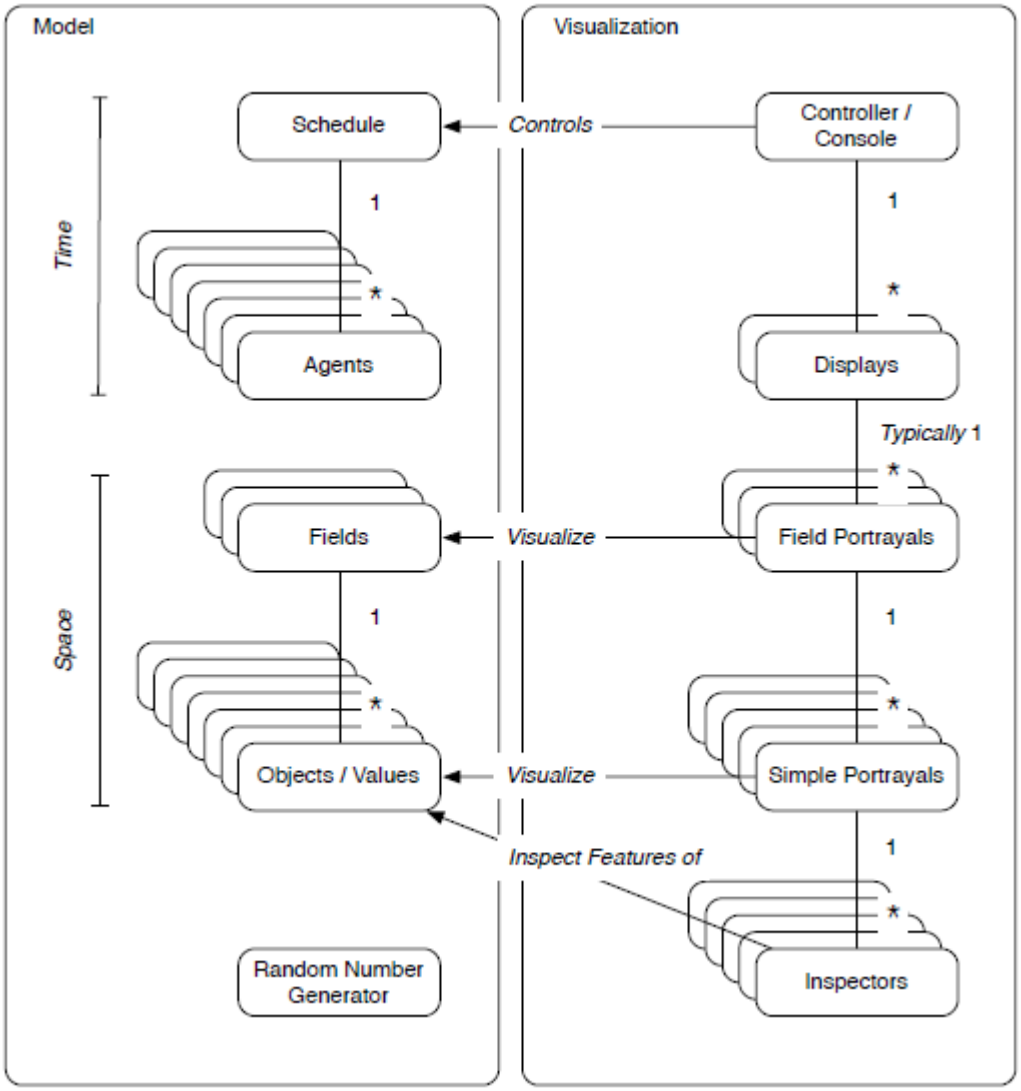


Figura 2.1: Architettura della libreria MASON

Capitolo 3

Doxygen

Introduzione

Il nostro lavoro ha come obiettivo quello di documentare in maniera assistita modelli si simulazione multi agente scritti in java che utilizzano la libreria *MASON*. *Doxygen* [2] è un tool open source che consente di generare documentazione partendo da codice opportunamente commentato. L'obiettivo iniziale è stato quello di utilizzare al massimo le potenzialità di questo tool per cercare di estrarre informazioni non solamente strutturali (inerenti quindi alla struttura del progetto, ai package, alle classi e alle loro relazioni) ma informazioni che riguardassero il modello di simulazione descritto dalla applicazione analizzata. In questo capitolo saranno esaminate alcune delle potenzialità di *Doxygen* e vedremo qual è stata l'idea iniziale che mirava a dare una prima soluzione al problema.

3.1 Overview

Doxygen è un tool per la generazione automatica di documentazione. E' un progetto **open source** e **multipiattaforma**. E' possibile generare documentazione per diversi linguaggi; in particolare C++, C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, ed UNO/OpenOffice flavors), Fortran, VHDL, Tcl). Per comprendere l'effettiva utilità del tool nel caso specifico, sono state analizzate le possibilità offerte ad ampio spettro. Lo scopo iniziale è stato quello di generare documentazione per una applicazione scritta con la libreria *MASON* ed analizzare la struttura e la qualità delle informazioni generate dal tool.

3.1.1 Note per l'uso

Una volta istallato il tool (su Windows) due sono le modalità per la esecuzione:

- da terminale
- tramite interfaccia grafica (vedi figura 3.1)

File di configurazione.

Il primo passo da eseguire per l'utilizzo del tool è la creazione di un file di configurazione. Fondamentalmente avremo un file di configurazione per ciascun progetto che vogliamo documentare. Se utilizziamo *Doxygen* da riga di comando, la sintassi per la creazione del file di configurazione è la seguente:

```
doxygen -g nomeFileDiConfigurazione
```

Verrà così creato un file di configurazione standard che dovrà essere editato a seconda del progetto da documentare; una volta creato tale file possiamo eseguire *Doxygen*; da riga di comando basta digitare:

```
doxygen pathFileDiConfigurazione
```

Verrà generato quindi output in *html* e *latex*. L'interfaccia grafica nasconde all'utente la fase di generazione del file di configurazione, consentendo direttamente di specificare il valore per le opzioni che il tool rende disponibili. Infatti una delle caratteristiche che rende il tool estremamente interessante è proprio l'enorme quantità di opzioni selezionabili direttamente dal file di configurazione.

Eclox.

E' disponibile un plug-in per la piattaforma *Eclipse*, *Eclox* [4], che

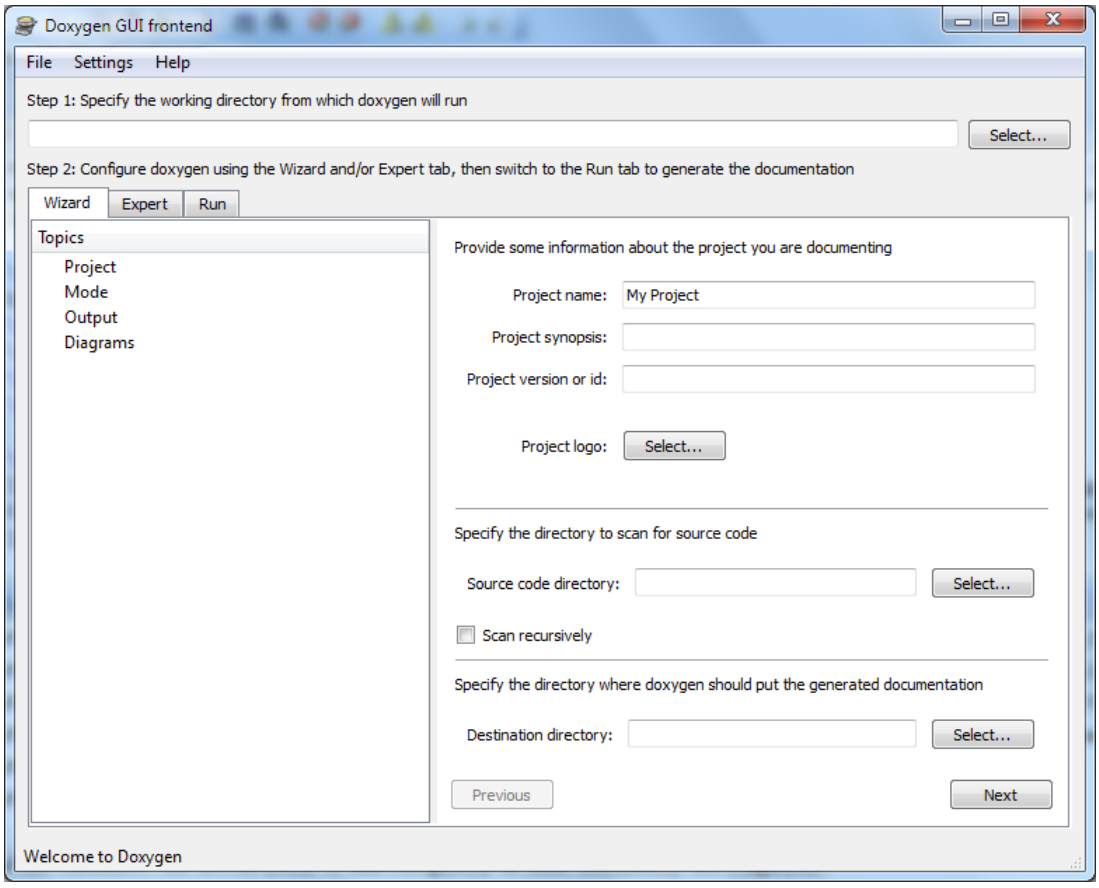


Figura 3.1: Interfaccia grafica di Doxygen su Windows

consente di eseguire *Doxygen* direttamente dalla IDE con semplici passaggi. Tale plug-in consente di specificare tutte le informazioni da interfaccia grafica; crea quindi il file di configurazione con le proprietà specificate dall'utente e lancia *Doxygen* sul progetto indicato (vedi 3.2). Le figure 3.3 e 3.4 mostrano degli screenshots del plug-in.

Vediamo alcune delle proprietà del file di configurazione:

- INPUT: *pathname* del progetto da documentare
- OUTPUT DIRECTORY: *pathname* in cui sarà generato l'output
- PROJECT NAME: nome del progetto che comparirà all'interno della documentazione

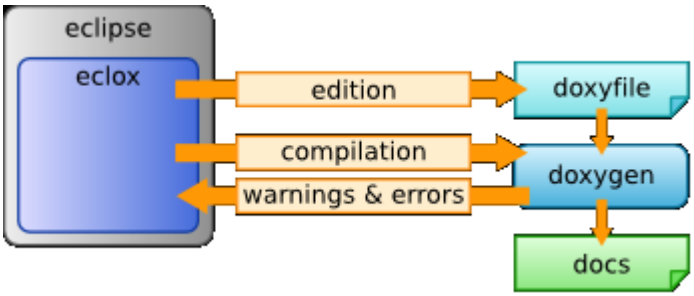


Figura 3.2: Struttura Eclox

- INLINE SOURCES: consente di inserire il codice sorgente all'interno della documentazione
- REFERENCED BY RELATION: impostato a *YES*, per ogni funzione documentata verranno listate tutte le altre funzioni documentate che hanno relazioni con questa
- REFERENCES RELATION: impostato a *YES*, per ogni funzione documentata verranno listate tutte le entità interagenti con questa
- OUTPUT LANGUAGE: consente di impostare un linguaggio che verrà utilizzato nella struttura html generata

Grafici con Doxygen.

Un'altra delle possibilità offerte dal tool è quella di generare grafici. Per garantire tale caratteristica il tool collabora con *Graphviz* [5], un tool open source per la generazione di grafici. Per la generazione dei grafici vanno opportunamente impostate alcune proprietà nel file di configurazione; in particolare:

- HAVE DOT: tale proprietà va impostata a *YES*
- DOT PATH: dobbiamo indicare la *path* dell'eseguibile di graphviz
- DOT NUM THREADS: quando invochiamo graphviz, il processo creato è *multi-threads*; possiamo indicare qual è il numero massimo di *threads* che graphviz può lanciare
- CLASS GRAPH: impostato a *YES* verrà generato il diagramma delle classi per il progetto che stiamo documentando (
- CALLER GRAPH: genera il grafico delle referenze per ogni metodo 3.5

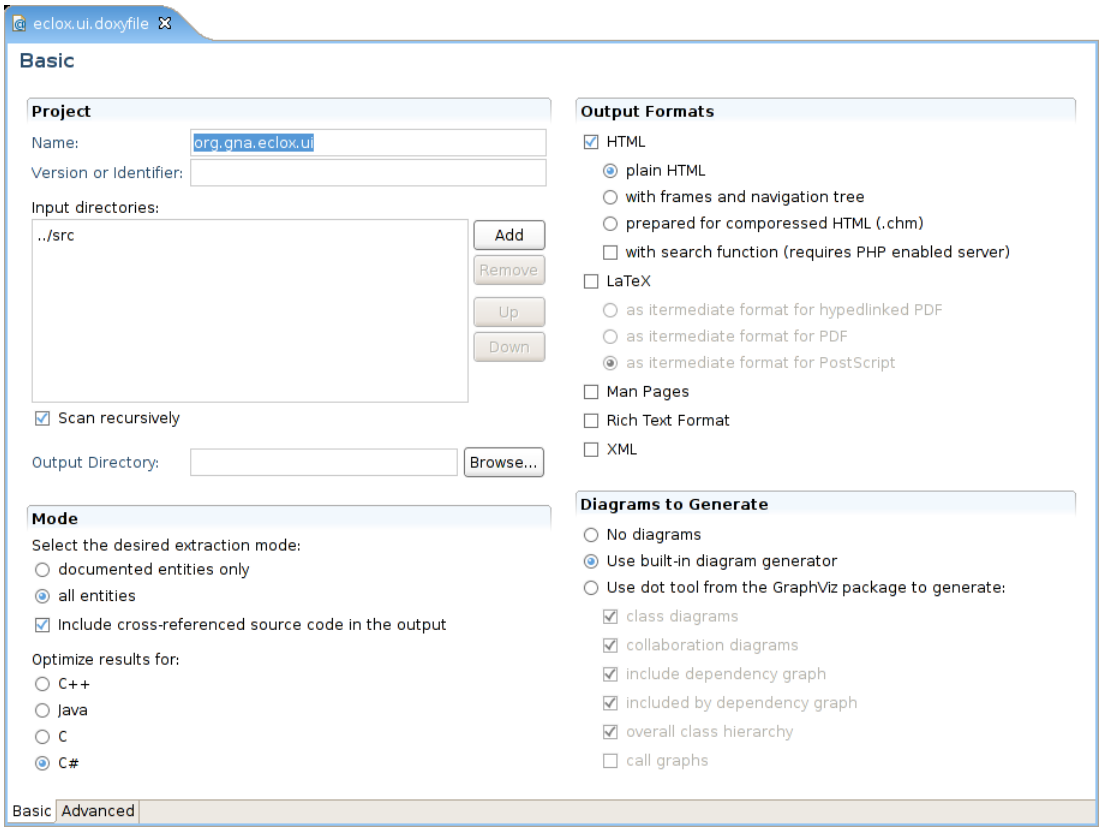


Figura 3.3: Eclox GUI

- CALL GRAPH: genera il grafico delle chiamate per ogni metodo 3.6

Commenti al codice.

Per far si che i commenti inseriti compaiano nella documentazione generata con *Doxygen* bisogna commentare il codice con oppurtuna sintassi. Diverse sono gli stili di commenti supportati da *Doxygen*; di seguito alcuni:

- JavaDoc style:

```
/**
* ... text ...
*/
```

-
```
///
/// ... text ...
```

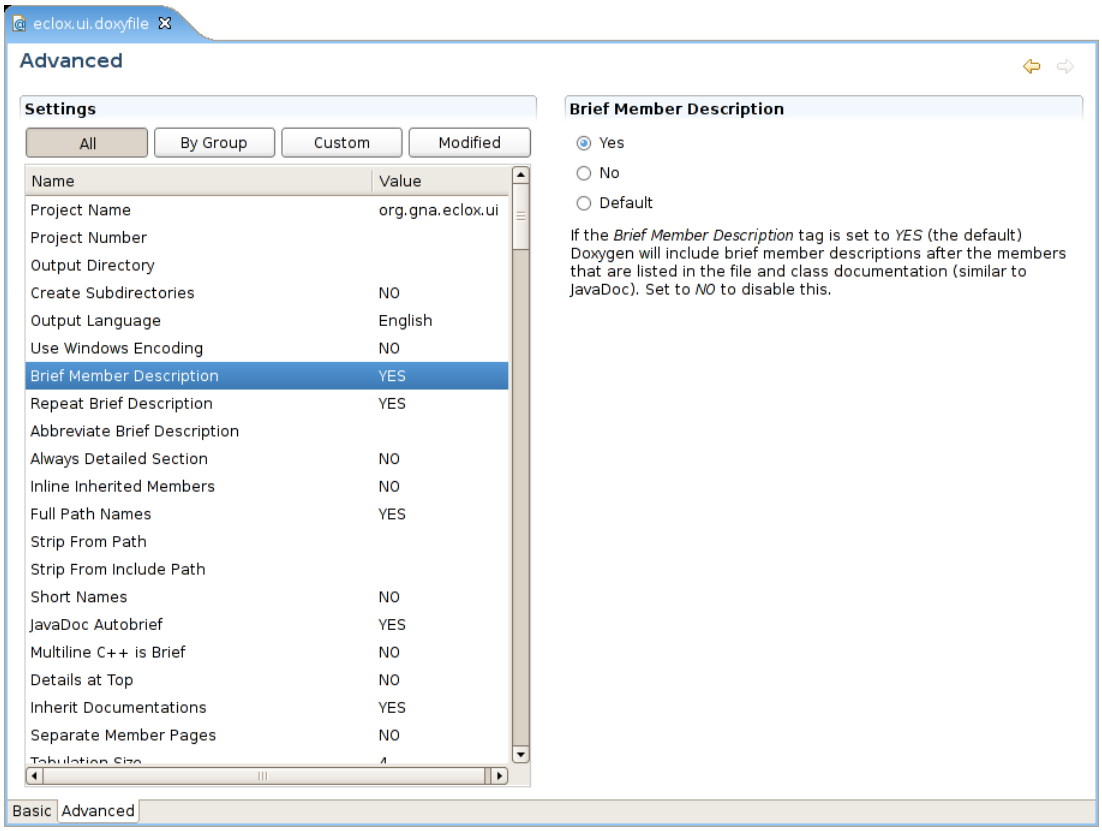


Figura 3.4: Eclox GUI

///

- /**//**
  * ... text
  **/

La tipologia di commenti utilizzata in questo lavoro è quella *JavaDoc style*, semplicemente perchè i programmi documentati sono programmi scritti in linguaggio java.

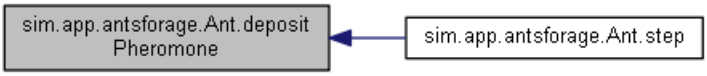


Figura 3.5: Caller graph

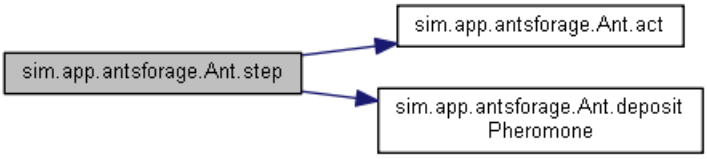


Figura 3.6: Call graph

3.1.2 Modules

Ricordiamo che il nostro scopo è quello di documentare quello che è il modello di simulazione descritto dalla applicazione. Come visto Doxygen permette, creando un opportuno file di configurazione, di avere delle informazioni aggiuntive (*REFERENCED BY RELATION, REFERENCES RELATION, CALL GRAPH, CALLER GRAPH*) in maniera *gratuita*; questa è già una delle motivazioni che ci spinge all'utilizzo di questo tool per il nostro scopo. In particolare quello che vogliamo raggiungere è una documentazione strutturata a nostro piacimento; vorremo avere in linea di principio una sola pagina nella documentazione che contenesse le informazioni del modello di simulazione. Questo eviterebbe all'esperto di modelli (non informatico) di ricercare informazioni all'interno di documentazione del progetto pensata per esperti informatici. Una delle caratteristiche di *Doxygen* che è da subito risultata interessante al fine del raggiungimento di tale scopo, è la possibilità di creare una sezione dedicata nella documentazione. All'interno di tale sezione possiamo aggiungere dei gruppi che andremo a popolare con le informazioni desiderate. La sintassi per la creazione di un gruppo è la seguente:

@defgroup GroupName Descrizione del gruppo

A questo punto possiamo inserire delle entità all'interno del gruppo aggiungendo quanto segue prima della entità:

@ingroup GroupName

Cosa altrettanto interessante è la possibilità di creare gruppi in altri gruppi già definiti.

La figura 3.7 mostra come viene strutturato l'output per quanto riguarda i moduli; in particolare è stato creato il gruppo *field* nel gruppo *SimulationModel*.

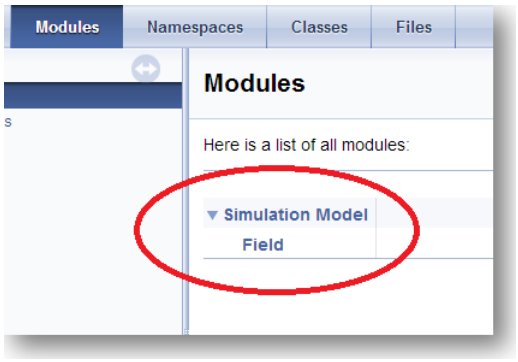


Figura 3.7: Moduli con Doxygen

3.2 MASON e Doxygen: l'idea iniziale

Abbiamo visto quelle che sono le caratteristiche di *Doxygen* che potenzialmente potevano offrire un supporto a quello che era il nostro scopo. Come è emerso da quanto detto finora, uno dei punti di forza di *Doxygen* è quello di essere altamente configurabile. Per quanto possa essere versatile non offre ('ovviamente') un supporto alla documentazione di programmi di simulazione multi agenti scritti in java con libreria *MASON*.

L'idea.

Nonostante ciò l'idea che è scaturita dopo uno studio generale del tool è stata la seguente: creare un gruppo che chiamiamo *SimulationModel* all'interno della documentazione ed aggiungiamo a tale gruppo quelle che sono le entità(variabili, classi, metodi) che hanno un significato in termini di modello di simulazione. Questa cosa è stata testata in primo momento manualmente; è stato commentato il codice di *AntsForage*(simulazione presente nel pacchetto *MASON*) per avere un primo esempio di risultato che è stato abbastanza soddisfacente. Quello che si voleva raggiungere era la generazione automatica di commenti al codice che seguissero la sintassi di *Doxygen*, mirati quindi alla creazione del gruppo *SimulationModel* contenente tutte le informazioni riguardanti il modello di simulazione. Sin dall'inizio l'idea è stata quella di creare un plug-in di export(MAD - (Mason Assisted Documentation)) per la piattaforma Eclipse che consentisse di generare la documentazione contenente il gruppo *SimulationModel*.

Capitolo 4

MAD(Mason Assisted Documentation)- prima versione

Introduzione

Nel capitolo precedente abbiamo descritto le caratteristiche del tool *Doxygen*, focalizzandoci su quelle che sembravano fornire supporto per il raggiungimento del nostro obiettivo. In particolare abbiamo discusso dei *Modules*, sezioni dedicate all'interno della documentazione in cui poter inserire informazioni in maniera personalizzata. Abbiamo quindi introdotto l'idea che ha portato ad un primo risultato, ovvero la generazione automatica di commenti al codice che seguissero la sintassi di *Doxygen*, mirati alla creazione del gruppo *SimulationModel* contenente tutte le informazioni riguardanti il modello di simulazione. Per raggiungere tale obiettivo, il primo step è stato quello di studiare programmi scritti con la libreria *MASON* al fine di individuare in qualche modo una struttura ricorrente all'interno di tali programmi e le informazioni significative riguardo il modello di simulazione. In questo capitolo vedremo quindi cosa è emerso in seguito ad uno studio di programmi scritti con la libreria *MASON* ed analizzeremo quella che è stata la prima versione di MAD(plug-in sviluppato) soffermandoci sugli aspetti positivi e sulle critiche a cui tale versione si prestava.

4.1 Struttura ricorrente in MASON

Ambiente ed agenti.

A seguito dello studio di alcuni esempi di simulazione forniti con la libreria *MASON* è stata individuata una struttura. Una prima informazione semantica che possiamo ricavare da un programma che utilizza la libreria *MASON* è la seguente: sappiamo che l'ambiente della simulazione sarà la classe che estende *SimState* e che gli agenti implementeranno l'interfaccia *Steppable*. Siamo quindi a conoscenza di quelli che sono gli agenti della simulazione e dell'ambiente in cui questi interagiscono. Vediamo dunque quali sono le caratteristiche individuate in entrambe le classi.

4.1.1 SimState

La classe *SimState* offre solitamente una serie di informazioni importanti ai fini della comprensione del modello. Di seguito una lista degli elementi che solitamente emergono dalla *SimState*:

- *Fields*: come introdotto in 2.2.1 i *Fields* rappresentano le griglie (2D o 3D) su cui si muovono gli agenti della simulazione; in *MASON* esistono degli oggetti tipici per i *fields*; possiamo dunque individuare queste informazioni automaticamente andando a ricercare le variabili il cui tipo corrisponde al tipo di un *field*
- Dimensioni dei *Fields*: spesso le dimensioni dei *fields* sono specificate in variabili esterne; possiamo automaticamente individuare il valore di tali variabili e possiamo creare un commento del tipo *La simulazione si svolge su una griglia 2D di dimensioni x per y*
- Metodo start: il metodo start della *SimState* ci consente di ricavare informazioni sullo stato iniziale della simulazione; analizzando tale metodo possiamo individuare come vengono schedulati gli agenti.

4.1.2 Steppable

Analizzando progetti che utilizzano la libreria *MASON* saranno presenti(nella maggior parte dei casi) una o più classi che implementano l'interfaccia *Steppable*. Come già detto tali classi rappresenteranno gli agenti della simulazione. Di seguito gli elementi che solitamente emergono dalla classe *Steppable*:

- Variabile descrivente la locazione dell'utente; concretamente sarà un oggetto di una delle seguenti classi:

  - sim.util.Int2D and sim.util.MutableInt2D
  - sim.util.Double2D and sim.util.MutableDouble2D
  - sim.util.Int3D and sim.util.MutableInt3D
  - sim.util.Double3D and sim.util.MutableDouble3D

- metodo step: tale metodo contiene le operazioni che l'agente svolgerà ad ogni passo della simulazione; ha come parametro un oggetto della classe *SimState* che consiste nell'ambiente su cui l'agente si muove

4.2 Semantic analysis of code

Quello che stavamo andando a realizzare era quindi un plug-in che andasse ad esaminare il codice alla ricerca delle informazioni sopra elencate. Per ognuna delle entità(classi, variabili) e dei metodi significativi, si voleva aggiungere il commento in stile *Doxygen* per inserire tale entità (o metodo) all'interno del gruppo *SimulationModel*. Si invocava a questo punto *Doxygen*, andando prima a creare il file di configurazione opportunamente impostato, e si analizzava l'output ottenuto. La figura 4.1 mostra un esempio di quello che si andava a realizzare. Nel caso specifico è mostrata l'anailisi di una variabile di tipo *IntGrid2D*; tale tipo rappresenta un *field*. Dunque andiamo ad analizzare la inizializzazione per individuare il valore dei parametri.

4.3 MAD (Mason Assisted Documentation)

Vediamo in questa sezione la prima versione di MAD, un plug-in per la piattaforma Eclipse che è il prodotto del nostro lavoro. Analizzeremo il prodotto ad alto livello, senza scendere in dettagli implementativi che saranno analizzati nel capitolo dedicato all'implementazione. MAD è un plug-in di export che consente di generare, partendo da un progetto scritto in java che utilizza la libreria *MASON*, documentazione in maniera assistita utilizzando *Doxygen*. Abbiamo visto nel paragrafo 4.1 quali sono le entità significative in programmi che utilizzano la libreria MASON; fondamentalmente quello che MAD faceva alla prima versione era andare a ricercare all'interno del progetto che si analizzava le entità significative in termini del modello; per ciascuna di queste entità generava una quantità di informazioni automaticamente lasciando all'utente la possibilità di inserire ulteriori informazioni. Creava quindi il gruppo *SimulationModel* a cui aggiungeve le entità; lanciava

Funzionamento.

quindi il tool generando documentazione. Per il parsing del codice è stata utilizzata la libreria AST (Abstract Syntax Tree) [6][1].

4.3.1 Critiche

Vediamo in questa sezione le critiche a cui la prima versione di MAD si prestava.

Dal punto di vista implementativo

Parsing del codice.

Lo sviluppo di questa applicazione richiedeva un parser del codice molto affidabile. Il programmatore java ha diverse modalità di scrittura del codice. Consideriamo ad esempio la dichiarazione di una variabile (un *field* della simulazioned): l'utente può scrivere quanto segue:

- public IntGrid2D sites = new IntGrid2D(GRID_WIDTH, GRID_HEIGHT);
- public IntGrid2D sites = new IntGrid2D(GRID_WIDTH,GRID_HEIGHT);
- public IntGrid2D sites = new IntGrid2D(1,2);
- public IntGrid2D sites = new IntGrid2D(new Integer(1),2);

Come mostra la figura 4.1, dalla inizializzazione di un *field* possiamo ricavare delle informazioni; in particolare potremo ad esempio dire che

> La variabile *nomeVariabile* rappresenta un *field* della simulazione su cui agiscono gli agenti. La larghezza del *field nomeVariabile* è *parametro1*; l'altezza è parametro2. *parametro1* vale *valoreParametro1*; *parametro2* vale *valoreParametro2*.

Concettualmente per ricavare le informazioni sopra citate bisogna *visitare* l'inizializzazione delle variabili della classe che estende *SimState*; per le variabili il cui tipo coincide con i tipi di un *Field* andiamo a *visitare* l'inizializzazione; sapendo cosa rappresentano i parametri di input dei *Field* possiamo ricavare l'informazione così come mostrata sopra. La prima versione andava a ricercare tali informazioni come segue: ricavava la stringa dell'inizializzazione di un *field* (utilizzando la libreria AST); a questo punto per ricavare i parametri eseguiva una serie di split; ricavati i parametri, se questi non erano interi, andava a ricercare le variabili all'interno della classe per ricavare i valori di tali variabili. Dall'elenco 4.3.1 si può intuire qual è il problema; consideriamo ad esempio l'inizializzazione seguente:

[1] I dettagli su come questo è stato realizzato sono nel capitolo relativo all'implementazione.

(GRID_WIDTH, GRID_HEIGHT)

in questo caso eseguendo uno *split* con parametro ',' (andando quindi a spezzare la stringa ad ogni ',') si ottengono le seguenti stringhe:

- 'GRID_WIDTH'
- ' GRID_HEIGHT' (si noti lo spazio prima della lettera 'G')

E' a questo punto evidente che la ricerca successiva delle variabili all'interno della classe porta ad un fallimento nel caso della variabile ' GRID_HEIGHT'.

Iterabilità.

Una delle cose a cui bisognava pensare sin dall'inizio è il fatto che questo tipo di processo deve garantire l'iterabilità; l'utente può infatti eseguire MAD più volte sullo stesso progetto, eventualmente modificando parametri, commenti e quant'altro e deve poter rivalutare commenti inseriti in una iterazione precedente. Nella prima versione la maniera in cui questa funzionalità era sviluppata non era eccellente. Come già detto le informazioni generate automaticamente o inserite dall'utente venivano aggiunte come commento (in stile *Doxygen*) sulle varie entità. Queste informazioni precedentemente inserite dall'utente nei successivi utilizzi del plug-in venivano recuperate dai commenti. In pratica veniva rigenerato il commento automatico, veniva confrontato con quello esistente e, nella casella dedicata all'input dell'utente, veniva inserita la *differenza* tra l'intero precedente commento (autogenerato ed inserito dall'utente) e il commento appena autogenerato. Nel caso in cui l'utente faceva delle modifiche al codice, il commento automatico generato poteva essere diverso dal commento automatico generato precedentemente; quindi quello che accadeva era quanto segue: nella casella relativa al commento automatico era presente il commento automatico appena generato; nella casella dedicata all'utente compariva l'intero commento precedente (automatico più quello iunserito dall'utente); le informazioni mostrate non erano quindi spesso coerenti con quelle inserite.

Dal punto di vista dell'output

Il problema principale della prima versione di MAD era tuttavia legato all'output generato. Quello che capitava è che l'output conteneva la sezione *Simulation Model* che conteneva tutte le entità commentate e documentate. Effettivamente tutte le informazioni presenti nella sezione erano significative in termini del modello di simulazione. Il problema era la struttura della documentazione; avere troppe informazioni non strutturate appositamente è equivalente a non avere alcuna informazione. Fondamentalmente bisognava capire come un ABMs va effettivamente documentato considerando quindi

le informazioni che l'esperto del modello si aspetta di trovare nella documentazione. Per risolvere questo problema è stato studiato il protocollo ODD(Overview, Design concepts and details) [1] che si pone come standard per la documentazione di ABMs.

4.3.2 Lati positivi

Questa prima versione di MAD è comunque risultata un importante punto di avvicinamento per l'obiettivo che ci eravamo posti. Avevamo individuato le informazioni interessanti in programmi che utilizzano la libreria MASON e come utilizzare *Doxygen* per raggruppare tali informazioni in un'unica sezione all'interno della documentazione.

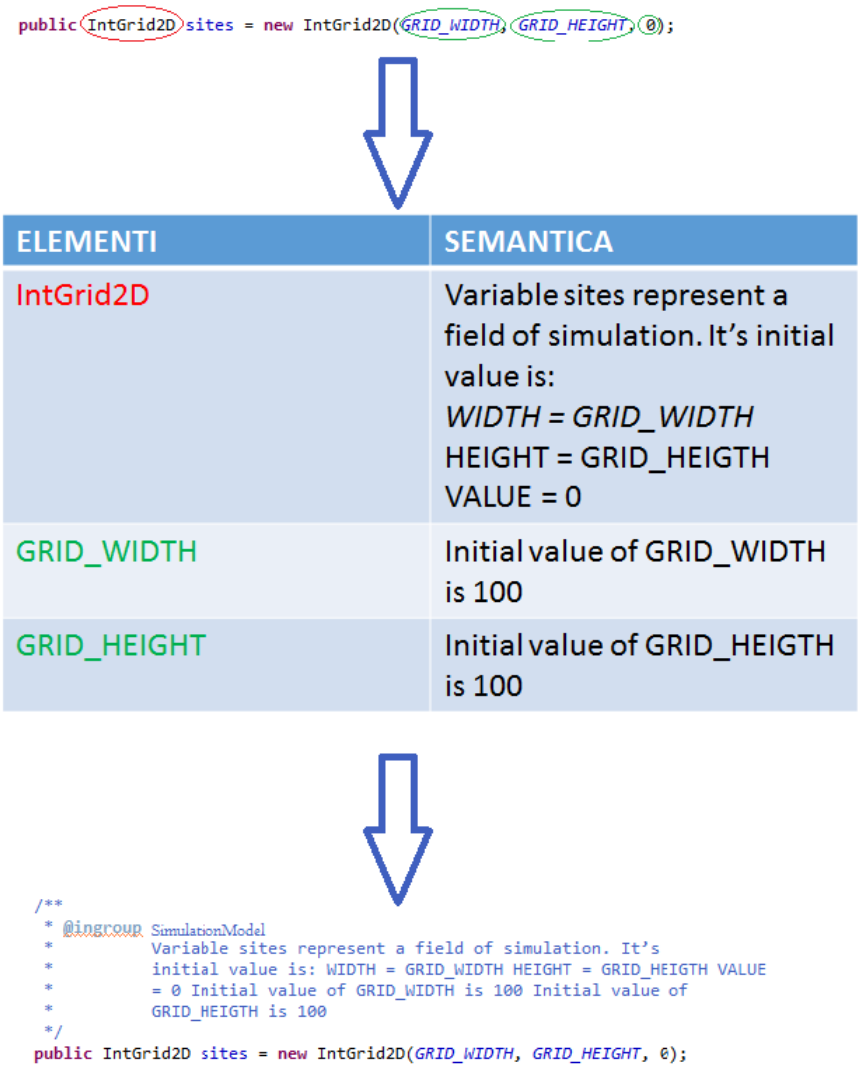


Figura 4.1: Esempio di analisi semantica di codice MASON

Capitolo 5

ODD protocol e MASON

Introduzione

Il capitolo precedente ha mostrato la prima versione di MAD evidenziando i lati positivi e le critiche a cui tale versione si prestava. Partendo proprio dalle problematiche della prima versione di MAD vedremo come la versione attuale le risolve. In particolare descriveremo il protocollo ODD(Overview, Design concepts and Details) [1], standard per la documentazione di ABMs, evidenziando le relazioni tra le linee guida del protocollo e le caratteristiche di MASON. Vedremo infine come il tool *Doxygen* ha aiutato a strutturare l'output secondo lo standard fornito da ODD.

5.1 ODD protocol

Come visto nel precedente capitolo, uno dei problemi della prima versione di MAD era legata all'output generato. Le informazioni contenute nella documentazione generata erano sicuramente significative in termini del modello di simulazione descritto dalla applicazione documentata; la struttura delle informazioni però non si prestava ad una facile lettura. Fornire quindi tante informazioni non strutturate è equivalente a non fornire alcuna informazione. E' proprio questa una delle critiche che spesso viene fatta alla documentazione di ABMs; essi presentano nella maggior parte dei casi una documentazione povera o non strutturata. Il fatto di non avere una dcoumentazione di qualità porta addirittura a non poter valutare un modello di simulazione. Il protocollo ODD(Overview, Design concepts, Details) [1] nasce nel 2006 ponendosi come obiettivo proprio quello di risolvere questo problema. Tale protocollo sta risultando molto efficace; è stato citato, nonostante sia relativamente di recente data, in 50 pubblicazioni. Utilizzando ODD è emerso che, oltre ad agevolare l'utente nella fase di documentazione del modello di simulazione, può essere usato anche in fase di definizione del modello in quanto propone una checklist che riassume tutte le domande a cui un buon modello deve rispondere. Risulta fondamentale ad ogni utilizzo di tale protocollo aggiungere la seguente frase nella documentazione: *the model description follows the ODD (Overview, Design concepts, Details) protocol (Grimm et al., 2006, this work.*

	Elements of the original ODD protocol (Grimm et al., 2006)	Elements of the updated ODD protocol
Overview	1. Purpose 2. State variables and scales 3. Process overview and scheduling	1. Purpose 2. Entities, state variables, and scales 3. Process overview and scheduling
Design concepts	4. Design concepts • Emergence • Adaptation • Fitness • Prediction • Sensing • Interaction • Stochasticity • Collectives • Observation	4. Design concepts • Basic principles • Emergence • Adaptation • Objectives • Learning • Prediction • Sensing • Interaction • Stochasticity • Collectives • Observation
Details	5. Initilization 6. Input 7. Submodels	5. Initialization 6. Input data 7. Submodels

Figura 5.1: I sette elementi del protocollo ODD.

La figura 5.1 mostra i sette elementi del protocollo, sia nella prima versione che nella versione aggiornata.

5.2 Elementi del protocollo ODD

Sette sono gli elementi del protocoll ODD. Si possono raggruppare in tre categorie:

- Overview: comprende le sottocategorie *Purpose, State variables and scales* e *Process overview and scheduling*
- Design concepts: contiene nove design concepts (*Emergence, Adaption, Fitness, Prediction, Sensing, Interaction, Stochasticity, Collectives* ed *Observation*)
- Details: comprende *Initialization, Input* e *Submodels*

Vediamo ora in dettaglio ciascuno dei sette elementi.

5.2.1 Purpose

La domanda a cui questa sezione deve rispondere è la seguente: *Qual è lo scopo del modello?*. E' importante non aggiungere dettagli sul funzionamento del modello a questo livello di documentazione.

5.2.2 Entities, state variables, and scales

La domanda a cui si risponde è: *Quali tipi di entità sono presenti nel modello? Da quali attributi sono caratterizzate? Quale è il modello temporale-spaziale?*. Una entità è un oggetto/attore distinto che interagisce con altre entità o è influenzato da fattori esterni. Le entità tipiche di un ABMs sono le seguenti:

- Agenti
- Unità spaziali (ad esempio griglie su cui sui svolge la simulazione)
- Ambiente
- Gruppi di agenti

Importante il concetto di stato di un'entità. Lo stato è descritto da alcune variabili che possono essere:

- Statiche (età, sesso, locazione di una entità non mobile)
- Dinamiche (variano nel tempo)

Una delle cose che potrebbe contribuire molto nella descrizione dello stato delle entità e di come questo varia nel corso della simulazione potrebbe essere quanto segue:

- Avviare la simulazione e stopparla ad un certo istante;
- Salvare le informazioni in questo intervallo
- Riavviare la simulazione e rifermarla
- Raccogliere le informazioni nel nuovo stato
- Confrontare le informazioni raccolte

5.2.3 Process overview and scheduling

La domanda a cui si deve rispondere in questa sezione è la seguente: *Chi fa cosa? In quale ordine? Il tempo è modellato con steps continui o discreti?*. Bisogna quindi descrivere le azioni svolte dalle varie entità del modello. Risulta fondamentale non scendere nei dettagli ma creare una sorta di lista di tali azioni; per ciascuna azione (che risulterà essere un elemento di tale lista) dovrà corrispondere un paragrafo nella sezione Submodels in cui verranno dati dettagli su come l'azione viene implementata. Idealmente il submodels dovrebbe contenere una sorta di pseudo-codice che potrebbe risultare utile nel caso di una nuova implementazione del modello.

5.2.4 Design concepts

Le informazioni che questa categoria contiene sono elencate di seguito:

1. Basic Principles: in questa sezione vanno inseriti i concetti di base che regolano la simulazione, le ipotesi, l'evoluzione della complessità.
2. Emergence: tale sezione risponde alla domanda *quali sono le chiavi emergenti del modello?*
3. Adaption: *in che maniera gli agenti prendono decisioni?*
4. Objectives: se alcuni agenti prendono delle decisioni al fine di raggiungere degli obiettivi, quali sono precisamente tali obiettivi? Quali sono i criteri che l'agente utilizza per raggiungere il suo scopo?
5. Learning: alcuni agenti cambiano il loro comportamento nel tempo in base alla loro esperienza; in questa sezione spieghiamo come un agente accumula esperienza

6. Prediction: in base alla esperienza maturata da un agente possono essere fatte delle assunzioni per il futuro?
7. Sensing: quali variabili d'ambiente considera un agente prima di prendere decisioni sui passi futuri?
8. Interaction: come interagiscono gli utenti? (direttamente tra loro, passando per l'ambiente...)
9. Stochasticity: ci sono delle entità che variano in maniera casuale nel tempo?
10. Collectives: come interagiscono gli agenti nella collettività?
11. Observation: in questa sezione vengono inserite delle osservazioni sul modello che non rientrano in nessun'altra categoria.

5.2.5 Initialization

Importante è definire lo stato iniziale della simulazione, ovvero descrivere i valori delle variabili di stato al tempo *t=0*, evidenziare se l'inizializzazione è casuale o è sempre la stessa ad ogni avvio della simulazione.

5.2.6 Input data

In modelli che descrivono sistemi reali, alcune informazioni potrebbero variare dinamicamente e quindi dipendere da alcuni input. In questa sezione rispondiamo quindi alla seguente domanda: *Does the model use input from external sources such as data files or other models to represent processes that change over time?*

5.2.7 Submodels

Come visto in 5.2.3, durante la fase *Process overview and scheduling* viene creata una lista delle azioni che vengono svolte ad ogni passo della simulazione. Per ogni azione in questa sezione vengono dati dettagli su come l'azione viene svolta; l'ideale sarebbe avere una sorta di pseudo-codice per ciascuna azione.

5.3 ODD e MASON

Ritorniamo quindi all'obiettivo del nostro lavoro che, ricordiamo, punta a documentare ABMs scritti in java utilizzando la libreria MASON. ODD come visto fornisce un buon approccio alla documentazione di ABMs. Quello che dobbiamo a questo punto capire è quali sono le relazioni tra le sezioni di ODD ed un programma che utilizza la libreria MASON.

Purpose. In generale, analizzando un programma java risulta difficile individuare qual è lo scopo che vuole raggiungere. Un programma potrebbe semplicemente anare in loop e non si vuole in questo lavoro di tesi risolvere il *problema della fermata*[1]. Pertanto anche analizzando un programma di simulazione non siamo in grado di ottenere una risposta alla domanda *Qual è lo scopo del modello?*.

Entities, state variables, and scales. In MASON possiamo(in parte) individuare queste informazioni. Possiamo individuare gli agenti della simulazione partendo dalla considerazione che un agente è rappresentato da una classe che implementa l'interfaccia *Steppable*; per quel che riguarda le unità spaziali possiamo analizzare la classe che estende *SimState* che, come descritto in 2.2.1. In particolare possiamo cercare all'interno di tale classe le variabili che rappresentano in *field* della simulazione e ricavare informazioni sulla misura delle griglie. Quello che non siamo capaci di individuare è a cosa la misura della griglia corrisponde; possimao dire che la griglia x è larga y ed alta z ma non possiamo conoscere l'unità di misura di x e y (metri, miglia...).

Process overview and scheduling Per rispondere alla domanda *Chi fa cosa e in quale ordine* possiamo analizzare i metodi *start()* della classe che estende *SimState*(cha in qualche modo farà riferimenti allo *schedule* per specificare l'ordine con cui gli agenti saranno processati) e il metodo *step()* dei vari agenti per capire quali sono le azioni svolte da ogni agente ad ogni passo della simulazione.

Design concepts. Per i concetti di design, non è stata ancora individuata una relazione tra questi ultimi e la libreria MASON. Per rimanere comunque coerenti al protocollo ODD, aggiungeremo questa sezione nella documentazione chiedendo all'utente di aggiungere informazioni per ciascuno dei concetti di design.

Initialization. Per quanto riguarda l'inizializzazione, possiamo mostrare il valore iniziale di ciascuna variabile della simulazione.

[1] Il problema della terminazione (dall'inglese Halting problem, tradotto anche con problema dell'arresto o problema della fermata) chiede se sia sempre possibile, descritto un algoritmo e un determinato input finito, stabilire se l'algoritmo in questione termini o continui la sua esecuzione all'infinito. È stato dimostrato che non può esistere un algoritmo generale che possa risolvere il problema per tutti i possibili input. La versione più nota del problema è quella proposta nel 1936 dal matematico Alan Turing, insieme alla dimostrazione della sua indecidibilità.

Ovviamente tali informazioni possono essere generate in maniera automatica, senza aiuto dell'utente. Come già accennato precedentemente, alcune simulazini potrebbere prendere degli input da file esterni, database o altre fonti. Questa informazione la lasciamo aggiungere all'utente. Nella fase *Process overview and scheduling* andiamo ad analizzare il metodo *start()* della classe che estende *SimState* ed il metodo *step()* dei vari agenti. Quello che andiamo a fare in questo caso è quanto segue: si *visita* ognuno di questi metodi *appuntando* le eventuali chiamate a funzioni. L'idea è creare una lista delle azioni svolte all'interno del metodo durante la fase *Process overview and scheduling* (ogni elemento della lista sarà l'invocazione di una funzione) e, nella sezione *Submodels* andiamo a ricavare la lista e ad analizzare dettagliatamente ogni funzione *appuntata*. E' ovvio che in questo contesto è fondamentale lo stile del programmatore; laddove il programmatore lasci nei metodi *start()* e *step()* semplicemente una serie di invocazioni a funzioni, l'output sarà perfettamente coerente con quanto richiesto dal protocollo ODD. Per rendere meglio l'idea mostriamo il metodo *step()* del programma *AntsForage*, uno degli esempi forniti con la libreria MASON:

Input data.

Submodels.

```
1       public void step(final SimState state){
2       depositPheromone(state);
3       act(state);
4       }
```

laddove l'utente inserisca tutte le istruzioni all'interno dei metodi, probabilmente la sezione *Process overview and scheduling* darà troppe informazioni. E' il caso del metodo *start()* della stessa simulazione *AntsForage*:

```
1       public void start() {
2           super.start();
3           sites = new IntGrid2D(GRID_WIDTH, GRID_HEIGHT, 0);
4           toFoodGrid = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT, 0);
5           toHomeGrid = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT, 0);
6           buggrid = new SparseGrid2D(GRID_WIDTH, GRID_HEIGHT);
7           obstacles = new IntGrid2D(GRID_WIDTH, GRID_HEIGHT, 0);
8           switch (OBSTACLES) {
9           case NO_OBSTACLES:
10              break;
11          case ONE_OBSTACLE:
12              for (int x = 0; x < GRID_WIDTH; x++)
13                  for (int y = 0; y < GRID_HEIGHT; y++) {
14                      obstacles.field[x][y] = 0;
```

```
15                      if (((x - 55) * 0.707 + (y - 35) * 0.707)
16                              * ((x - 55) * 0.707 + (y - 35) * 0.707) / 36
17                              + ((x - 55) * 0.707 - (y - 35) * 0.707)
18                              * ((x - 55) * 0.707 - (y - 35) * 0.707) / 1024 <= 1)
19                          obstacles.field[x][y] = 1;
20                  }
21              break;
22          case TWO_OBSTACLES:
23              for (int x = 0; x < GRID_WIDTH; x++)
24                  for (int y = 0; y < GRID_HEIGHT; y++) {
25                      obstacles.field[x][y] = 0;
26                      if (((x - 45) * 0.707 + (y - 25) * 0.707)
27                              * ((x - 45) * 0.707 + (y - 25) * 0.707) / 36
28                              + ((x - 45) * 0.707 - (y - 25) * 0.707)
29                              * ((x - 45) * 0.707 - (y - 25) * 0.707) / 1024 <= 1)
30                          obstacles.field[x][y] = 1;
31                      if (((x - 35) * 0.707 + (y - 70) * 0.707)
32                              * ((x - 35) * 0.707 + (y - 70) * 0.707) / 36
33                              + ((x - 35) * 0.707 - (y - 70) * 0.707)
34                              * ((x - 35) * 0.707 - (y - 70) * 0.707) / 1024 <= 1)
35                          obstacles.field[x][y] = 1;
36                  }
37              break;
38          case ONE_LONG_OBSTACLE:
39              for (int x = 0; x < GRID_WIDTH; x++)
40                  for (int y = 0; y < GRID_HEIGHT; y++) {
41                      obstacles.field[x][y] = 0;
42                      if ((x - 60) * (x - 60) / 1600 + (y - 50) *
43                          (y - 50) / 25 <= 1)
44                          obstacles.field[x][y] = 1;
45                  }
46              break;
47          }
48          for (int x = HOME_XMIN; x <= HOME_XMAX; x++)
49              for (int y = HOME_YMIN; y <= HOME_YMAX; y++)
50                  sites.field[x][y] = HOME;
51          for (int x = FOOD_XMIN; x <= FOOD_XMAX; x++)
52              for (int y = FOOD_YMIN; y <= FOOD_YMAX; y++)
53                  sites.field[x][y] = FOOD;
54          for (int x = 0; x < numAnts; x++) {
```

```
55              Ant ant = new Ant(reward);
56              buggrid.setObjectLocation(ant, (HOME_XMAX + HOME_XMIN) / 2,
57                      (HOME_YMAX + HOME_YMIN) / 2);
58              schedule.scheduleRepeating(Schedule.EPOCH + x, 0, ant, 1);
59          }
60          schedule.scheduleRepeating(Schedule.EPOCH, 1, new Steppable() {
61              public void step(SimState state) {
62                  toFoodGrid.multiply(evaporationConstant);
63                  toHomeGrid.multiply(evaporationConstant);
64              }
65          }, 1);
66      }
```

5.4 ODD e Doxygen

Abbiamo quindi introdotto ODD sottolineando le relazioni con la libreria MASON. A questo punto l'idea è stata ovviamente quella di strutturare le informazioni generate dalla prima versione di MAD secondo lo standard ODD, aggiungendo le informazioni che non comparivano. Ricordiamo che la prima versione di MAD è legata al tool *Doxygen* per la generazione dell'output. Come visto in 3.1.2 possiamo creare moduli nella documentazione generata con *Doxygen* e possiamo creare moduli in altri moduli già esistenti. L'idea è stata quindi quelle di ricreare la struttura di ODD nella documentazione. La figura 5.2 mostra la struttura del protocollo ODD ricreata con Doxygen.

La sintassi da utilizzare nei commenti per ricreare la struttura della figura 5.2 è la seguente:

```
1   /**
2     * @defgroup SimulationModel ODD model documentation
3       * @defgroup Purpose
4       * @ingroup SimulationModel
5       * @defgroup Entities Entities, state variables, and scales
6       * @ingroup SimulationModel
7       * @defgroup Process Process overview and scheduling
8       * @ingroup SimulationModel
9       * @defgroup Design Design concepts
10      * @ingroup SimulationModel
11      * @defgroup Initialization Initialization
12      * @ingroup SimulationModel
```

```
13        * @defgroup Input Input data
14        * @ingroup SimulationModel
15        * @defgroup Submodels
16        * @ingroup SimulationModel
17        */
```

Ricreare una struttura come questa e popolare in maniera adeguata le varie sezioni richiede un certo lavoro al programmatore. In particolare l'utente dovrebbe studiare il protocollo ODD e capire le relazioni tra ODD e MASON, dovrebbe creare commenti con la sintassi opportuna per Doxygen per creare la struttura in figura 5.2 e commentare in maniera opportuna le varie entità del programma. Per le sezioni *Process overview and scheduling* e *Submodels* dovrebbe generare pseudo-codice ed inserirlo in maniera opportuna in commenti comprensibili da Doxygen. Quello che MAD promette è proprio una semplice procedura guidata che nasconde tutto lo studio fatto in questo lavoro di tesi generando documentazione di qualità per programmi di simulazione scritti in java con libreria MASON.

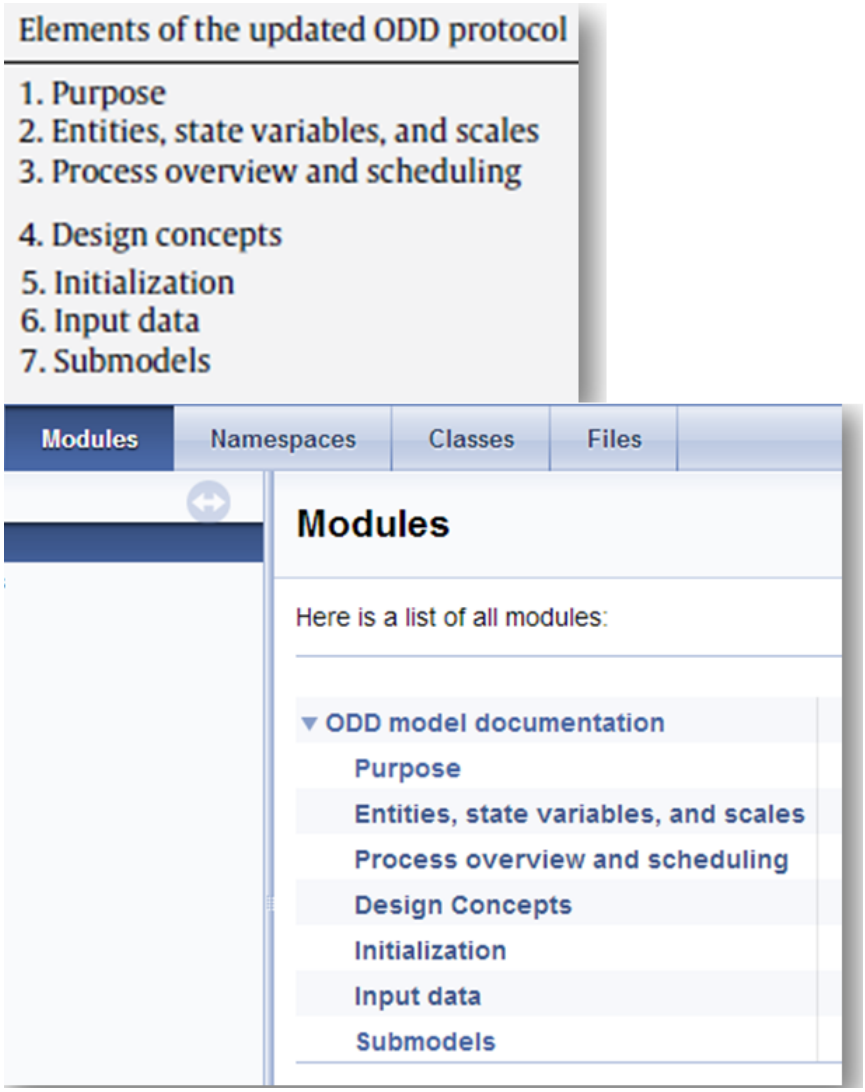


Figura 5.2: Struttura del protocollo ODD in documentazione generata con Doxygen.

Capitolo 6

MAD- versione corrente

Introduzione

Il capitolo 4 ha mostrato la prima versione di MAD, analizzando le critiche a cui tale versione si prestava. Come già illustrato, una delle problematiche principali della prima versione era la struttura della documentazione generata. Abbiamo quindi introdotto nel capitolo 5 il protocollo ODD e le relazioni tra tale protocollo e la libreria MASON, oggetto di studio di questo lavoro. In questo capitolo illustreremo quindi come la versione attuale di MAD ha risolto le problematiche della prima versione, sia per quel che riguarda la struttura dell'output sia per le altre problematiche (parsing del codice, iterabilità) descritte in 4.3.1.

6.1 Seconda versione

Descriviamo in questa sezione le novità introdotte dalla versione successiva alla prima. In particolare vedremo come sono stati risolti i problemi di parsing del codice, di iterabilità (vedi 4.3.1) e vedremo le novità introdotte.

6.1.1 Parsing del codice

Come descritto in 4.3.1, uno dei problemi della prima versione era legato alla maniera in cui era effettuato il parsing del codice. Utilizzando al meglio le caratteristiche della libreria AST [6] di eclipse tale problema è stato risolto.

6.1.2 Visitor pattern

In 4.3.1 abbiamo descritto ad esempio il problema legato alla ricerca dei parametri dei *field* della simulazione. Utilizzando il visitor pattern di ast è stato realizzato l'oggetto 'FieldDeclarationVisitor' che visita la 'compilationUnit' data come parametro implicito 'visitando' gli 'Assignment'; trovato il node Assignment istanza di 'ClassInstanceCreation' e con nome uguale alla field che cerchiamo, si estrae la lista di parametri con 'arguments()'(dettagli in A.1.2).

6.1.3 Iterabilità

In 4.3.1 è illustrato il problema legato all'iterabilità del processo; l'utente può utilizzare MAD più volte sullo stesso progetto, modificando entità, commenti e quant'altro. Per risolvere i problemi descritti in 4.3.1 è stato introdotto l'ogettto ODD A.1.1. Tale oggetto ha come obiettivo quello di raccogliere le informazioni durante le varie fasi del wizard. L'oggetto viene serializzato alla fine della procedura guidata e recuperato ad un nuovo utilizzo. Dunque le informazioni eventualmente inserite dall'utente non vengono recuperate dai commenti al codice(come avveniva nella prima versione di MAD) ma vengono recuperate direttamente dall'oggetto ODD. Lo scopo quindi per cui tale oggetto è stato introdotto è fondamentalmente quello di risolvere il problema dell'iterabilità. Quello che invece è emerso è che l'oggetto ODD ci ha svincolato dall'utilizzo del tool *Doxygen*. Infatti una volta raccolte tutte le informazioni secondo lo standard ODD siamo liberi di generare outuput in qualsiasi altra modalità.

6.1.4 Process overview and scheduling

Vediamo adesso come la seconda versione si comporta con la sezione *Process overview and scheduling* del protocollo ODD. Quello che accade è proprio quanto descritto in 5.3. Le classi che realizzano quanto descritto in 5.3 sono riportate in 7.2.5.

6.1.5 Color feature

Una delle caratteristiche aggiunte alla seconda versione è la possibilità di selezionare un diverso colore per quanto riguarda le informazioni generate automaticamente da MAD e quelle inserite dall'utente. Questo perchè l'utente potrebbe voler considerare con diverso peso questi due tipi di informazione. *Doxygen* consente di selezionare un colore per la documentazione generata direttamente dal file di configurazione agendo sulle seguenti proprietà:

- HTML_COLORSTYLE_HUE
- HTML_COLORSTYLE_SAT
- HTML_COLORSTYLE_GAMMA

Ovviamente l'intera documentazione avrà le caratteristiche inserite nel file di configurazione. Non è quello che vogliamo. Dunque l'idea è stata quella di inserire codice *css* all'interno dei commenti fatti sulle entità, circondando con un tag $< span >$ le informazioni aggiunte come commenti alle entità. Il tag *span* conteneva il colore in formato *RGB*. Avendo da subito raccolto le informazioni nell'oggetto ODD facendo distinzione tra le informazioni generate automaticamente e quelle prese in input dall'utente, è stato possibile selezionare colori diversi per questi due livelli di informazioni. Quello che però accadeva è che nella documentazione compariva il tag $< span >$. Questo perchè *Doxygen* rendeva visibile il tag aggiungendo le sequenze di escape *&lt.span*. Fortunatamente *Doxygen* consente di aggiungere i tag *htmlonly* e
endhtmlonly che lasciano il contenuto inalterato; possiamo quindi inserire codice *html, css* liberamente; tale codice verrà interpretato in maniera opportuna dal browser. La figura 6.1 mostra la schermata di MAD in cui si richiedono all'utente le informazioni necessarie per l'utilizzo di *Doxygen* tra cui anche i colori per l'output prodotto.

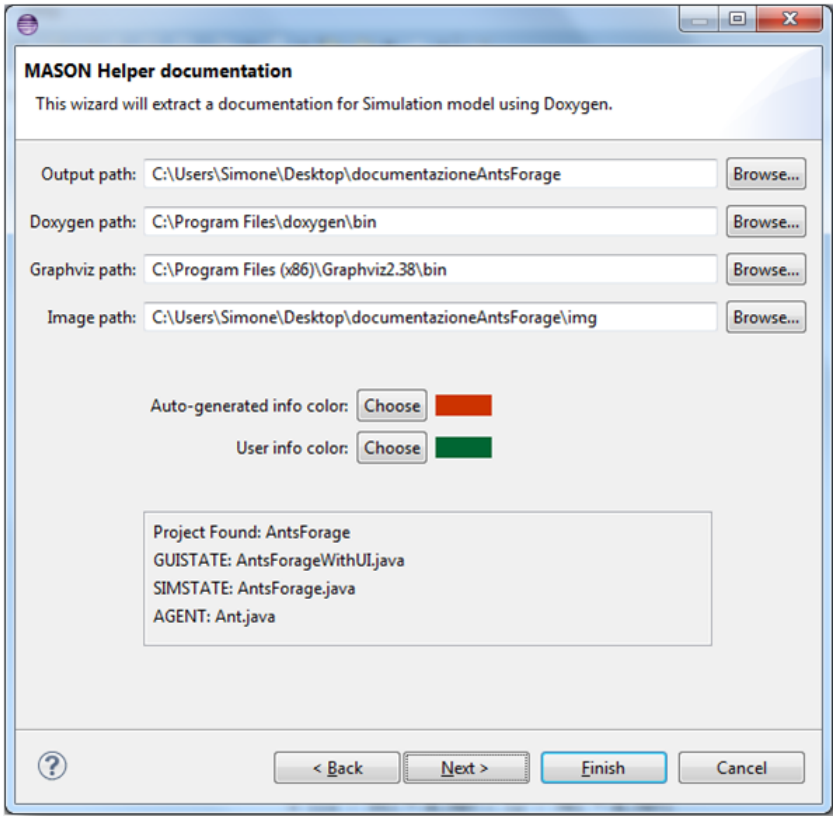


Figura 6.1: MAD: doxygen settings.

6.1.6 Files di configurazione

MAD ha necessità(come detto in 6.1.3) di serializzare l'oggetto ODD. Oltre all'oggetto ODD serializzato MAD genera anche un file di configurazione contenente le seguenti informazioni:

typeOutput=pdf [1]
output=outputPath
doxygenPath=doxygenPath
graphvizPath=graphvizPath
imgPath=imgPath
R_Value_auto=x
G_Value_auto=y

[1]Nella terza versione di MAD è stata aggiunta la possibiltà di scegliere tra diversi tipi di output (vedi 6.2); questa proprietà contiene l'ultima modalità selezionata dall'utente.

B_Value_auto=z
R_Value_user=x1
G_Value_user=y1
B_Value_user=z1

Come si intuisce vengono salvate le informazioni inserite dall'utente riguardo le *path* di Doxygen e Graphviz, una directory contenente le immagini eventuali da inserire nella documentazione ed i valori RGB per le informazioni generate automaticamente e quelle inserite dall'utente. Inoltre ad ogni esecuzione di MAD viene generato un file di *log* (vedi 6.3) contentente tutte le informazioni di ciò che avviene durante la esecuzione. Questi file(*configFile, log, oggetto ODD serializzato*) vengono salvati in una directory creata appositamente per ogni progetto che andiamo a documentare. Verrà creata, alla prima esecuzione di MAD, una directory all'interno della radice del disco; per ogni progetto documentato verrà creata una sottodirectory contenente i file sopra elencati(vedi 6.2).

6.2 Terza versione - Output type

Come visto in 6.1.3 la seconda versione puntava a slegarsi dal tool Doxygen. Per provare questa cosa, la terza versione di MAD aggiunge la possibilità di produrre documentazione anche senza utilizzare Doxygen. in Particolare è stata aggiunta la possibilità di scegliere fra tre modalità di output (vedi 6.4):

- Doxygen
- pdf
- txt
- All

Il processo è identico per tutte le scelte di output; le informazioni raccolte nell'oggetto ODD sono quindi le stesse. Non dimentichiamo però che Doxygen aggiunge alcune informazioni aggiuntive alla documentazione con un opportuno file di configurazione. Inoltre aggiunge alcuni grafici interessanti(come spiegato in 3) ed aggiunge la documentazione della struttura del codice (classi, package...). Selezionando *All* verranno generati tutti i possibili output. La libreria utilizzata per la generazione del pdf è stata *iText* [7].

Flusso degli eventi in MAD

La figura 6.5 mostra il flusso degli eventi dell'applicazione. La fase iniziale contiene un'introduzione al wizard dove è spiegata qual è la funzione che questo svolge. Successivamente si sceglie il tipo di output desiderato. A questo punto si controlla se esiste un oggetto ODD serializzato in precedenza per questo progetto. Si parte quindi con la raccolta di informazioni(automatica con supporto dell'utente) puntando a creare(o eventualmente aggiornare) l'oggetto ODD. A questo punto si genera l'output in base alla scelta effettuata dall'utente nella fase iniziale.

Commenti

Alla fine della procedura guidata MAD genera quindi una serie di commenti al codice in stile Doxygen. Tali commenti intaccano la pulizia del codice. Per risolvere tale problema, alla fine della procedura i commenti generati da MAD vengono eliminati ed il codice ritorna allo stato iniziale. E' tuttavia possibile conservare una copia di backup del codice contenente i commenti generati in automatico spuntando la casella *Save backup of commented code* nell'ultima pagina del wizard. Verrà creata una cartella nella directory contenente le copie dei files sorgenti con estensione *._MAD* (6.2).

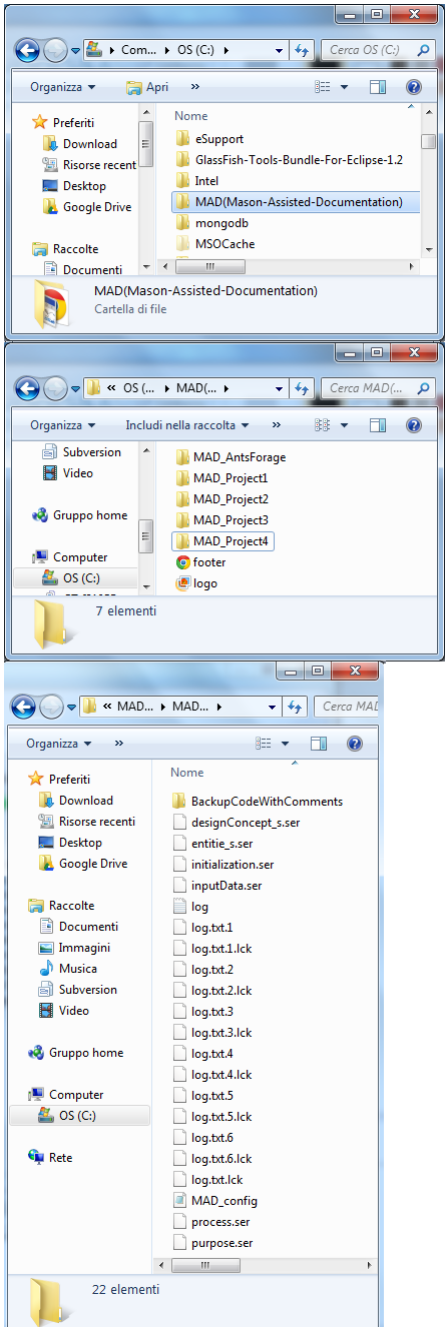


Figura 6.2: MAD: Struttura dei files di configurazione.

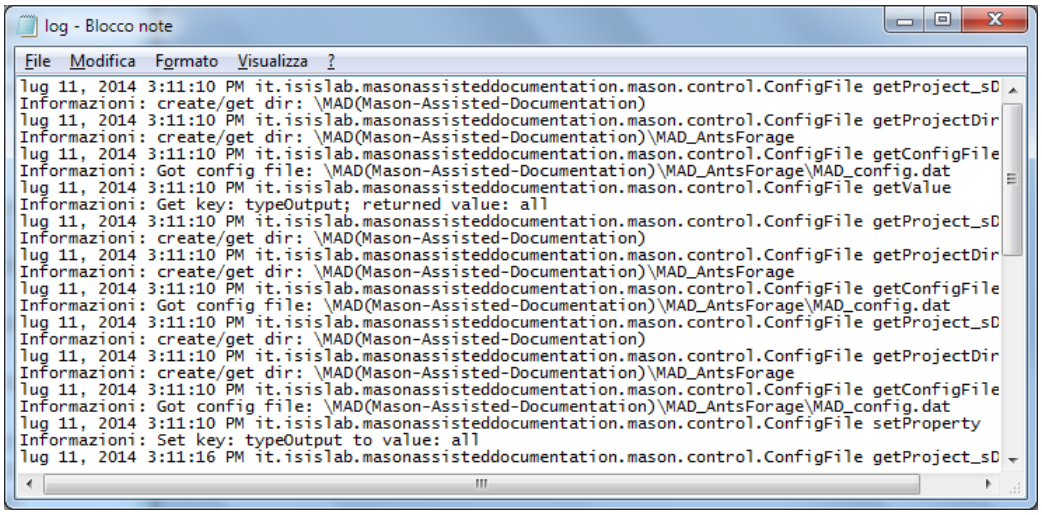


Figura 6.3: Log file.

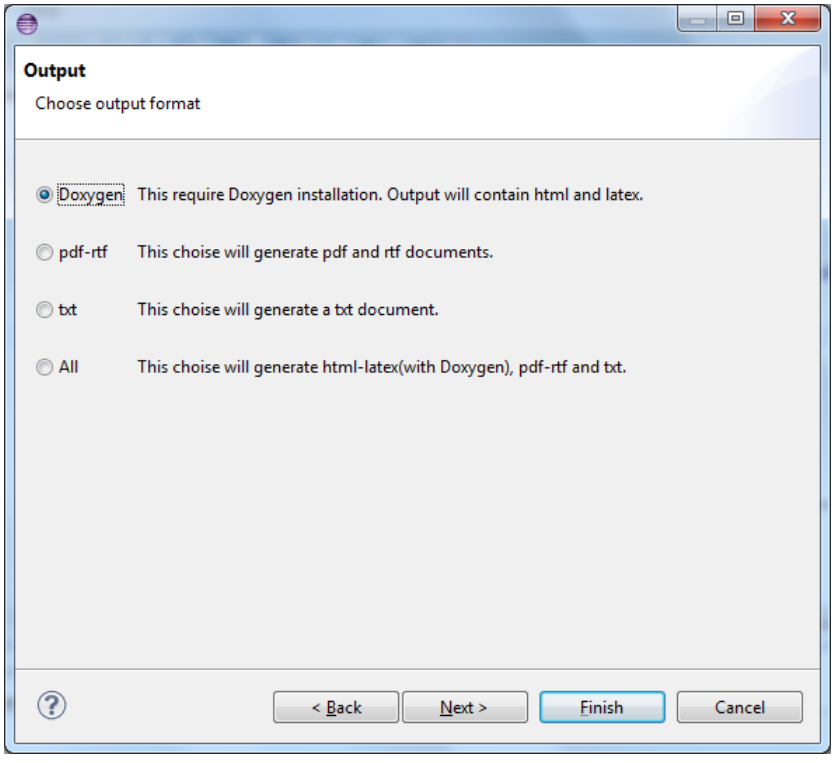


Figura 6.4: Schermata di scelta dell'output in MAD

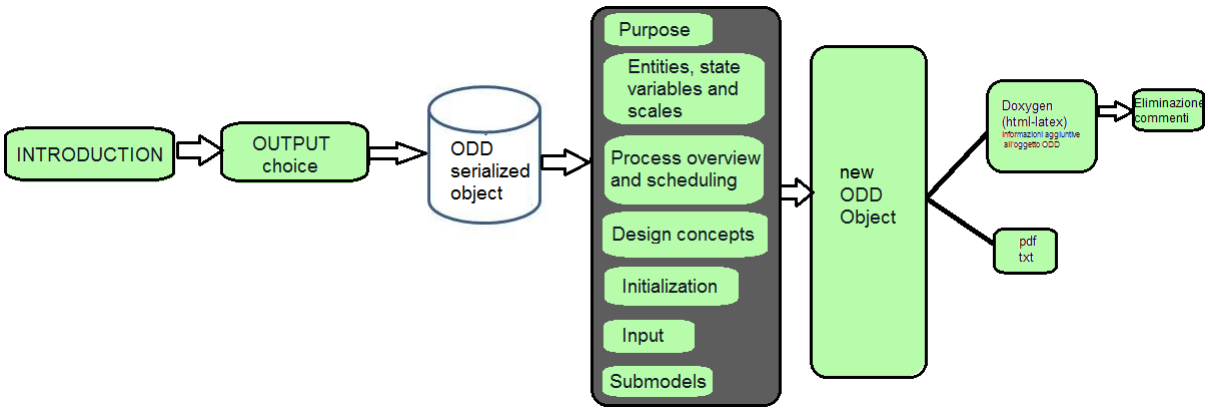


Figura 6.5: Flusso degli eventi in MAD

Capitolo 7

Implementazione

Introduzione

In questo capitolo daremo qualche dettaglio implementativo di MAD. In particolare vedremo la struttura del progetto (packages e classi), le librerie utilizzate e commenteremo alcuni frammenti di codice. Per concludere parleremo di possibili sviluppi futuri.

7.1 Librerie utilizzate

Le librerie che sono risultate utili per il raggiungimento del nostro obiettivo sono:

- AST(Abstract synta tree) [6]
- iText [7]

7.1.1 AST(Abstract syntax tree)

La libreria AST è una libreria di Eclipse, utilizzata all'interno della IDE per il *Refactoring, Quick Fix and Quick Assist*. Consente, partendo da codice java, di ottenere un albero che possiamo quindi navigare, modificare e riscrivere. Le sue potenzialità hanno consentito di navigare i sorgenti java per estrapolare tutte le informazioni richiesti e, nel caso dell'utilizzo di Doxygen, di inserire commenti alle entità. La figura 7.1 mostra il workflow di tale libreria.

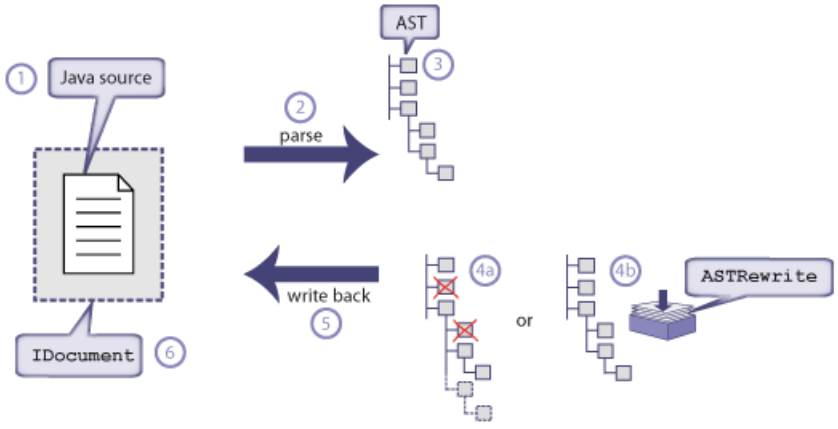


Figura 7.1: Workflow della libreria AST.

7.1.2 iText

La libreria iText è una libreria che supporta diversi tipi di linguaggio e consente in maniera agevole di generare file in formato pdf ed rtf. Possiamo generare tali agendo su oggetti come *Chunk, Phrase, Parahraph, List, Image* che astraggono al programmatore molte difficoltà.

7.2 Struttura

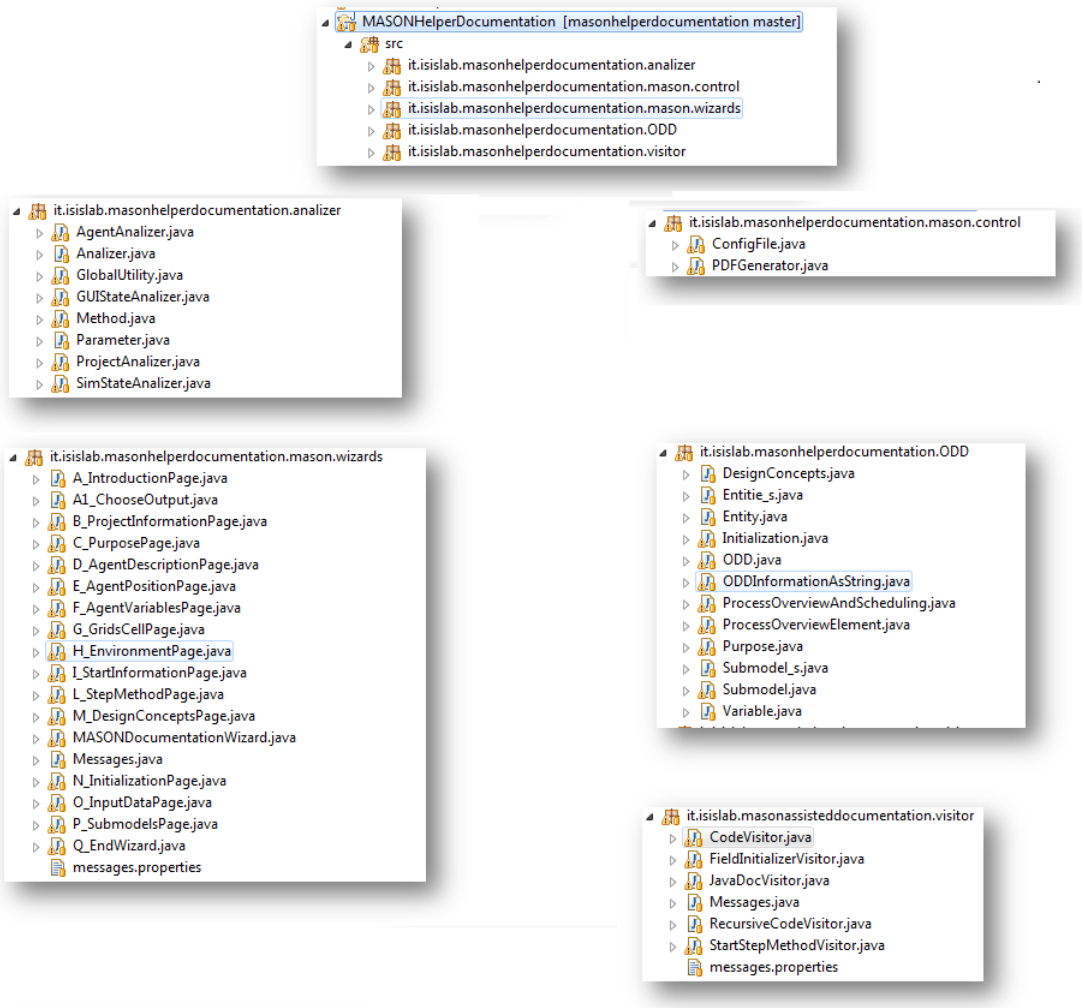


Figura 7.2: Struttura del progetto.

La figura 7.2 riassume mostra la struttura del progetto. Quest'ultimo è suddiviso in cinque packages:

1. it.isislab.masonhelperdocumentation.analizer
2. it.isislab.masonhelperdocumentation.mason.control

3. it.isislab.masonhelperdocumentation.mason.wizards
4. it.isislab.masonhelperdocumentation.ODD
5. it.isislab.masonhelperdocumentation.visitor

Descriviamo di seguito in maniera più dettagliata ciascun package.

7.2.1 Analizer

Le classi presenti nel package *analizer* utilizzano la libreria AST per ricavare informazioni dalle varie classi del progetto che vogliamo esaminare. Le classi principali sono le seguenti:

- ProjectAnalizer
- SimStateAnalizer
- AgentAnalizer

che espongono rispettivamente i metodi per ricavare informazioni riguardanti il progetto, la SimState e gli agenti.

ProjectAnalizer

Vediamo ora le più importanti funzionalità della classe *ProjectAnalizer*. Il metodo *isMasonProject* occupa di capire se il progetto che andiamo ad analizzare è un progetto che utilizza la libreria MASON(teoricamente potremo lanciare MAD su un progetto che non usa tale libreria).

isMasonProject().

```
1   /**
2        * Check if select project is a Mason project and
3        * contains a SimState class.
4        * @return true if Project use a mason.*.jar library
5        *         and contains a SimState class
6        */
7       public boolean isMasonProject() {
8           try {
9               IPackageFragmentRoot[] elements =
10                                      javaProject.getPackageFragmentRoots();
11              for (IPackageFragmentRoot p:elements){
12                  if (p.getElementName().contains("mason")){
13                      log.info("Selected project uses MASON library");
```

```
14                      return true;
15                  }
16              }
17          } catch (JavaModelException e) {
18              log.severe("Exception on getPackageFragmentRoots();
19                                  message: " + e.getMessage());
20              e.printStackTrace();
21          }
22          return false;
23      }
```

Il metodo *getSimStateAndAgent()* ricava le *CompilationUnit* della *SimState* e degli agenti. Di seguito il codice:

getSimStateAndAgent()

```
1   private void getSimStateAndAgent() {
2           try{
3               agent_s_CU = new ArrayList<ICompilationUnit>();
4               IPackageFragment[] packages = javaProject.getPackageFragments();
5               for (IPackageFragment mypackage : packages) {
6                 if (mypackage.getKind() == IPackageFragmentRoot.K_SOURCE) {
7                     for (ICompilationUnit unit : mypackage.getCompilationUnits()) {
8                         IType[] allTypes = unit.getAllTypes();
9                         for(IType type:allTypes){
10                           if (type.getSuperclassName()!=null){
11                                  //class should not extends others
12                                  //class (type.getSuperlassName will be null)
13                                  if (type.getSuperclassName()
14                                              .equalsIgnoreCase("SimState"))
15                                              this.simStateCU = unit;
16                                  if (type.getSuperclassName()
17                                              .equalsIgnoreCase("GUIState"))
18                                              this.guiStateCompilationUnit = unit;
19                           }
20                          String[] interfaces = type.getSuperInterfaceNames();
21                          for (String interfaceName : interfaces){
22                              if (interfaceName.equalsIgnoreCase("steppable")){
23                               this.agent_s_CU.add(unit);
24                              }
25                           }
26                      }
```

```
27                  }
28                }
29              }
30          }catch(JavaModelException e){
31              log.severe("Exception getting SimState,
32                  +Agent and GUIState\n" + e.getMessage());
33              e.printStackTrace();
34          }
35      }
```

SimStateAnalizer

La class *SimStateAnalizer* espone i metodi per ricavare le informazioni dalla *SimState*. Di seguito mostriamo alcuni dei metodi più importanti. Il metodo *getFieldsParameters()* consente di ricavare tutti i *fields* contenuti nella classe *SimState*. Di seguito il codice:

getFieldsParameters()

```
1   /**
2        * This method extracts a list of VariableDeclarationFragment
3        * for grids variables.
4        * @return  ArrayList of grids as VariableDeclarationFragment
5        */
6       public ArrayList<Parameter> getFieldsParameters(){
7           if (GlobalUtility.getAllParameters(compilationUnit) == null)
8                       return null;
9           ArrayList<Parameter> toReturn = new ArrayList<Parameter>();
10          for (Parameter p : GlobalUtility.getAllParameters(compilationUnit)){
11                  for (int i=0; i<simFiledGrid.length; i++){
12                      if (p.getVariableType().equalsIgnoreCase(simFiledGrid[i]))
13                                  toReturn.add(p);
14                  }
15          }
16          return toReturn;
17      }
```

Il metodo statico *GlobalUtility.getAllParameters(compilationUnit)*, presa in input la *compilationUnit* della *SimState* restituisce una lista di tutti i parametri contenuti nella classe. A questo punto si considera il tipo della variabile e, se questo coincide con il tipo di qualche *field* allora viene aggiunto alla lista da restituire.

GlobalUtility

La classe *GlobalUtility* espone diversi metodi utili. Il metodo *getAllParameters(CompilationUnit cu)* consente, data una *CompilationUnit*, di avere una lista di tutti i parametri in essa contenuti. Di seguito il codice:

getAllParameters(CompilationUnit cu).

```
1   /**
2        * Return all parameter in CompilationUnit
3        * @param cu
4        * @return
5        */
6       public static ArrayList<Parameter> getAllParameters(CompilationUnit cu){
7           ArrayList<Parameter> toReturn = new ArrayList<Parameter>();
8           List<TypeDeclaration> types = cu.types();
9           TypeDeclaration classOfCU = types.get(0);
10          FieldDeclaration[] fields = classOfCU.getFields();
11          for (FieldDeclaration f:fields){
12              List<FieldDeclaration> fragments = f.fragments();
13              for (Object o : fragments){
14                  if (o instanceof VariableDeclarationFragment){
15                      VariableDeclarationFragment variable =
16                              (VariableDeclarationFragment) o;
17                      toReturn.add(new Parameter(f, variable));
18                  }
19              }
20          }
21          return toReturn;
22      }
```

Il metodo *getAllMethods(CompilationUnit cu)* consente di ricavare una lista di tutti i metodi contenuti in una *CompilationUnit*. Il codice:

getAllMethods(CompilationUnit cu).

```
1       /**
2        * Return Method list from a CompilationUnit.
3        * @param cu    CompilationUnit
4        * @return  ArrayList<Method>
5        */
6       public static ArrayList<Method> getAllMethods(CompilationUnit cu){
7           ArrayList<Method> toReturn = new ArrayList<Method>();
8           List<TypeDeclaration> types = cu.types();
9           TypeDeclaration classOfCU = types.get(0);
```

```
10          MethodDeclaration[] methods = classOfCU.getMethods();
11          for (MethodDeclaration m : methods){
12              toReturn.add(new Method(m));
13          }
14          return toReturn;
15      }
```

7.2.2 Control

Questo package contiene le classi:

- ConfigFile
- PdfRtfGenerator

La classe *ConfigFile* gestisce il file di configurazione descritto in 6.1.6. Per quanto riguarda la classe *PdfRtfGenerator* è quella che si occupa di generare output in formato pdf ed rtf partendo dall'oggetto ODD; utilizza a tale scopo la libreria *iText*.

7.2.3 Wizard

Questo package contine la parte grafica dell'applicazione; ciascuna classe presente in questo package corrisponde ad una pagina del wizard.

7.2.4 ODD

Questo package contiene l'oggetto ODD A.1.1 e tutti gli oggetti che collaborano con esso. Abbiamo un oggetto per ciascun elemento del protocollo(eccetto per la sezione *input data* le cui informazioni sono contenute in una stringa all'interno dell'oggetto ODD). L'oggetto ODD contiene una variabile per ciascun elemento del protocollo ed espone fondamentalmente metodi per accere a tali variabili e metodi per la serializzazione-deserializzazione dell'oggetto. La figura 7.3 mostra il diagramma delle collaborazioni dell'oggetto ODD generato con Doxygen.

7.2.5 Visitor

Il package *visitor* contiene le classi:

- CodeVisitor A.1.4
- FieldInitializerVisitor A.1.2

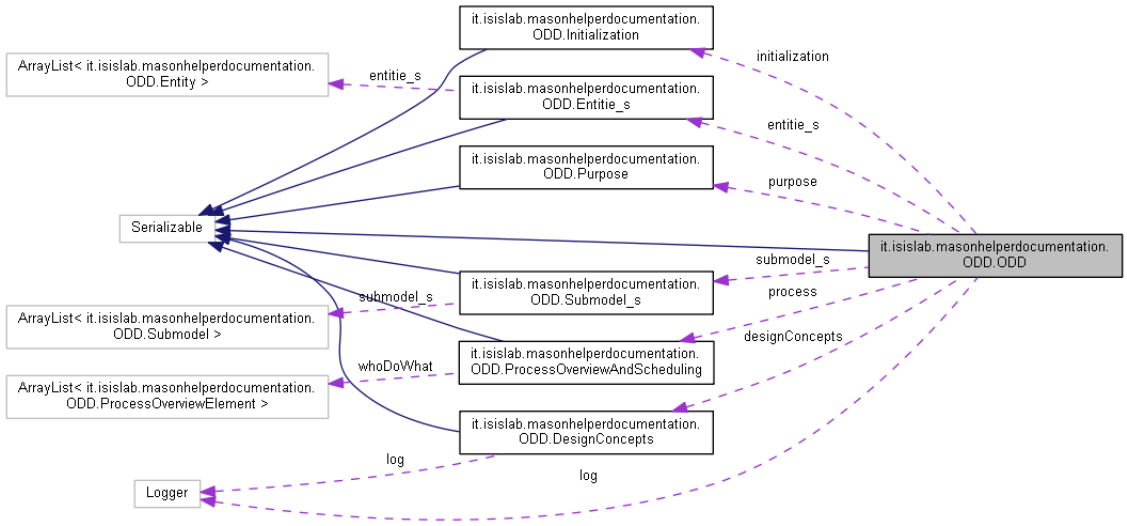


Figura 7.3: Diagramma delle collaborazioni dell'oggetto ODD.

- JavaDocVisitor A.1.3
- RecoursiveCodeVisitor A.1.6
- StartStepMethodVisitor A.1.5

La figura 7.4 mostra la gerarchia delle classi di questo package. La classe astratta *CodeVisitor*A.1.4 si occupa di visitare la *CompilationUnitestende* su cui è invocato il metodo *accept*. Il metodo

```
1       public abstract boolean visit(MethodInvocation node);
```

è implementato dalle classi *StartStepMethodVisitor* e *RecoursiveCodeVisitor* in maniera diversa(vedi A.1.5 A.1.6) che estendono *CodeVisitor*.

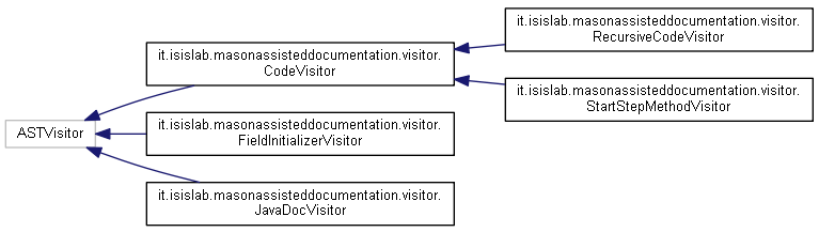


Figura 7.4: Gerarchia delle classi del package *visitor*.

7.3 Sviluppi futuri

Internazionalizzazione

Tra le funzionalità da supportare sicuramente c'è l'internazionalizzazione. Attualmente MAD utilizza file di *properties* contenenti porzioni di frasi da comporre(attualmente in lingua inglese). Bisognerebbe aggiungere file di *proerties* per diversi linguaggi ed aggiungere la gestione di tali file.

Submodels

Come visto in 5.3 la qualità delle informazioni generate durante per la fase di *Process overview and scheduling* dipende molto dallo stile del programmatore. Potremo risolvere tale problematica consentendo all'utente di organizzare i metodi *start()* e *step()* da interfaccia grafica, selezionando porzioni di codice che rappresentano *submodels*.

Altro

Utilizzando il plugin su diversi esempi forniti dalla libreria MASON è emerso quanto segue:

- MAD non supporta progetti che non contengono una classe che estende *SimState* il che è possibile(possiamo creare una simulazione contenente ad esempio solamente l'interfaccia grafica)
- si può migliorare la classe *StartStepMethodVisitor*; in particolare il metodo *public boolean visit(MethodInvocation node)* attualmente supporta solo alcuni dei metodi forniti dalla libreria MASON(*scheduleRepeating*, *setObjectLocation*, *getObjectLocation*, *multiply*, *nextDouble*, *nextInt*, *addEdge*, *addNode*, *getAllObjects*); laddove l'invocazione a metodo non corrisponde a nessuno dei metodi 'noti', viene creato un *submodels*. Se questo submodels è semplicemente un altro dei metodi noti di MASON, la sezione *Submodels* non lo conterrà(la visita di tale *submodel* non darà informazioni).

Capitolo 8

Manuale d'uso

1

Introduzione

In questo capitolo mostriamo un esempio di download e di utilizzo di MAD. Quello che vedremo quindi è la generazione della documentazione per l'esempio *AntsForage* fornito con la libreria MASON. Saranno mostrate e commentate le varie fasi della procedura guidata.

8.1 Manuale utente MAD

8.1.1 Download

MAD può essere installato da Eclipse accedendo alla funzionalità

help->Install new software

ed inserendo il link [8].

8.1.2 Utilizzo

Per utilizzare quindi il plug-in possiamo cliccare col tasto destro sul progetto che vogliamo documentare, cliccare su *export* e scegliere la voce *MAD(Mason Assisted Documentation)*(vedi 8.1).

La prima schermata mostrata è quella mostrata in figura 8.2.

Tale pagina riassume la funzionalità del plug-in. Successivamente viene mostrata la pagina 8.3 dove possiamo decidere quale output vogliamo produrre(scegliendo *All* verranno prodotti tutti i possibili output).

Una volta selezionato l'output viene mostrata la schermata 8.4. In tale schermata l'utente deve inserire:

- Output path: percorso in cui sarà generato l'output
- Doxygen path: percorso in cui è presente il file *bin.** di Doxygen
- Graphviz path: percorso contenente il *bin.** di Graphviz(utilizzato da Doxygen per la generazione dei grafici)
- Image path: percorso della directory contenente eventuali immagini da inserire nella documentazione
- Auto-generated info color: colore delle informazioni generate in automatico
- User info color: colore delle informazioni inserite dall'utente

Nel caso in cui l'output selezionato non è di tipo *Doxygen*, tale schermata sarà quella mostrata in 8.5(alcune delle precedenti informazioni non hanno senso per output diverso da *Doxygen*).

A questo punto MAD comincia a raccogliere informazioni che saranno strutturate secondo il protocollo ODD. La pagina successiva è quindi quella mostrata in 8.6 dove viene chiesto all'utente di inserire informazioni riguardo lo scopo del modello descritto dall'applicazione che sta documentando.

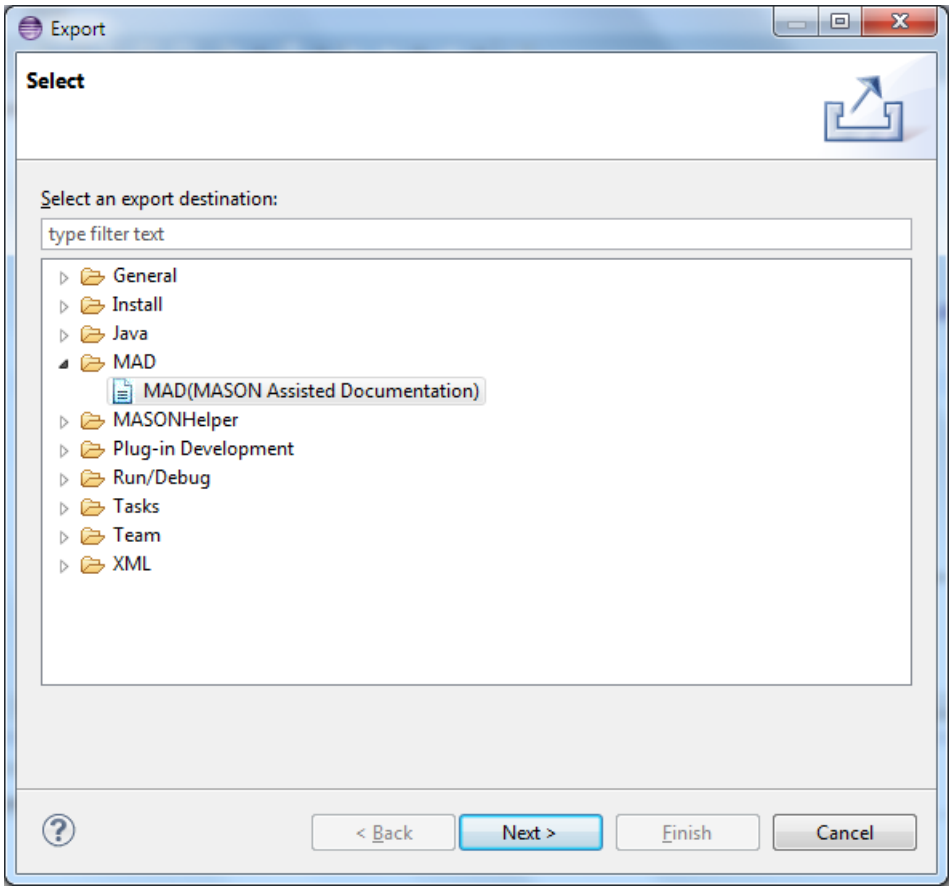


Figura 8.1: MAD export.

Inserito lo scopo del modello MAD comincia a raccogliere informazioni riguardo la sezione *Entities, state variables and scales* del protocollo ODD. Parte con informazioni sugli agenti. Per l'esempio *AntsForage* che stiamo andando a commentare, abbiamo un agente rappresentato dalla classe *Ant.java*. La schermata 8.7 mostra questa fase; l'utente può inserire una descrizione per l'agente.

Successivamente vengono mostrate le variabili che rappresentano la posizione dell'agente. Viene chiesto un commento all'utente e viene generato un commento automatico(vedi 8.8).

La schermata successiva mostra tutte le altre variabili dell'agente; anche qui per ogni variabile viene generato un commento automatico e viene richiesto un commento all'utente(vedi 8.9).

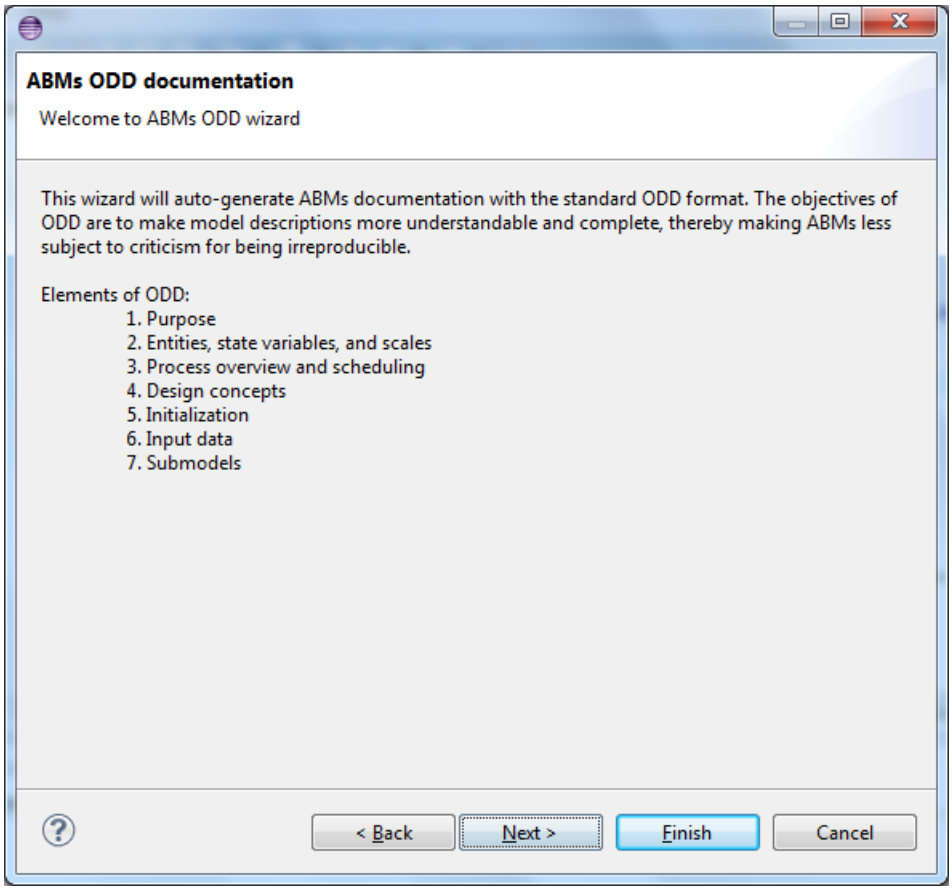


Figura 8.2: Pagina introduttiva.

Si passa quindi ad analizzare l'ambiente della simulazione. La prima pagina che si occupa di analizzare l'ambiente, mostra informazioni sui *field* della simulazione. In particolare per ciascuno genera un commento automatico contenente informazioni sull'ampiezza del *field* e chiede un commento all'utente(vedi 8.10).

La schermata successiva mostra le altre variabili dell'ambiente(vedi 8.11).

A questo punto si comincia a raccogliere informazioni sulla fase *Process overview and scheduling*. Inizialmente si analizza il metodo *start()* dell'ambiente. La pagina dedicata a questa fase riporta il codice del metodo, il commento generato automaticamente e lascia spazio all'utente per inserire ulteriori eventuali informazioni(vedi 8.12, 8.13)

La stessa cosa avviene per il metodo *step()* dell'agente(*Ant* in questo

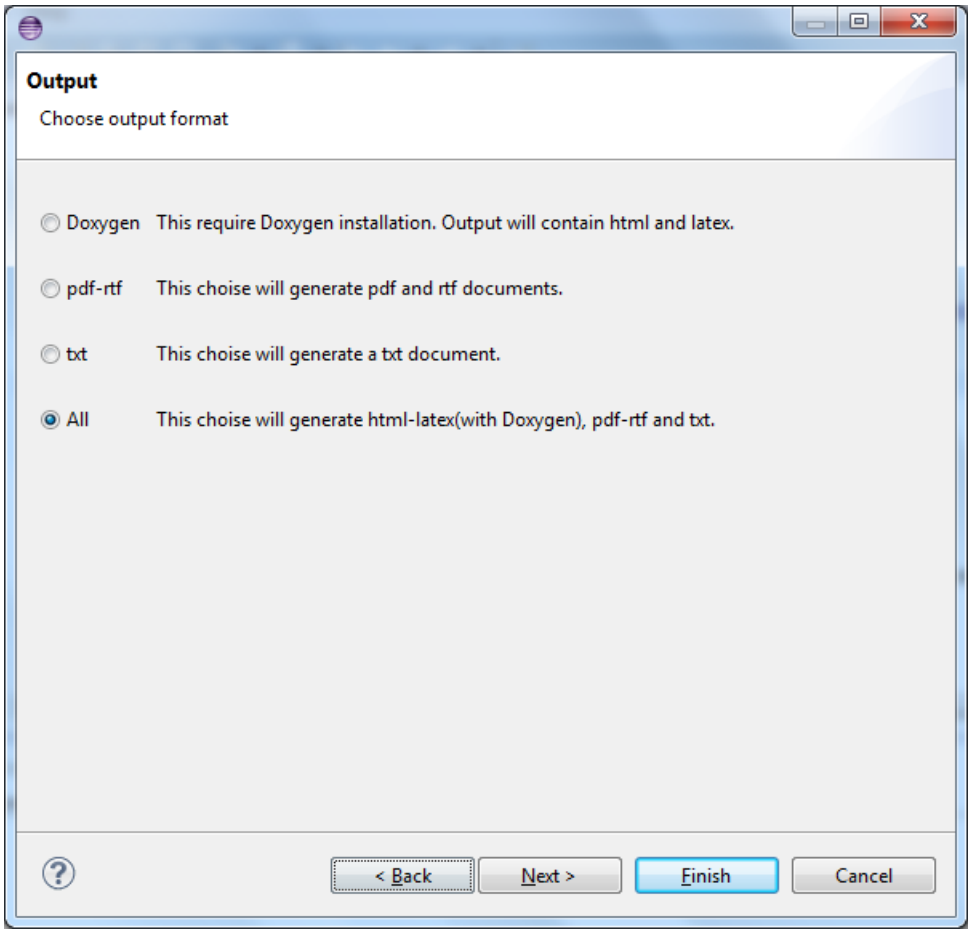


Figura 8.3: Selezione dell'output.

caso(vedi 8.14, 8.15).

La schermata successiva consente all'utente di inserire informazioni riguardo i *Design concepts*(siamo alla fase 4 del protocollo ODD). Per ciascuno dei concetti di design è riportata la domanda a cui bisogna rispondere(vedi 8.16).

A questo punto si passa alla fase *Initialization* di ODD. Viene generato un commento automatico che riporta il valore iniziale di tutte le variabili del progetto che stiamo analizzando raggruppate per classi e viene lasciato spazio per un eventuale aggiunta dell'utente(vedi **??**).

Per quanto riguarda la sezione *Input data* le informazioni vengono richieste all'utente(vedi 8.18).

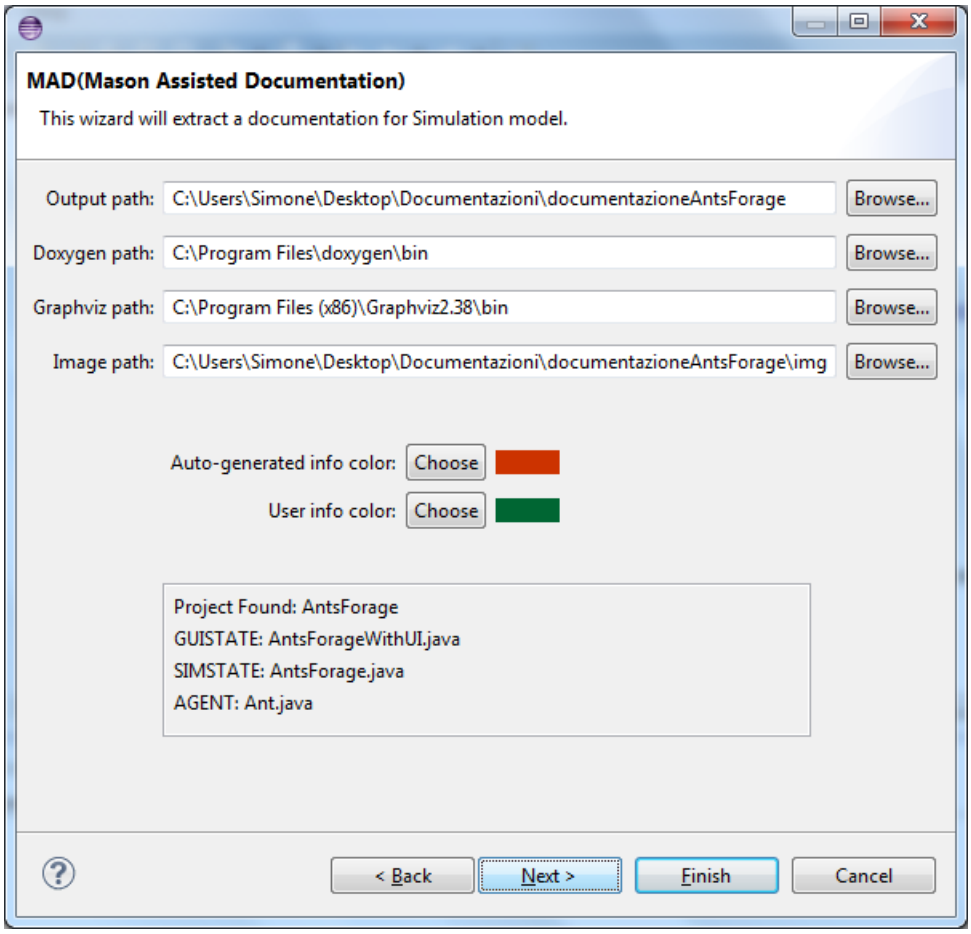


Figura 8.4: Impostazioni iniziali.

Successivamente vengono approfonditi(automaticamente) i *Submodels* individuati nelle fasi di *Process overview and scheduling*(vedi 8.19).

Si arriva quindi alla fase finale in cui possiamo generare la documentazione cliccando sul pulsante *Generate*. La casella *Show Output* consente di mostrare l'output prodotto al termine dell'operazione(vedi 8.20, 8.21). La casella *Save backup of commented copy* consente di salvare una copia di ogni codice sorgente contenente i commenti generati in automatico con estensione *._MAD*. Tali files(insieme ai files descritti in reffig:configFiles saranno creati nella dircetory creata da MAD nella root del disco(su Windows C:\MAD(Mason-Assisted-Documentation)\MAD_AntsForage\BackupCodeWithComments).

La figura 8.22 mostra l'output generato. Il contenuto del pdf è riportato

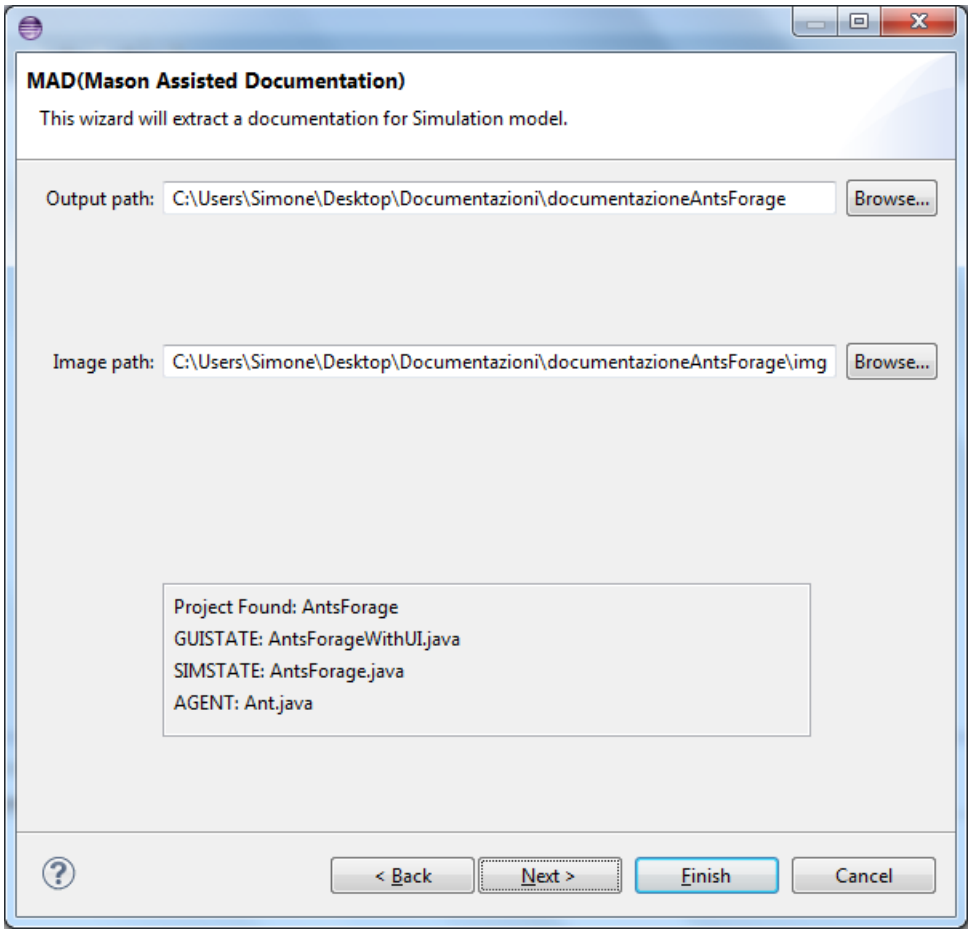


Figura 8.5: Impostazioni iniziali.

in A.2.

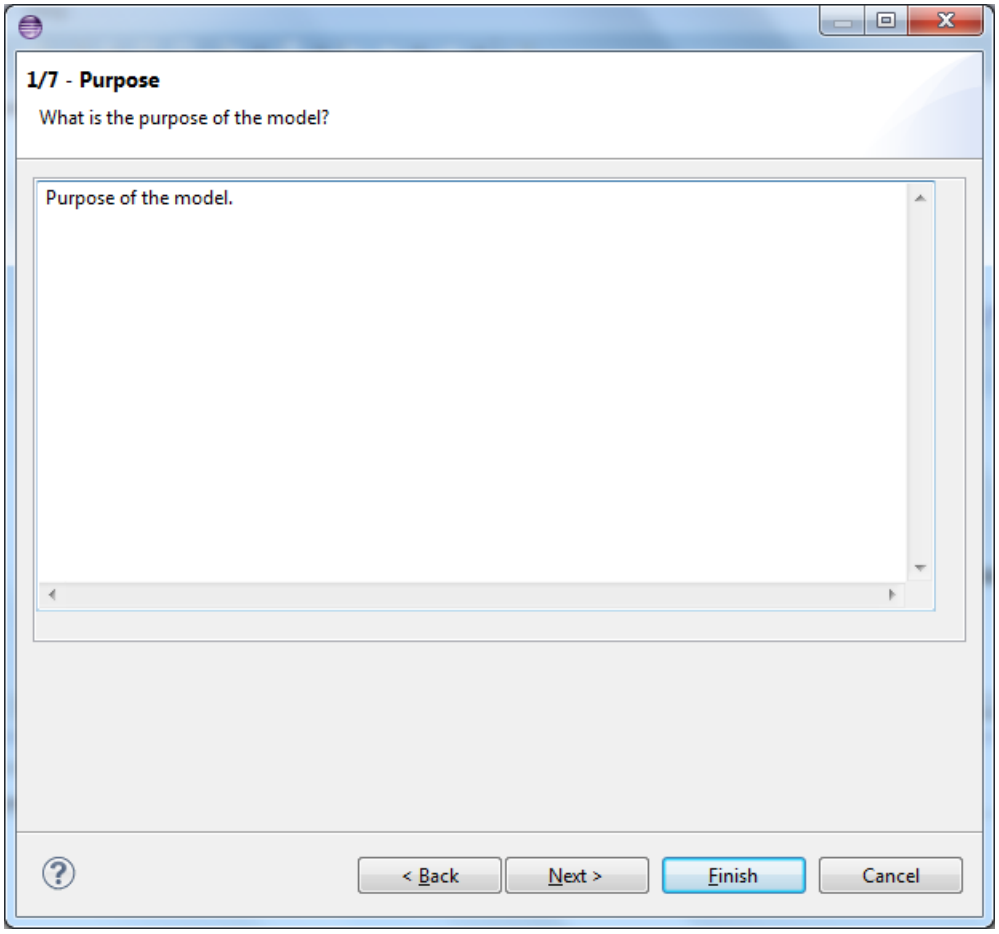


Figura 8.6: Purpose.

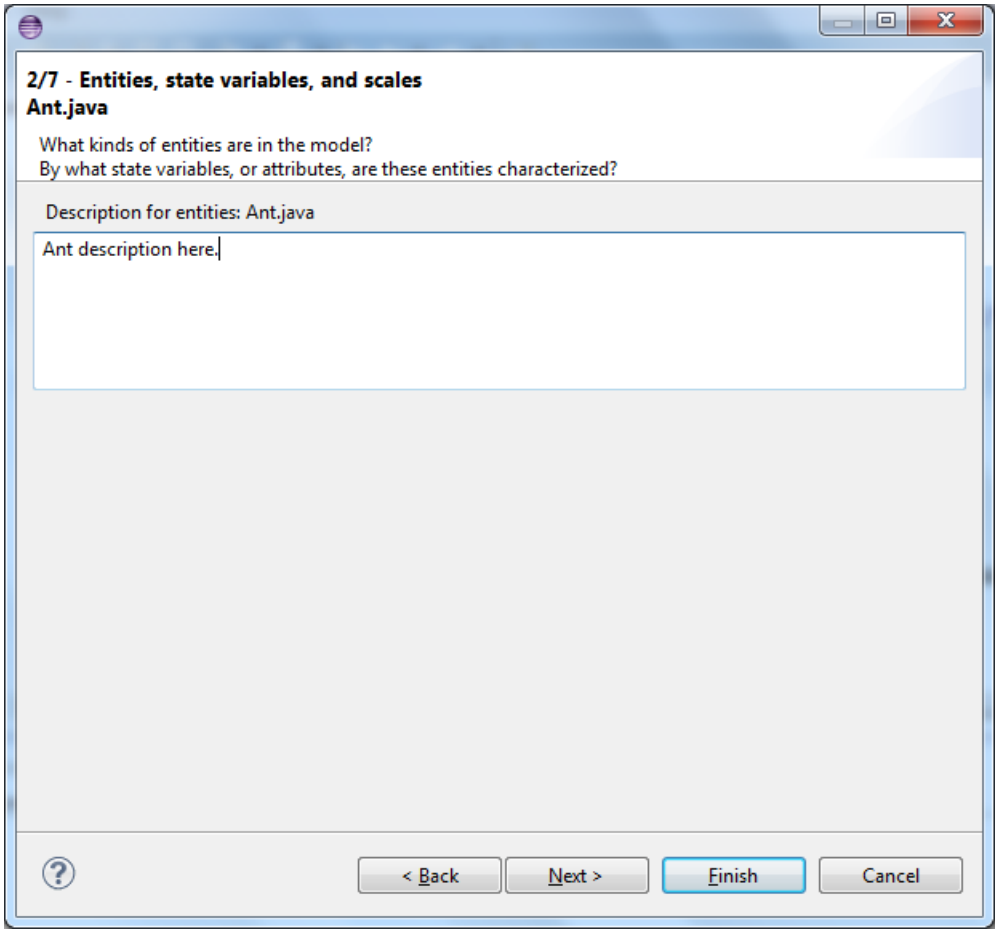


Figura 8.7: Informazioni sull'agente *Ant.java*.

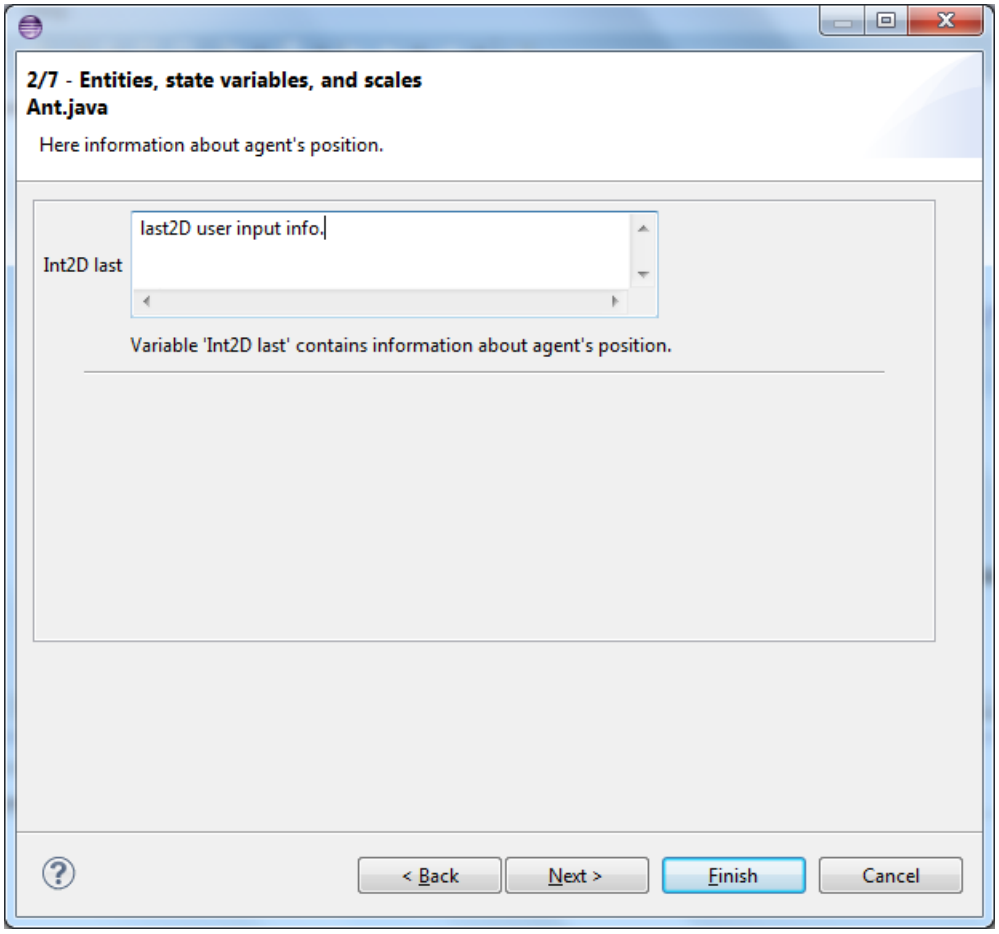


Figura 8.8: Informazioni sull'agente *Ant.java*.

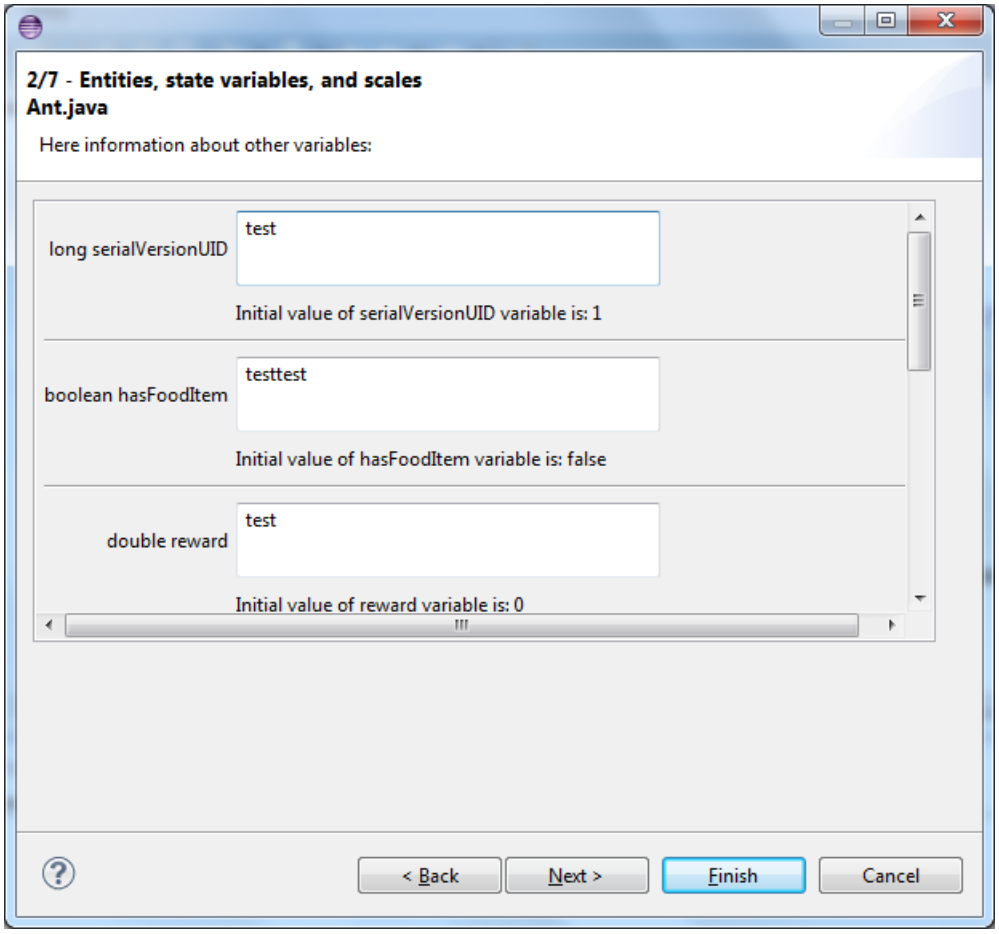


Figura 8.9: Informazioni sull'agente *Ant.java*.

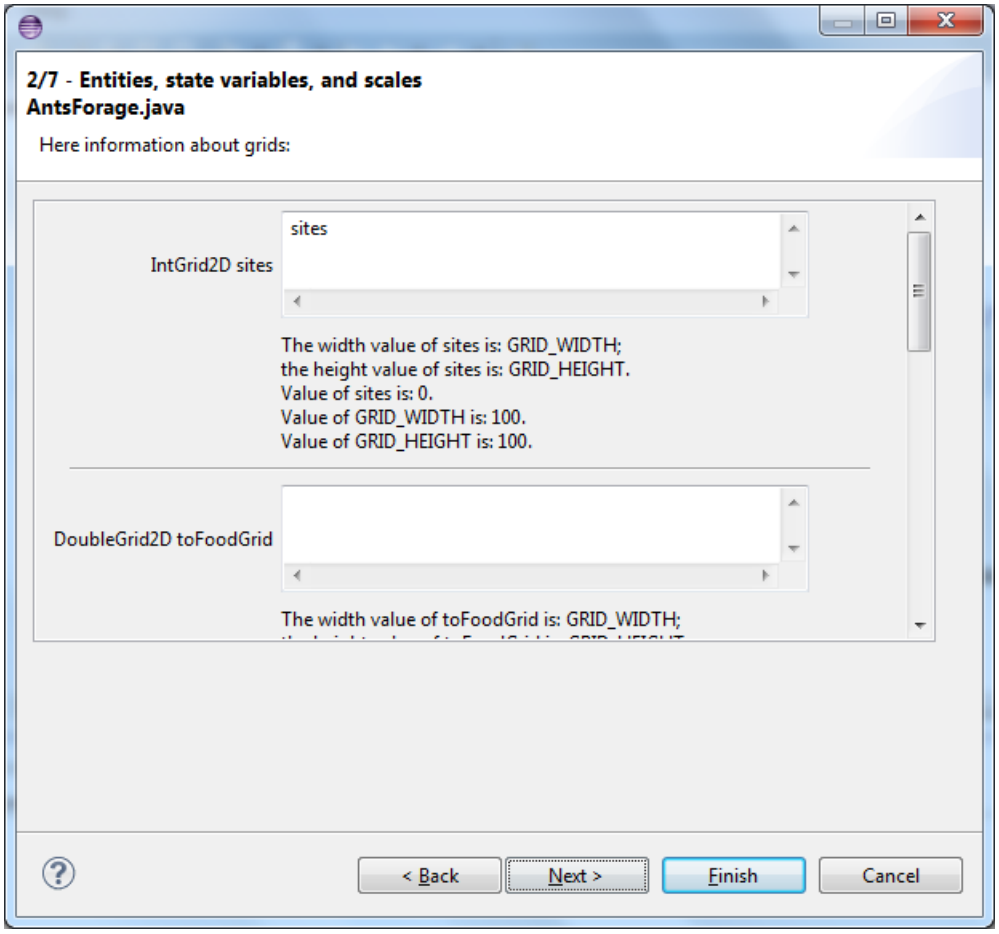


Figura 8.10: Informazioni sui *Fields*.

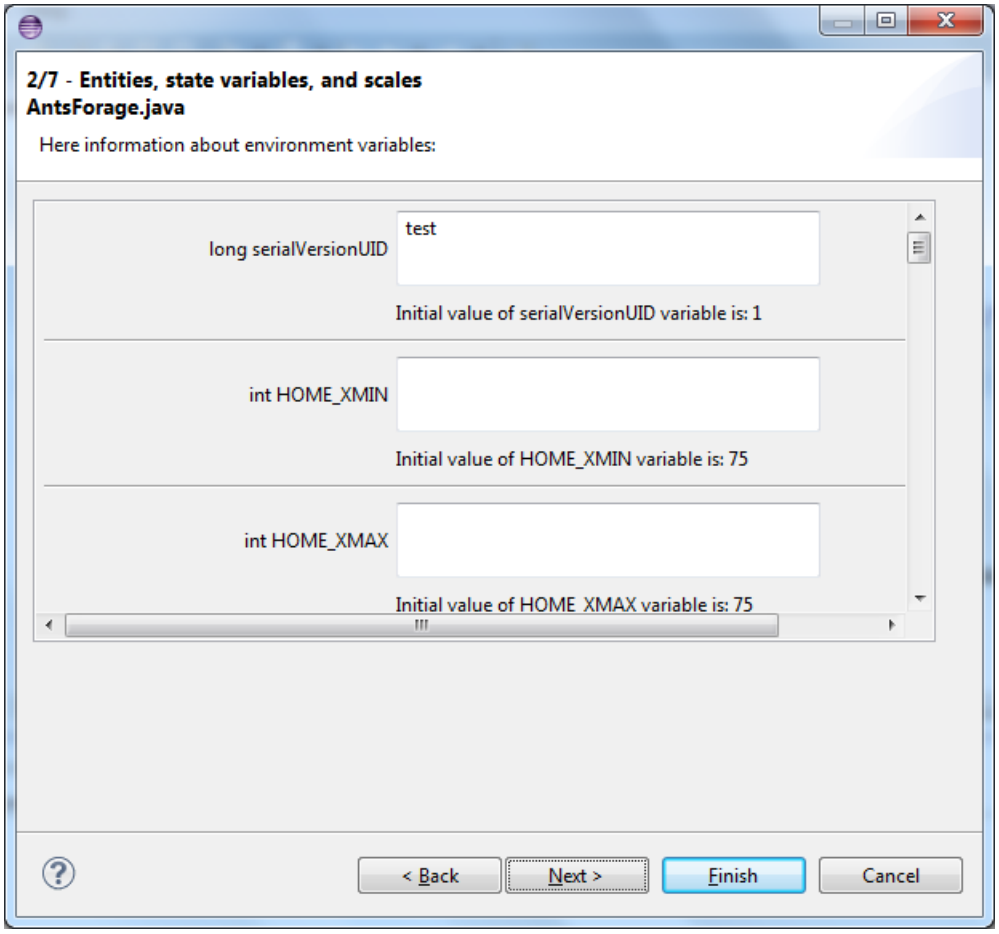


Figura 8.11: Informazioni sulle variabili della *SimState* che non sono *Field*.

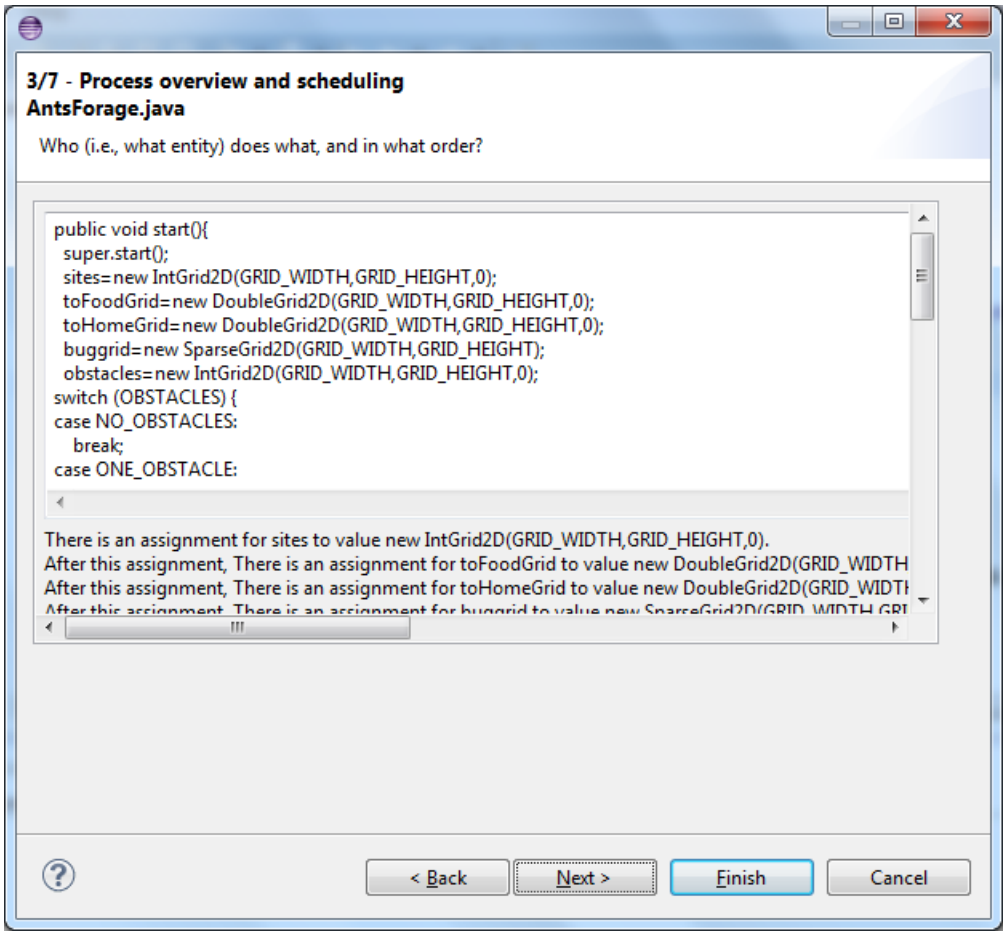


Figura 8.12: Informazioni sul metodo *start()* della *SimState*.

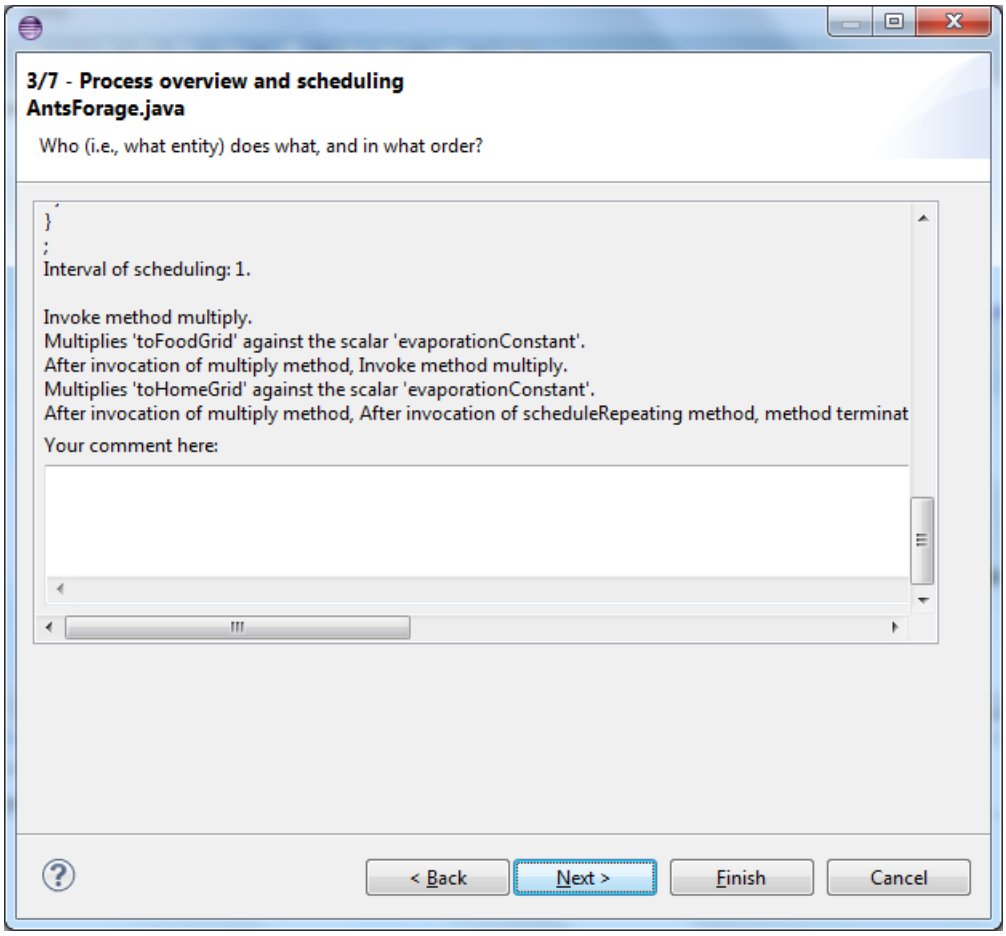


Figura 8.13: Informazioni sul metodo *start()* della *SimState*.

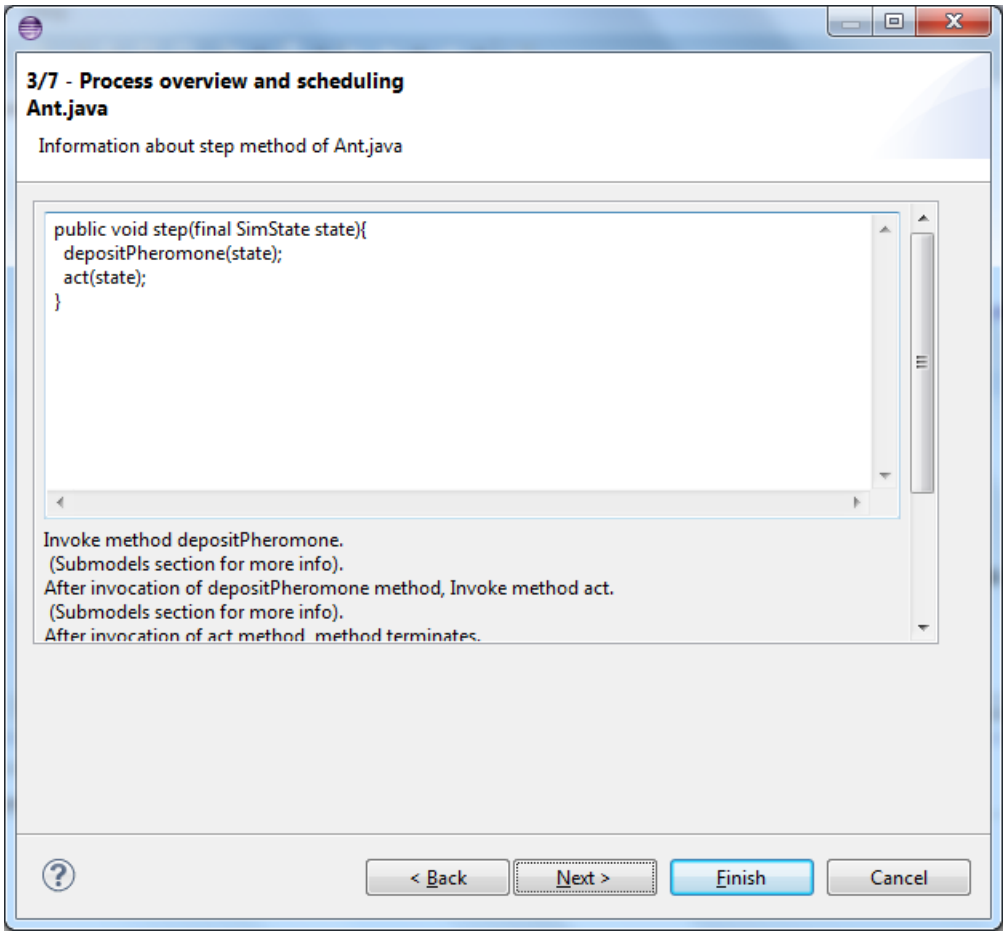


Figura 8.14: Informazioni sul metodo *step()* dell'agente *Ant*.

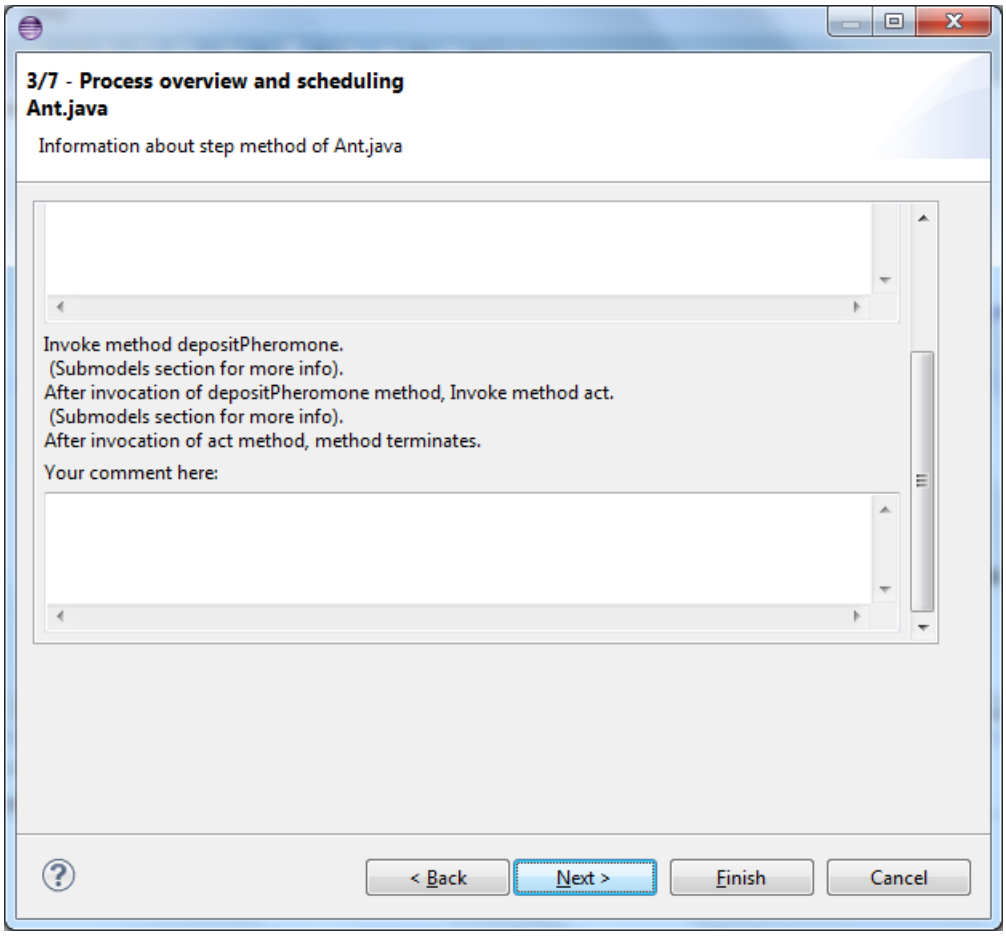


Figura 8.15: Informazioni sul metodo *step()* dell'agente *Ant*.

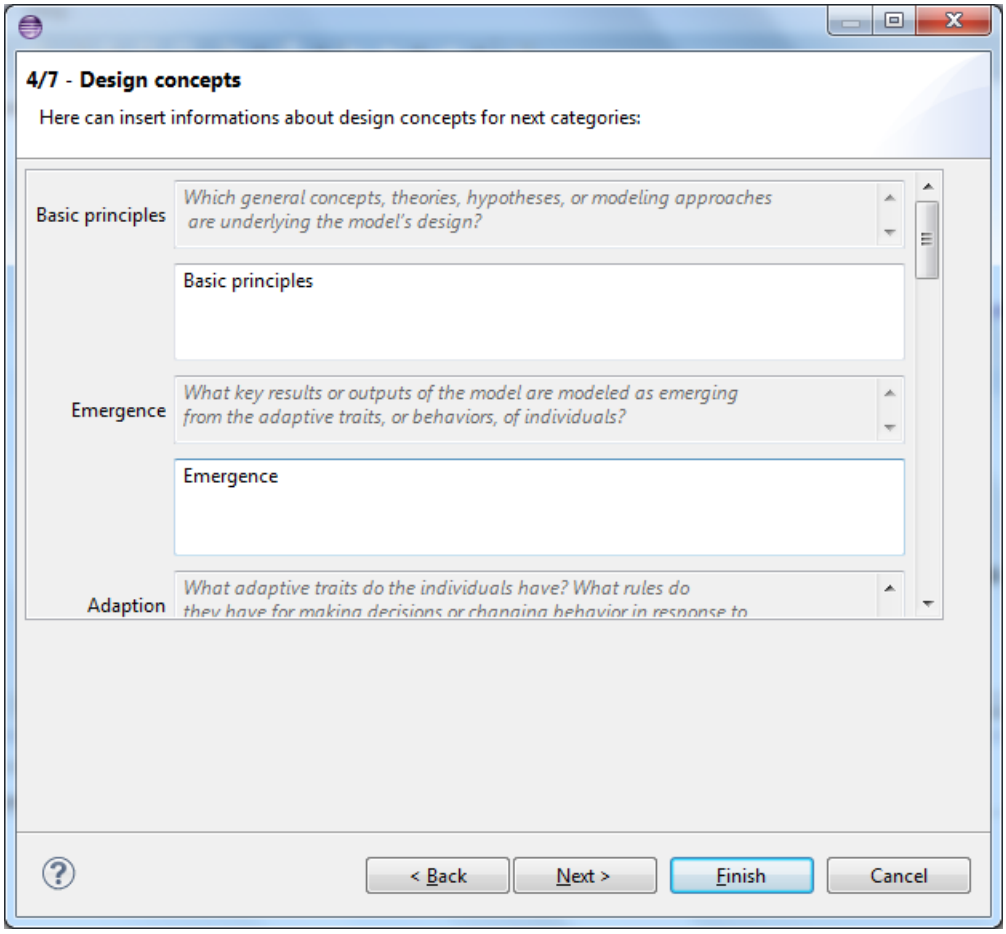


Figura 8.16: Informazioni sui Design concepts.

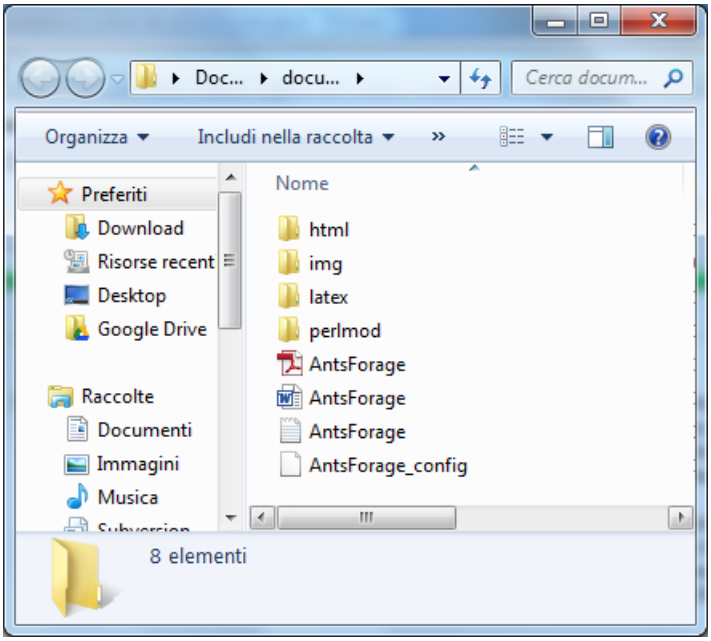


Figura 8.22: Ouput generato.

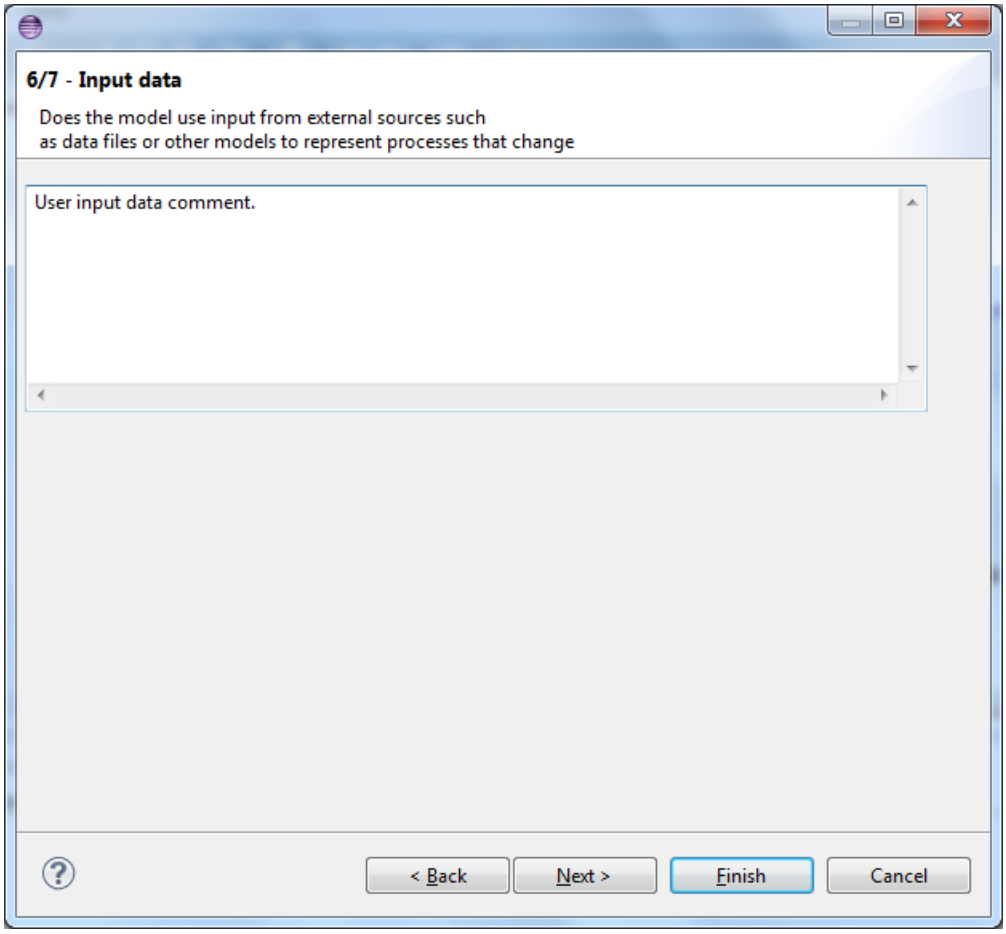


Figura 8.18: Informazioni sull'input.

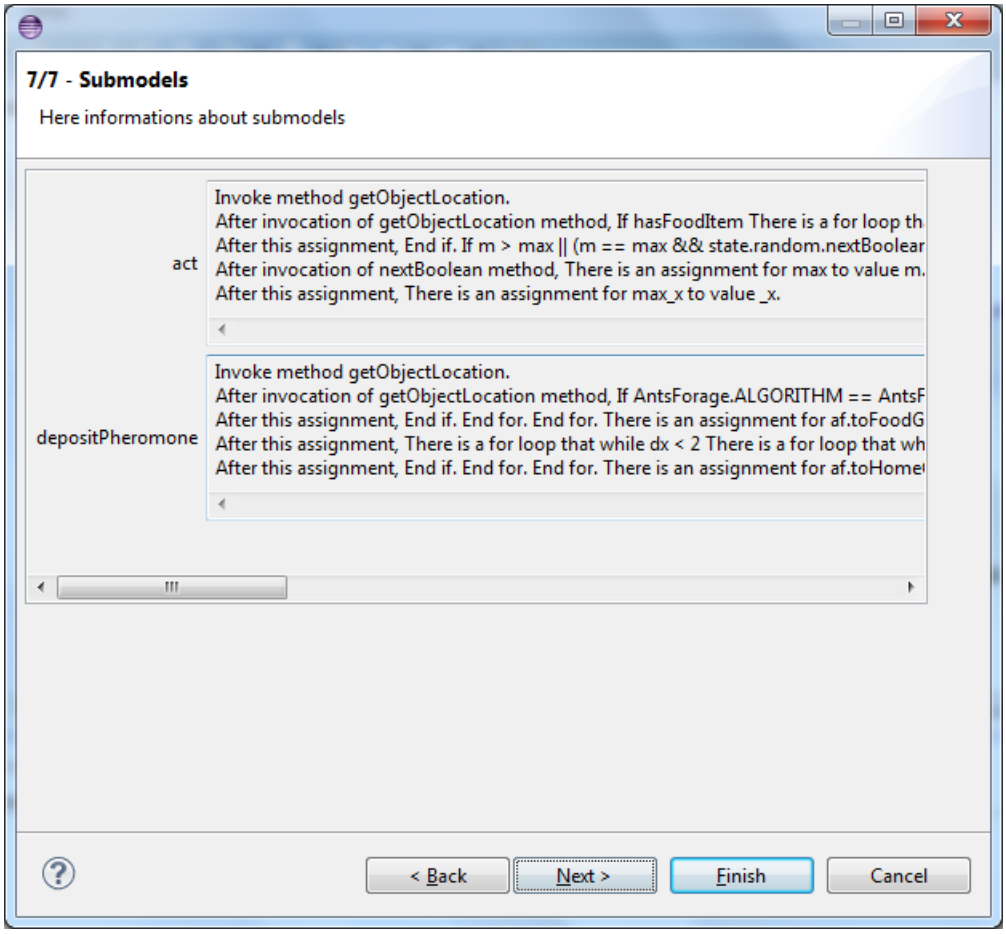


Figura 8.19: Submodels.

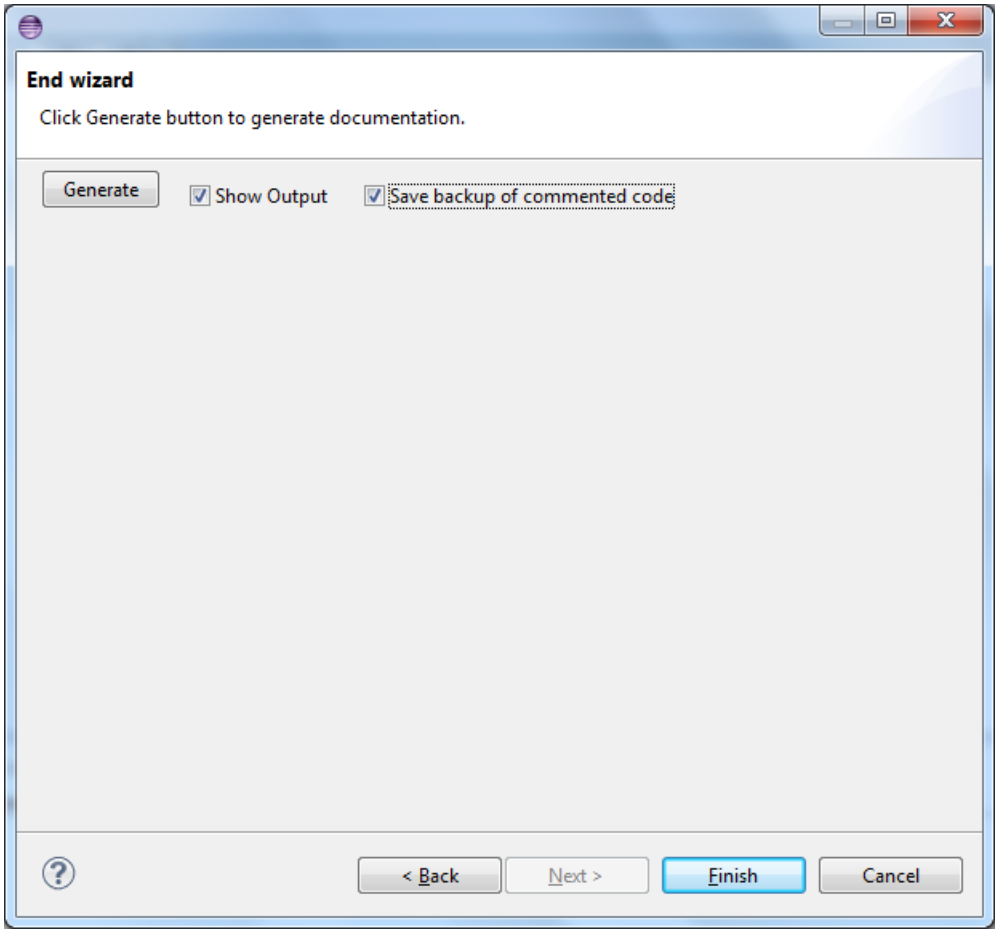


Figura 8.20: Schermata finale.

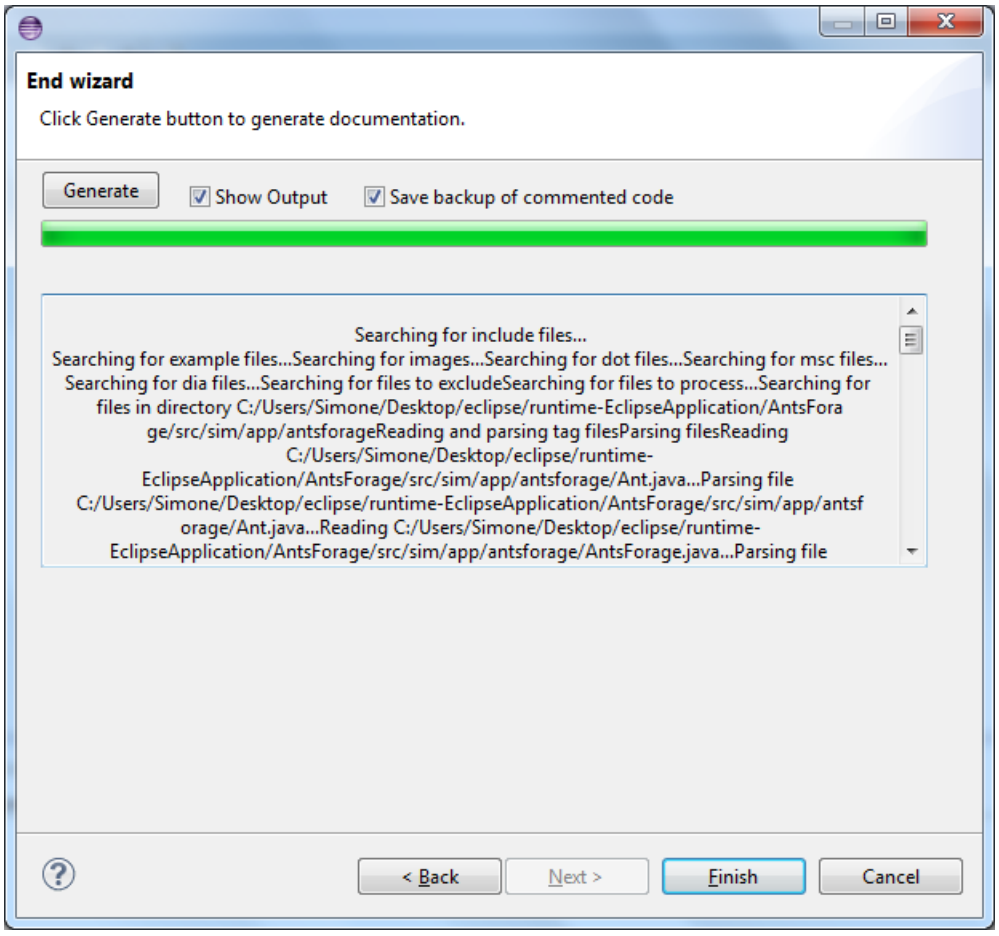


Figura 8.21: Schermata finale.

Capitolo 9

Conclusioni

Possiamo a questo punto trarre delle conclusioni sul lavoro svolto. Riprendiamo da quello che era l'obiettivo iniziale: 'Documentare programmi di simulazione multi-agente scritti in java con libreria MASON'. Il prodotto realizzato e descritto in questo lavoro di tesi produce un risultato esaustivo rispetto all'obiettivo iniziale. MAD supporta l'utente durante la creazione della documentazione per un programma di simulazione multi-agente, generando documentazione che non da la sensazione di un qualcosa di generato in maniera automatica. L'output *racconta* quello che è il modello di simulazione, generando in alcuni punti un discorso in linguaggio naturale che rende l'idea di quello che avviene nel modello. Inoltre la struttura dell'output segue uno standard noto(protocollo ODD [1]) agevolando la diffusione di tale standard. Utilizzando MAD si può uniformare la struttura delle documentazioni di ABMs scritti in java con libreria MASON, supportando quanto illustrato dagli autori in [1]. L'esperto del modello potrà così abituarsi ad uno standard che ritroverà in tutte le documentazioni; il programmatore non dovrà preoccuparsi di strutturare la documentazione, ma dovrà solo contribuire alla raccolta di informazioni effettuata da MAD.

Bibliografia

[1] "ODD(Overview, Design concepts and details) protocol" -Volker Grimma, Uta Bergerb, Donald L. DeAngelisc, J. Gary Polhilld, Jarl Giskee, Steven F. Railsbackf - 'The ODD protocol: A review and first update' - 13-08-2010 http://bio.uib.no/te/papers/Grimm_2010_The_ODD_protocol_.pdf

[2] "Doxygen" http://www.stack.nl/~dimitri/doxygen/

[3] "MASON library" http://cs.gmu.edu/~eclab/projects/mason/

[4] "Eclox" http://home.gna.org/eclox/ 1.2, 1.3, 1.4, 4.3.1, 5, 5.1, 9

[5] "Graphviz" http://www.graphviz.org/ 1.3, 3

[6] "Abstract syntax tree library" https://eclipse.org/jdt/ui/astview/index.php 1.3, 2

3.1.1

[7] "iText library" http://itextpdf.com/ 3.1.1

[8] "MAD(Mason Assisted Documentation)" - Simone Romano http://www.isislab.it/projects/MASONHelperSite

4.3, 6.1.1, 7.1

6.2, 7.1

Appendice A

Appendice

A.1 Listati

A.1.1 L'oggetto ODD

La classe *ODD* si occupa della raccolta delle informazioni, generate in automatico e inserite dall'utente durante il wizard. Fondamentalmente il problema da risolvere con questa classe era legato all'iterabilità del processo; tale oggetto ha in qualche modo svincolato la generazione dell'output dal tool *Doxygen*. Infatti una volta raccolte tutte le informazioni secondo lo standard ODD siamo liberi di generare outuput in qualsiasi altra modalità.

```
1  package it.isislab.masonhelperdocumentation.ODD;
2
3  import it.isislab.masonhelperdocumentation.analizer.GlobalUtility;
4  import it.isislab.masonhelperdocumentation.mason.control.ConfigFile;
5
6  import java.io.FileInputStream;
7  import java.io.FileNotFoundException;
8  import java.io.FileOutputStream;
9  import java.io.IOException;
10 import java.io.ObjectInputStream;
11 import java.io.ObjectOutputStream;
12 import java.io.Serializable;
13 import java.nio.file.Files;
14 import java.nio.file.Paths;
15 import java.util.ArrayList;
16 import java.util.Collection;
```

```
17  import java.util.HashMap;
18  import java.util.logging.Logger;
19
20  import org.eclipse.jdt.core.dom.CompilationUnit;
21
22  /**
23   *
24   * @author Romano Simone 0512101343
25   * This class collect all the informations about
26   * model, grouped by ODD logic.
27   */
28  public class ODD implements Serializable{
29      private static final long serialVersionUID = 1;
30      public static Purpose purpose;
31      public static Entitie_s entitie_s;
32      public static ProcessOverviewAndScheduling process;
33      public static Submodel_s submodel_s;
34      public static DesignConcepts designConcepts;
35      public static Initialization initialization;
36      public static String inputData;
37      private static Logger log = Logger.getLogger("global");
38      public static String getStandardDefinition = "The model description
39              +follows the ODD (Overview, Design concepts, Details)
40              +protocol (Grimm et al., 2006, this work).";
41      private static boolean differentsColors = true;
42
43      public static Purpose getPurpose(){
44          if (purpose == null)
45              purpose = new Purpose();
46          return purpose;
47      }
48
49      public static Entitie_s getEntitie_s(){
50          if (entitie_s == null)
51              entitie_s = new Entitie_s();
52          return entitie_s;
53      }
54
55      public static ProcessOverviewAndScheduling getProcessOverviewAndScheduling(){
56          if (process == null)
```

```
57              process = new ProcessOverviewAndScheduling();
58          return  process;
59      }
60
61      public static Submodel_s getSubmodel_s(){
62          if (submodel_s == null)
63              submodel_s = new Submodel_s();
64          return clearSubmodel_s();
65      }
66
67
68      /**
69       * Remove void submodels (without description) and duplicate submodels.
70       * @param submodel_s2
71       * @return submodel_s
72       */
73      private static Submodel_s clearSubmodel_s() {
74          //***********removing void submodel_s***********//
75          ArrayList<Submodel> voidSubmodel_s = new ArrayList<Submodel>();
76          for (Submodel s : submodel_s.getSubmodel_s()){
77              if (s.getDescription().equals(""))
78                  voidSubmodel_s.add(s);
79          }
80          for (Submodel s : voidSubmodel_s)
81              submodel_s.getSubmodel_s().remove(s);
82
83          //***********removing duplicate submodel_s***********//
84          //to remove duplicate entry I create an hashmap with
85          //key = description+name of submodel
86          //and entry = submodel. Submodels with equals
87          //description+name will be rewrite (delete).
88          HashMap<String,Submodel> hashSubmodel_s = new HashMap<String,Submodel>();
89          for (Submodel s : submodel_s.getSubmodel_s()){
90              hashSubmodel_s.put(s.getDescription() + s.getName(), s);
91          }
92          ArrayList<Submodel> newSubmodel_s = new ArrayList<Submodel>();
93          Collection<Submodel> collectionSubmodel_s = hashSubmodel_s.values();
94          for (Submodel s : collectionSubmodel_s)
95              newSubmodel_s.add(s);
96          submodel_s.setSubmodel_s(newSubmodel_s);
```

```
97          return submodel_s;
98      }
99
100     public static DesignConcepts getDesignConcepts(){
101         if (designConcepts == null)
102             designConcepts = new DesignConcepts();
103         return designConcepts;
104     }
105
106     public static Initialization getInitialization(){
107         if (initialization == null)
108             initialization = new Initialization();
109         return initialization;
110     }
111
112     public static void setPurpose(Purpose purpose) {
113         ODD.purpose = purpose;
114     }
115
116     public static void setEntitie_s(Entitie_s entitie_s) {
117         ODD.entitie_s = entitie_s;
118     }
119
120     public static void setProcess(ProcessOverviewAndScheduling process) {
121         ODD.process = process;
122     }
123
124     public static void setSubmodel_s(Submodel_s submodel_s) {
125         ODD.submodel_s = submodel_s;
126     }
127
128     public static void setDesignConcepts(DesignConcepts designConcepts) {
129         ODD.designConcepts = designConcepts;
130     }
131
132     public static void setInitialization(Initialization initialization) {
133         ODD.initialization = initialization;
134     }
135
136     public static void setInputData(String inputData) {
```

```
137             ODD.inputData = inputData;
138         }
139 
140         /**
141          * Add entity e in entite_s collection. First there is
142          * a control to check if Entity is already in collection.
143          * This has sense because also at first run of program
144          * some entities could be get from ODD serialized files.
145          * @param e
146          */
147         public static void addEntity(Entity e){
148             boolean alreadyIn = false;
149             ArrayList<Entity> entitie_s = getEntitie_s().getEntitie_s();
150             for (Entity entity : entitie_s){
151                 if (entity.getName().equals(e.getName())){
152                     entity.setDescription(e.getDescription());
153                     alreadyIn = true;
154                 }
155             }
156             if (!alreadyIn)
157                 getEntitie_s().add(e);
158         }
159 
160         public static Entity getEntity(String name){
161             return getEntitie_s().getEntity(name);
162         }
163 
164         public static void addVariableToEntity(String entityName, Variable v){
165             if (getEntity(entityName) == null)
166                 addEntity(new Entity(entityName, ""));
167             getEntity(entityName).addVariable(v);
168         }
169 
170         public static void addProcessOverviewElement
171                 (String who, String autoDoWhat, String userDoWhat){
172             getProcessOverviewAndScheduling()
173                     .addProcessOverviewElement(who, autoDoWhat, userDoWhat);
174         }
175 
176         public static void addSubmodel(Submodel s){
```

```
177             getSubmodel_s().addSubmodel(s);
178         }
179 
180         /**
181          * This method serialize all objects that it contains
182          * calling "serializeObj" method.
183          */
184         public static void serialize(){
185             try {
186                 serializeObj(Purpose.serializedName, getPurpose());
187                 serializeObj(Entitie_s.serializedName, getEntitie_s());
188                 serializeObj
189                     (ProcessOverviewAndScheduling.serializedName,
190                      getProcessOverviewAndScheduling());
191                 //serializeObj
192                     (Submodel_s.serializedName, getSubmodel_s());
193                 serializeObj(DesignConcepts.serializedName, getDesignConcepts());
194                 serializeObj(Initialization.serializedName, getInitialization());
195                 serializeObj("inputData.ser", inputData);
196                 log.info("ODD object Serialized");
197             } catch (IOException e) {
198                 log.severe("Error serializing ODD objects: " + e.getMessage());
199                 e.printStackTrace();
200             }
201         }
202 
203         /**
204          * This method deserialize all objects that it contains
205          * calling "deserializeObj" method.
206          */
207         public static void deserialize(){
208             try {
209                 ODD.setPurpose((Purpose) deserializeObj(Purpose.serializedName));
210                 ODD.setEntitie_s((Entitie_s)
211                     deserializeObj(Entitie_s.serializedName));
212                 ODD.setProcess((ProcessOverviewAndScheduling)
213                      deserializeObj(ProcessOverviewAndScheduling.serializedName));
214                 //ODD.setSubmodel_s((Submodel_s)
215                 //deserializeObj(Submodel_s.serializedName));
216                 ODD.setDesignConcepts((DesignConcepts)
```

```
217                 deserializeObj(DesignConcepts.serializedName));
218             ODD.setInitialization((Initialization)
219                 deserializeObj(Initialization.serializedName));
220             ODD.setInputData((String) deserializeObj("inputData.ser"));
221             log.info("ODD object deserialized");
222         } catch (IOException e) {
223             log.severe("IOException deserializing ODD objects: " + e.getMessage())
224             e.printStackTrace();
225         }
226         catch (ClassNotFoundException e) {
227             log.severe("ClassNotFound: " + e.getMessage());
228         }
229     }
230 
231     /**
232      * This method serialize object give in input with name
233      * objName in path "ConfigFile.getODDPath".
234      * @param objName   name to give to file
235      * @param toSerialize   object to serialize
236      * @throws FileNotFoundException
237      * @throws IOException
238      */
239     private static void serializeObj(String objName, Object toSerialize)
240             throws FileNotFoundException, IOException {
241         String serializedObjPath = ConfigFile.gettODDPath() + objName;
242         Files.deleteIfExists(Paths.get(serializedObjPath));
243         FileOutputStream fileOut = new FileOutputStream(serializedObjPath);
244         ObjectOutputStream out = new ObjectOutputStream(fileOut);
245         out.writeObject(toSerialize);
246         out.close();
247         fileOut.close();
248         log.info("Serialized: " + objName);
249     }
250 
251     /**
252      * This method return object read from "ConfigFile.getODDPath() + objname".
253      * @param objName
254      * @return
255      * @throws FileNotFoundException
256      * @throws IOException
```

```
257         * @throws ClassNotFoundException
258         */
259        private static Object deserializeObj(String objName)
260                    throws FileNotFoundException,IOException, ClassNotFoundException
261            FileInputStream fis = new FileInputStream(ConfigFile.gettODDPath() + objNa
262            ObjectInputStream ois = new ObjectInputStream(fis);
263            Object toReturn = ois.readObject();
264            ois.close();
265            log .info("Deserialized: " + objName);
266            return toReturn;
267        }
268
269        /**
270         * Return inputData if it isn't null;
271         * else return void String.
272         * @return
273         */
274        public static String getInputData(){
275            if (differentsColors){
276                if (inputData == null)  return "";
277                return GlobalUtility
278                .surroundWithSpan(GlobalUtility.userOutputColor, inputData);
279            }
280            else{
281                if (inputData == null)  return "";
282                return inputData;
283            }
284        }
285    }
```

A.1.2 La classe FieldInitializerVisitor

Di seguito la classe *FieldInitializerVisitor*. Tale classe *visita* le inizializzazione dei *field* della simulazione, creando una lista dei parametri di input di tali variabili.

```
1   package it.isislab.masonhelperdocumentation.visitor;
2
3   import java.util.List;
4
```

```
5  import it.isislab.masonhelperdocumentation.analizer.Parameter;
6
7  import org.eclipse.jdt.core.dom.ASTVisitor;
8  import org.eclipse.jdt.core.dom.Assignment;
9  import org.eclipse.jdt.core.dom.ClassInstanceCreation;
10 import org.eclipse.jdt.core.dom.CompilationUnit;
11 import org.eclipse.jdt.core.dom.Expression;
12
13 /**
14  * This visitor will get information about variables constructor.
15  * Implici parameter is the compilation unit of class that
16  * contains the variable. Explicit parameter is the parameter
17  * that we search in compilation Unit.
18  * ES. simStateCU.accept(new FieldInitializerVisitor(myParameter))
19  * will search all parameters in simStateCU and when myParameter
20  * is found will extract a list of constructor arguments.
21  * @author Romano Simone 0512101343
22  *
23  */
24 public class FieldInitializerVisitor extends ASTVisitor {
25     private String information_s;
26     private Parameter parameter;
27     private List<Expression> constructorParameter_s;
28
29     public FieldInitializerVisitor(Parameter parameter){
30         this.parameter = parameter;
31     }
32
33     /**
34      * Here we will extract parameters of field
35      * constructors.
36      * (Ex: new Grid(width,length...)->extract width,length,...).
37      */
38     public boolean visit(Assignment node){
39         if (node.getRightHandSide() instanceof ClassInstanceCreation){
40             ClassInstanceCreation constructor =
41                 (ClassInstanceCreation) node.getRightHandSide();
42             if (node.getLeftHandSide().toString().
43                     equals(parameter.getVariableName()))
44                 constructorParameter_s = constructor.arguments();
```

```
45          }
46          return super.visit(node);
47      }
48
49      public String getInformation_s() {
50          return information_s;
51      }
52
53      public Parameter getParameter() {
54          return parameter;
55      }
56
57      public List<Expression> getConstructorParameter_s() {
58          return constructorParameter_s;
59      }
60
61  }
```

A.1.3 La classe JavaDocVisitor

Di seguito la classe *JavaDocVisitor*. Tale classe *visita* i commenti in stile *JavaDoc* e, se sono stati generati da MAD(contengono la firma di MAD), allora vengono eliminati. Tale classe è stata quindi utilizzata per pulire il codice in seguito all'utilizzo del plugin che, come illustrato nel nostro lavoro, genera commenti in stile Doxygen.

```
1   package it.isislab.masonassisteddocumentation.visitor;
2
3   import it.isislab.masonassisteddocumentation.mason.analizer.GlobalUtility;
4
5   import org.eclipse.jdt.core.dom.ASTVisitor;
6   import org.eclipse.jdt.core.dom.CompilationUnit;
7   import org.eclipse.jdt.core.dom.Javadoc;
8
9   /**
10   * This class visits a JavaDoc node of a compilation unit
11   * and delete it if contains default MAD message.
12   * This to clear code from comment add by MAD.
13   * @author Simone
14   *
```

```
15   */
16  public class JavaDocVisitor extends ASTVisitor {
17      private CompilationUnit cu;
18
19      public JavaDocVisitor(CompilationUnit cu) {
20          this.cu = cu;
21      }
22
23      public boolean visit(Javadoc node) {
24          if (node.toString().contains(GlobalUtility.COMMENT_SIGNATURE)){
25              node.delete();
26          }
27          return super.visit(node);
28      }
29  }
```

A.1.4 CodeVisitor

Tale classe astratta contiene i metodi per visitare una *CompilationUnit*. Durante tale visita genera informazioni appropriate per ogni elemento che incontra. Il metodo

```
1       public abstract boolean visit(MethodInvocation node);
```

è implementato dalle classi *StartStepMethodVisitor* e *RecoursiveCodeVisitor* in maniera diversa(vedi A.1.5 A.1.6) che estendono *CodeVisitor*.

```
1   package it.isislab.masonassisteddocumentation.visitor;
2
3
4   import org.eclipse.jdt.core.dom.ASTVisitor;
5   import org.eclipse.jdt.core.dom.Assignment;
6   import org.eclipse.jdt.core.dom.CompilationUnit;
7   import org.eclipse.jdt.core.dom.DoStatement;
8   import org.eclipse.jdt.core.dom.ForStatement;
9   import org.eclipse.jdt.core.dom.IfStatement;
10  import org.eclipse.jdt.core.dom.MethodInvocation;
11  import org.eclipse.jdt.core.dom.SwitchStatement;
12  import org.eclipse.jdt.core.dom.TypeDeclaration;
13
14  public abstract class CodeVisitor extends ASTVisitor {
```

```
15      private CompilationUnit cu;
16      protected String information_s;
17
18      public CodeVisitor(CompilationUnit cu){
19          this.cu = cu;
20          information_s = "";
21      }
22
23      @Override
24      public boolean visit(IfStatement node) {
25          information_s += Messages.Visitor_If1 + node.getExpression() + " ";
26          return super.visit(node);
27      }
28
29      public void endVisit(IfStatement node){
30          information_s += Messages.Visitor_EndIf;
31      }
32
33      public boolean visit(ForStatement f){
34          information_s += Messages.Visitor_While1 + f.getExpression() + " ";
35      }
36
37      public void endVisit(ForStatement node){
38          information_s += Messages.Visitor_EndFor; //$NON-NLS-2$;
39      }
40
41      public boolean visit (DoStatement d){
42          information_s += Messages.Visitor_Do1 + d.getExpression() + " ";
43          return super.visit(d);
44      }
45
46      public void endVisit(DoStatement node){
47          information_s += Messages.Visitor_DoWhileEnd;
48      }
49
50      public boolean visit(SwitchStatement s){
51          information_s += Messages.Visitor_SwitchInfo + s.getExpression() + " ";
52      }
53
54      public void endVisit(SwitchStatement node){
```

```
55          information_s += Messages.Visitor_EndSwitch;
56      }
57
58      public boolean visit(TypeDeclaration t){
59          information_s += Messages.Visitor_TypeDeclaration + t.getName();
60          return super.visit(t);
61      }
62
63      public void endVisit(TypeDeclaration node){
64          information_s += Messages.Visitor_EndTypeDeclaration;
65      }
66
67      public boolean visit(Assignment node){
68          information_s += Messages.Visitor_Assignment + node.getLeftHandSide()
69          + Messages.Visitor_Assignemnt1 + node.getRightHandSide() + ".";
70          return super.visit(node);
71      }
72
73      public void endVisit(Assignment node){
74          information_s += Messages.Visitor_EndAssignent;
75      }
76
77
78      public void endVisit(MethodInvocation node){
79          information_s += Messages.Visitor_EndInvocation + node.getName()
80              + Messages.Visitor_EndInvocation1;
81      }
82
83      public String getInformation_s() {
84          return information_s + Messages.Visitor_EndMethod;
85      }
86
87      /**
88       * This method add '\n' each 8 white space.
89       */
90      private void addSomeNewLine() {
91          int eachNumSpace = 8;
92          int spaceCounter = 0;
93          StringBuilder result = new StringBuilder(information_s);
94          for (int i=0; i<information_s.length(); i++){
```

```
95              if (information_s.charAt(i) == ' '){
96                  spaceCounter++;
97                  if (spaceCounter >= eachNumSpace){
98                      result.setCharAt(i, '\n');
99                      spaceCounter = 0;
100                 }
101             }
102         }
103         information_s = result.toString();
104     }
105
106     public abstract boolean visit(MethodInvocation node);
107 }
```

A.1.5 StartStepMethodVisitor

Tale classe estende la classe *CodeVisitor* implementando il metodo

```
1       public abstract boolean visit(MethodInvocation node);
```

Durante tale metodo crea una lista per i metodi definiti dall'utente che analizzerà durante la fase dedicata ai *Submodels*.

```
1   package it.isislab.masonassisteddocumentation.visitor;
2
3   import it.isislab.masonassisteddocumentation.ODD.ODD;
4   import it.isislab.masonassisteddocumentation.ODD.Submodel;
5   import java.util.List;
6   import java.util.logging.Logger;
7
8   import org.eclipse.jdt.core.dom.CompilationUnit;
9   import org.eclipse.jdt.core.dom.Expression;
10  import org.eclipse.jdt.core.dom.MethodInvocation;
11
12  /**
13   * This visitor will visit Start method
14   * and will extract information about schedule
15   * and objects locations.
16   * @author Romano Simone 0512101343
17   *
18   */
```

```
19  public class StartStepMethodVisitor extends CodeVisitor {
20      private CompilationUnit cu;
21      private static Logger log = Logger.getLogger("global");
22
23      public StartStepMethodVisitor(CompilationUnit cu) {
24          super(cu);
25          this.cu = cu;
26          information_s = ""; //$NON-NLS-1$
27      }
28
29      public boolean visit(MethodInvocation node) {
30          information_s += Messages.Visitor_InvokeMethod + node.getName() +".\n";
31          if (node.getName().toString().equals("scheduleRepeating"))
32              generateScheduleInformations(node);
33          else if (node.getName().toString().equals("setObjectLocation"))
34              generateSetObjectLocationInformations(node);
35          else if (node.getName().toString().equals("getObjectLocation"))
36              generateGetObjectLocationInformations(node);
37          else if (node.getName().toString().equals("multiply"))
38              generateMultiplyInformations(node);
39          else if (node.getName().toString().equals("nextDouble"))
40              generateNextDoubleInformations();
41          else if (node.getName().toString().equals("nextInt"))
42              generateNextIntInformations();
43          else if (node.getName().toString().equals("addEdge"))
44              generateAddEdgeInformations(node);
45          else if (node.getName().toString().equals("addNode"))
46              generateAddNodeInformations(node);
47          else if (node.getName().toString().equals("getAllObjects"))
48              generateGetAllObjectsInformations(node);
49          else{   //add submodel for last ODD phase ("submodels")
50              information_s = information_s + Messages.Visitor_SeeSubmodel;
51              Submodel toAdd = new Submodel(node.getName().toString(), cu);
52              ODD.addSubmodel(toAdd);
53              log.info(Messages.Visitor_SubmodelFound + node.getName() +"'.");
54          }
55          return true;
56      }
57
58      private void generateAddEdgeInformations(MethodInvocation node) {
```

```
59          List<Expression> argument_s = node.arguments();
60          if (argument_s.size() == 3){
61              information_s += Messages.Visitor_AddEdgeInformation
62                  +node.getExpression() +Messages.Visitor_AddEdgeInfo1
63                  + argument_s.get(0) +Messages.Visitor_AddEdgeInfo2
64                  + argument_s.get(1) +Messages.Visitor_AddEdgeInfo3;
65          }
66          if (argument_s.size() == 1){
67              information_s += Messages.Visitor_AddEdgeInfo4
68                  + argument_s.get(0) + Messages.Visitor_AddEdgeInfo5
69                  + node.getExpression() + "'.\n";
70          }
71      }
72
73      private void generateNextIntInformations() {
74          information_s += Messages.Visitor_IntRandom;
75      }
76
77      private void generateGetObjectLocationInformations(MethodInvocation node) {
78          information_s += Messages.Visitor_GetObj1 + node.getExpression()
79              +Messages.Visitor_GetObj2;
80      }
81
82      private void generateGetAllObjectsInformations(MethodInvocation node) {
83          information_s += Messages.Visitor_AllObj1 + node.getExpression() +"'.\n";
84      }
85
86      private void generateAddNodeInformations(MethodInvocation node) {
87          information_s += Messages.Visitor_AddNode1 + node.getExpression() + ".\n";
88      }
89
90      private void generateNextDoubleInformations() {
91          information_s += Messages.Visitor_RadomDouble1;
92      }
93
94      private void generateMultiplyInformations(MethodInvocation node) {
95          List<Expression> argument_s = node.arguments();
96          if (argument_s.size() == 1)
97              information_s = information_s + Messages.Visitor_MutlipliesInfo1
98                  + node.getExpression() +Messages.Visitor_MutliplesInfo2
```

```
99                  + argument_s.get(0) +"'.\n";
100         }
101
102         /**
103          * Get information about 'setObjectLocation' method.
104          * @param node
105          */
106         private void generateSetObjectLocationInformations(MethodInvocation node) {
107             List<Expression> argument_s = node.arguments();
108             if (argument_s.size() == 3){    //schedule.repeating(3 parameters)
109                 information_s = information_s + Messages.Visitor_SetObjLoc1
110                     + argument_s.get(0)
111                     + Messages.Visitor_SetObjLoc2 + node.getExpression() + "': \n"
112                     + Messages.Visitor_SetObjLoc3 + argument_s.get(1) + ";\n"
113                     + Messages.Visitor_SetObjLoc4 + argument_s.get(2) + ".\n\n";
114             }
115             else if (argument_s.size() == 2){
116                 information_s = information_s + Messages.Visitor_SetObjLoc1
117                     + argument_s.get(0) + Messages.Visitor_SetObjLoc2
118                     + node.getExpression() + "':\n" //$NON-NLS-1$
119                         + "In position: " + argument_s.get(1) + ";\n";
120             }
121         }
122
123         /**
124          * Get information about schedule.
125          * @param node
126          */
127         private void generateScheduleInformations(MethodInvocation node) {
128             List<Expression> argument_s = node.arguments();
129             if (argument_s.size() == 4){    //schedule.repeating(4 parameters here)
130                 information_s = information_s + Messages.Visitor_Schedule1;
131                 information_s = information_s + Messages.Visitor_Schedule2
132                         + argument_s.get(0) +";\n"
133                         + Messages.Visitor_Schedule3 + argument_s.get(1) + ";\n"
134                         + Messages.Visitor_Schedule4 + argument_s.get(2) + ";\n"
135                         + Messages.Visitor_Schedule5 + argument_s.get(3) + ".\n\n";
136             }
137
138         }
```

```
139 }
```

A.1.6 RecoursiveCodeVisitor

Tale classe estende la classe *CodeVisitor* implementando il metodo

```
1       public abstract boolean visit(MethodInvocation node);
```

Durante tale metodo visita ricorsivamente i metodi invocati.

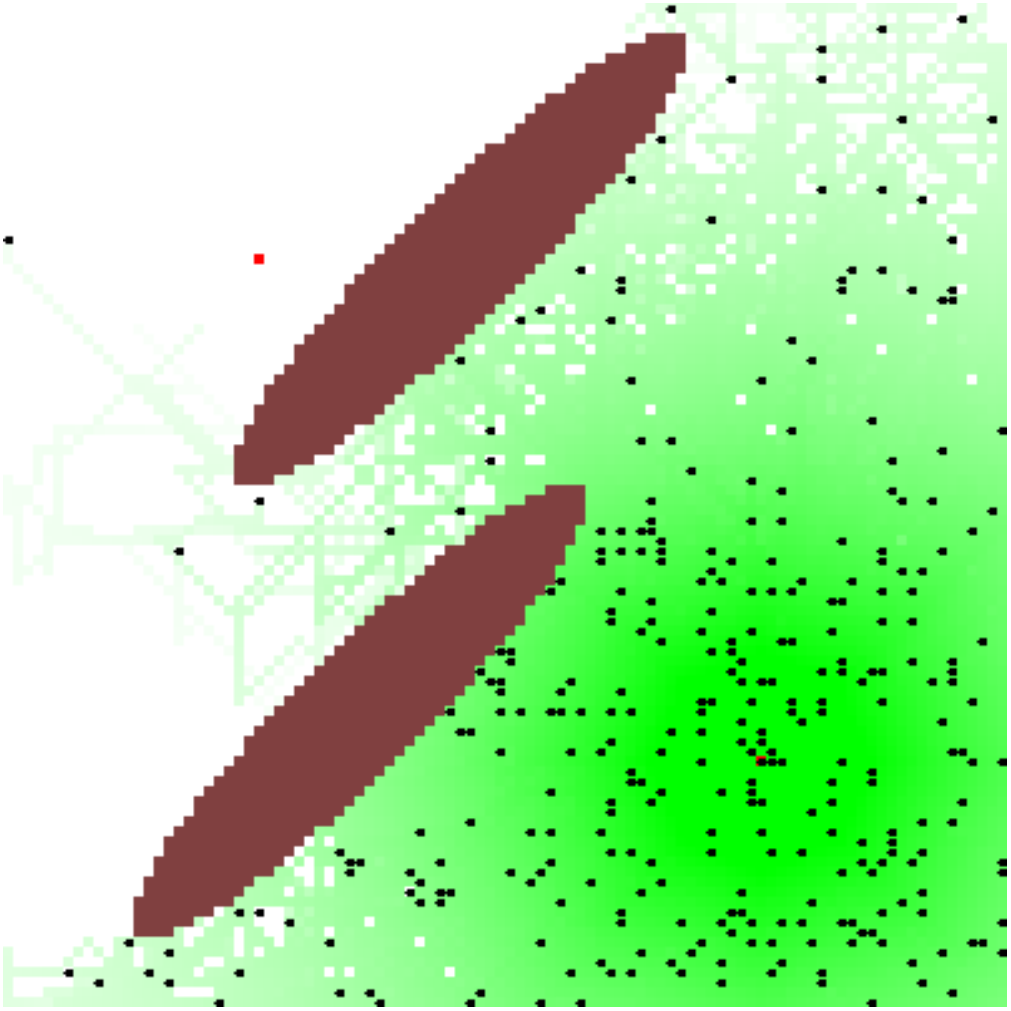
```
1  package it.isislab.masonassisteddocumentation.visitor;
2
3  import it.isislab.masonassisteddocumentation.mason.analizer.GlobalUtility;
4  import it.isislab.masonassisteddocumentation.mason.analizer.Method;
5
6  import java.util.ArrayList;
7
8  import org.eclipse.jdt.core.dom.CompilationUnit;
9  import org.eclipse.jdt.core.dom.MethodInvocation;
10
11 /**
12  * This class represent a visitor for method.
13  * Purpose of this class is extract a description
14  * of method in natural language.
15  * @author Romano Simone 0512101343
16  *
17  */
18 public class RecursiveCodeVisitor extends CodeVisitor{
19     private CompilationUnit cu;
20
21     public RecursiveCodeVisitor(CompilationUnit cu){
22         super(cu);
23         this.cu = cu;
24         information_s = "";
25     }
26
27     public boolean visit(MethodInvocation node) {
28         boolean visitedSubmetod = false;
29         information_s += "Invoke method " + node.getName();
30         ArrayList<Method> method_s = GlobalUtility.getAllMethods(cu);
31         for (Method m : method_s){
```

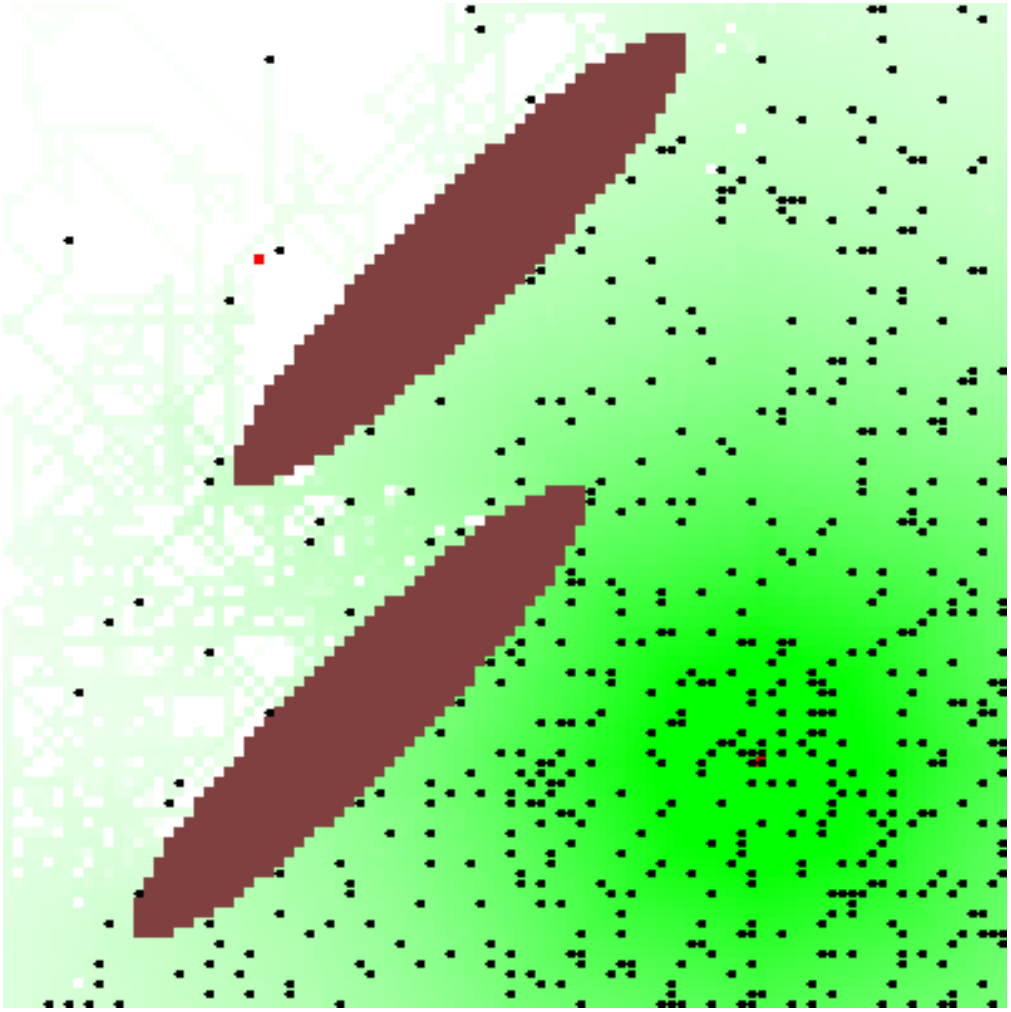
```
32              if (m.getName().toString().equals(node.getName().toString())){
33                  visitedSubmetod = true;
34                  information_s += " that: ";
35                  m.getMethod().getBody().accept(new RecursiveCodeVisitor(cu));
36              }
37          }
38          if (!visitedSubmetod) information_s += ".\n";
39          return true;
40      }
41  }
```

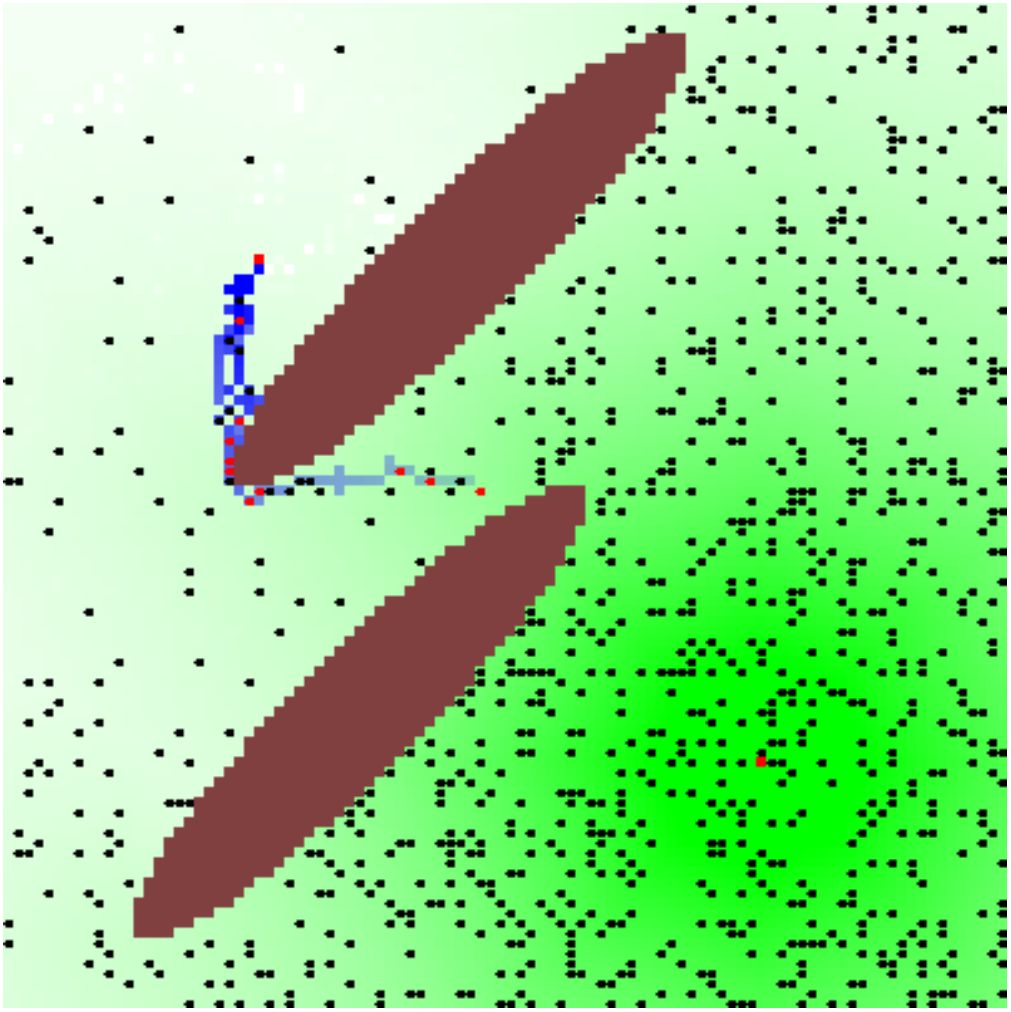
A.2 Esempio di utilizzo di MAD

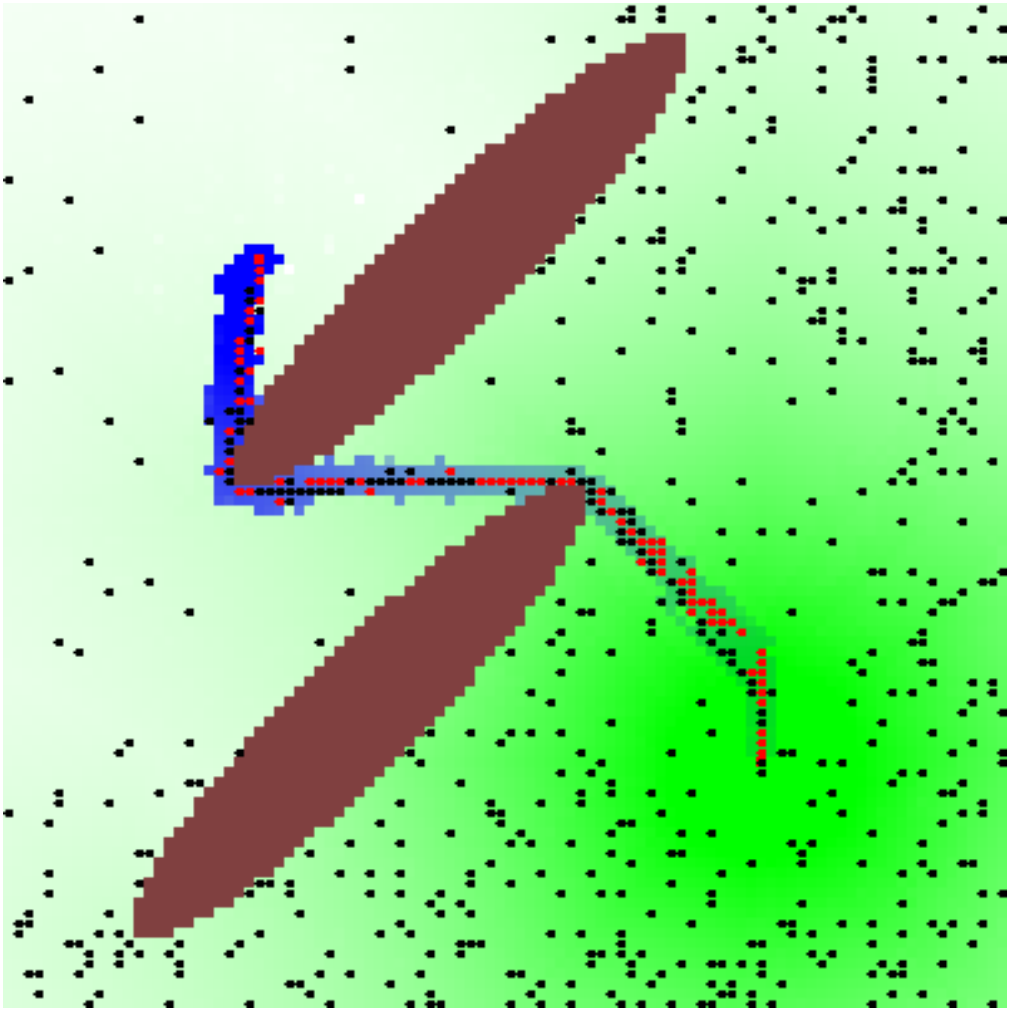
Di seguito sarà mosrato un esempio di documentazione del programma *AntsForage* fornito con la libreria MASON. La documentazione mostrata è in formato *.pdf* ed è stata prodotta utilizzando MAD con *output type = pdf-rtf*. Da notare che utilizzando MAD con *output type = Doxygen* vengono prodotte informazioni aggiuntive generate dal tool Doxygen.

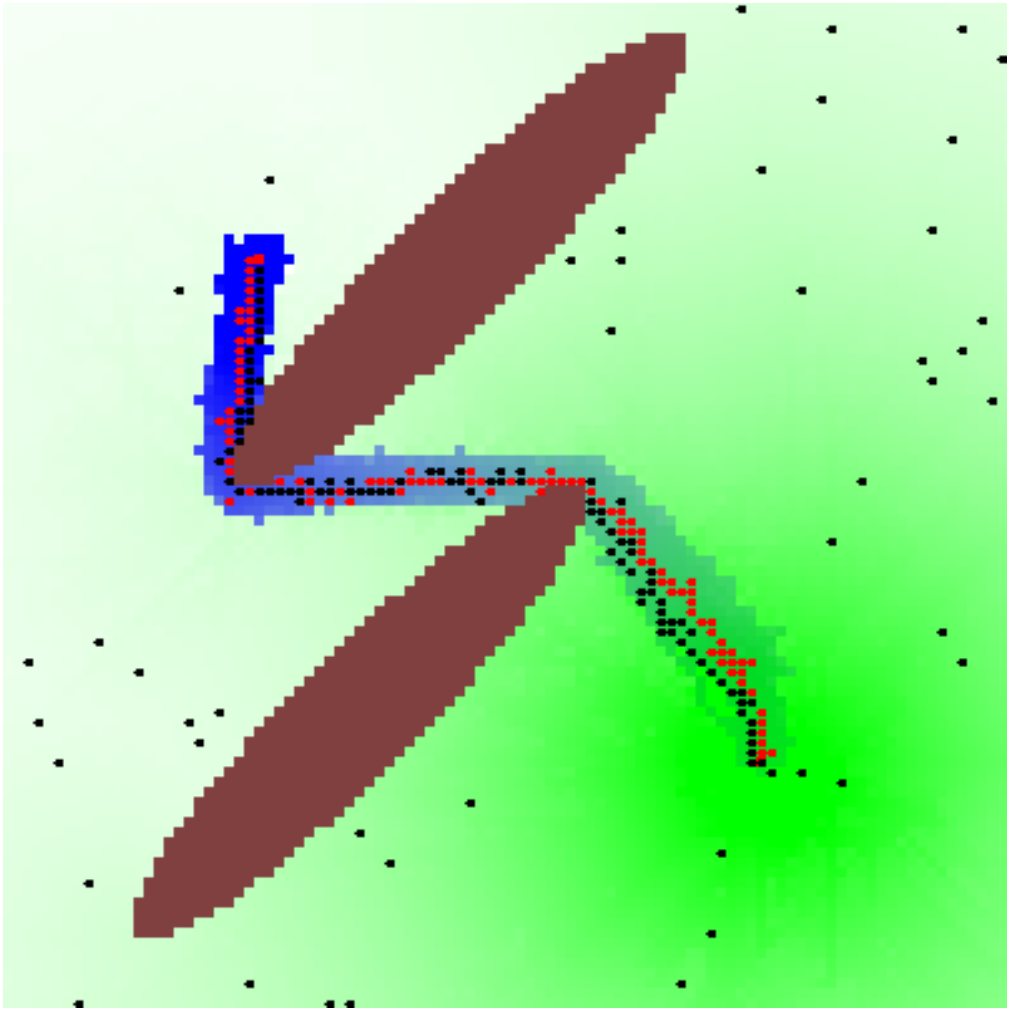
AntsForage documentation

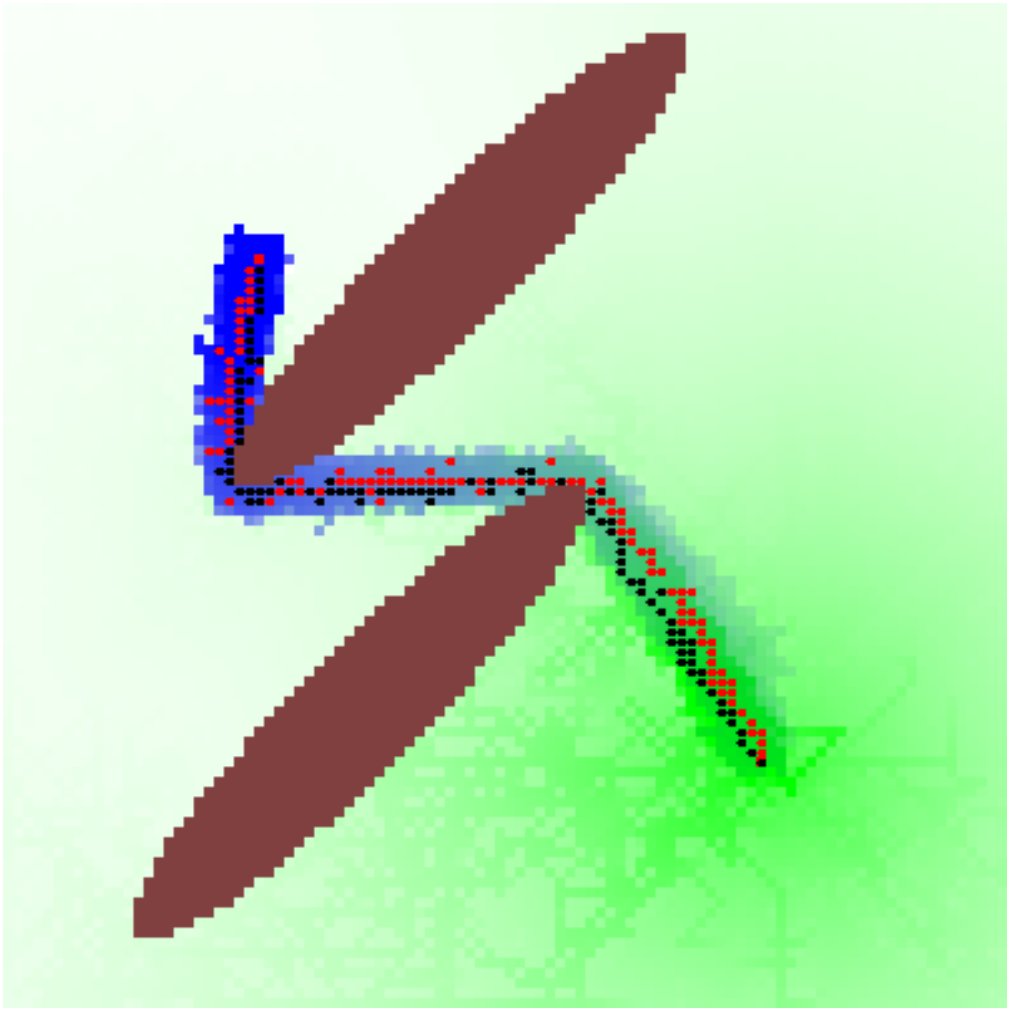












ODD Description

Document has been generated by MAD(Mason assisted documentation), built by Simone Romano in ISISLab.
The model description follows the ODD(Overview, Design concepts, Details) protocol (Grimm et al., 2006, this work).This documentation contains information on the simulation model. These information are partially self-generated and partially entered.The information is grouped according to the protocol ODD. Each section of the protocol answers some questions. In particular:

- *Purpose*

What is the purpose of the model?

- *Entities, state variables and scales*

What kinds of entities are in the model?bBy what state variables, or attributes, are these entities characterized?

- *Process overview and scheduling*

Who (i.e., what entity) does what, and in what order?

- *Design concepts*

Which general concepts, theories, hypotheses, or modeling approaches are underlying the model's design?What key results or outputs of the model are modeled as emerging from the adaptive traits, or behaviors, of individuals?What

adaptive traits do the individuals have? What rules do they have for making decisions or changing behavior in response to changes in themselves or their environment If adaptive traits explicitly act to increase some measure of theindividual's success at meeting some objective, what exactly is that objective and how is it measured? Many individuals or agents (but also organizations and institutions change their adaptive traits over time as a consequence of their experience? If so, how? Prediction is fundamental to successful decision-making; if an agent's adaptive traits or learning procedures are based on estimating future consequences of decisions,howdo agents predict the future conditions (either environmental or internal) they will experience? What internal and environmental state variables are individuals assumed to sense and consider in their decisions? What kinds of interactions among agents are assumed? Are there direct interactions in which individuals encounter and affect others, or are interactions indirect, e.g., via competition for amediating resource? What processes are modeled by assuming they are random or partly random? Do the individuals form or belong to aggregations that affect, and are affected by, the individuals? What data are collected from the ABM for testing, understanding, and analyzing it, and how and when are they collected?

- *Initialization*

What is the initial state of the model world, i.e., at time t = 0 of a simulation run?

- *Input data*

Does the model use input from external sources such as data files or other models to represent processes that change over time?

- *Submodels*

Information about submodels identified in seciont 'Process overview and scheduling'.

Purpose

Purpose of the model.

Entities, variables, and scales

Entity: AntsForage.java

- *randomActionProbability*

Initial value of randomActionProbability variable is: 0.1

- *buggrid*

The width value of buggrid is: GRID_WIDTH;
the height value of buggrid is: GRID_HEIGHT.
Value of GRID_WIDTH is: 100.
Value of GRID_HEIGHT is: 100.

- *toHomeGrid*

The width value of toHomeGrid is: GRID_WIDTH;
the height value of toHomeGrid is: GRID_HEIGHT.
Value of toHomeGrid is: 0.
Value of GRID_WIDTH is: 100.
Value of GRID_HEIGHT is: 100.

- *FOOD_XMAX*

Initial value of FOOD_XMAX variable is: 25

- *momentumProbability*

Initial value of momentumProbability variable is: 0.8

- *diagonalCutDown*

Initial value of diagonalCutDown variable is: computeDiagonalCutDown()

- *FOOD_XMIN*

Initial value of FOOD_XMIN variable is: 25

- *ONE_OBSTACLE*

Initial value of ONE_OBSTACLE variable is: 1

- *ALGORITHM_VALUE_ITERATION*

Initial value of ALGORITHM_VALUE_ITERATION variable is: 1

- *evaporationConstant*

Initial value of evaporationConstant variable is: 0.999

- *FOOD_YMAX*

Initial value of FOOD_YMAX variable is: 25

- *ONE_LONG_OBSTACLE*

Initial value of ONE_LONG_OBSTACLE variable is: 3

- *TWO_OBSTACLES*

Initial value of TWO_OBSTACLES variable is: 2

- *LIKELY_MAX_PHEROMONE*

Initial value of LIKELY_MAX_PHEROMONE variable is: 3

- *NO_OBSTACLES*

Initial value of NO_OBSTACLES variable is: 0

- *serialVersionUID*

Initial value of serialVersionUID variable is: 1

test

- *OBSTACLES*

Initial value of OBSTACLES variable is: TWO_OBSTACLES

- *toFoodGrid*

The width value of toFoodGrid is: GRID_WIDTH;
the height value of toFoodGrid is: GRID_HEIGHT.
Value of toFoodGrid is: 0.
Value of GRID_WIDTH is: 100.
Value of GRID_HEIGHT is: 100.

- *FOOD_YMIN*

Initial value of FOOD_YMIN variable is: 25

- *sites*

The width value of sites is: GRID_WIDTH;
the height value of sites is: GRID_HEIGHT.
Value of sites is: 0.
Value of GRID_WIDTH is: 100.
Value of GRID_HEIGHT is: 100.
sites

- *reward*

Initial value of reward variable is: 1.0

- *FOOD*

Initial value of FOOD variable is: 2

- *obstacles*

The width value of obstacles is: GRID_WIDTH;
the height value of obstacles is: GRID_HEIGHT.
Value of obstacles is: 0.
Value of GRID_WIDTH is: 100.
Value of GRID_HEIGHT is: 100.

- *ALGORITHM*

Initial value of ALGORITHM variable is: ALGORITHM_VALUE_ITERATION

- *updateCutDown*

Initial value of updateCutDown variable is: 0.9

- *HOME_XMIN*

Initial value of HOME_XMIN variable is: 75

- *HOME*

Initial value of HOME variable is: 1

- *IMPOSSIBLY_BAD_PHEROMONE*

Initial value of IMPOSSIBLY_BAD_PHEROMONE variable is: -1

- *ALGORITHM_TEMPORAL_DIFERENCE*

Initial value of ALGORITHM_TEMPORAL_DIFERENCE variable is: 2

- *HOME_XMAX*

Initial value of HOME_XMAX variable is: 75

- *HOME_YMIN*

Initial value of HOME_YMIN variable is: 75

- *HOME_YMAX*

Initial value of HOME_YMAX variable is: 75

- *numAnts*

Initial value of numAnts variable is: 1000

Entity: Ant.java

Ant description here.

Ant description here.

- *foodColor*

Initial value of foodColor variable is: Color.red

- *reward*

Initial value of reward variable is: 0

test

- *last*

Variable 'Int2D last' contains information about agent's position.

last2D user input info.

- *noFoodColor*

Initial value of noFoodColor variable is: Color.black

- *serialVersionUID*

Initial value of serialVersionUID variable is: 1

test

- *hasFoodItem*

Initial value of hasFoodItem variable is: false

testtest

Process, overview and schedule

- *Start method of AntsForage.java*

There is an assignment for sites to value new IntGrid2D(GRID_WIDTH,GRID_HEIGHT,0).
After this assignment, There is an assignment for toFoodGrid to value new DoubleGrid2D(GRID_WIDTH,GRID_HEIGHT,0).
After this assignment, There is an assignment for toHomeGrid to value new DoubleGrid2D(GRID_WIDTH,GRID_HEIGHT,0).
After this assignment, There is an assignment for buggrid to value new SparseGrid2D(GRID_WIDTH,GRID_HEIGHT).
After this assignment, There is an assignment for obstacles to value new IntGrid2D(GRID_WIDTH,GRID_HEIGHT,0).
After this assignment, There is a switch on OBSTACLES There is a for loop that while x < GRID_WIDTH There is a for loop that while y < GRID_HEIGHT There is an assignment for obstacles.field[x][y] to value 0.
After this assignment, If ((x - 55) * 0.707 + (y - 35) * 0.707) * ((x - 55) * 0.707 + (y - 35) * 0.707) / 36 + ((x - 55) * 0.707 - (y - 35) * 0.707) * ((x - 55) * 0.707 - (y - 35) * 0.707) / 1024 <= 1 There is an assignment for obstacles.field[x][y] to value 1.
After this assignment, End if. End for. End for. There is a for loop that while x < GRID_WIDTH There is a for loop that while y < GRID_HEIGHT There is an assignment for obstacles.field[x][y] to value 0.
After this assignment, If ((x - 45) * 0.707 + (y - 25) * 0.707) * ((x - 45) * 0.707 + (y - 25) * 0.707) / 36 + ((x - 45) * 0.707 - (y - 25) * 0.707) * ((x - 45) * 0.707 - (y - 25) * 0.707) / 1024 <= 1 There is an assignment for obstacles.field[x][y] to value 1.
After this assignment, End if. If ((x - 35) * 0.707 + (y - 70) * 0.707) * ((x - 35) * 0.707 + (y - 70) * 0.707) / 36 + ((x - 35) * 0.707 - (y - 70) * 0.707) * ((x - 35) * 0.707 - (y - 70) * 0.707) / 1024 <= 1 There is an assignment for obstacles.field[x][y] to value 1.
After this assignment, End if. End for. End for. There is a for loop that while x < GRID_WIDTH There is a for loop that while y < GRID_HEIGHT There is an assignment for obstacles.field[x][y] to value 0.
After this assignment, If (x - 60) * (x - 60) / 1600 + (y - 50) * (y - 50) / 25 <= 1 There is an assignment for obstacles.field[x][y] to value 1.
After this assignment, End if. End for. End for.
After this switch-statement, There is a for loop that while x <= HOME_XMAX There is a for loop that while y <= HOME_YMAX There is an assignment for sites.field[x][y] to value HOME.
After this assignment, End for. End for. There is a for loop that while x <= FOOD_XMAX There is a for loop that while y <= FOOD_YMAX There is an assignment for sites.field[x][y] to value FOOD.
After this assignment, End for. End for. There is a for loop that while x < numAnts Invoke method setObjectLocation.
Move agent 'ant' on grid 'buggrid':
In x position: (HOME_XMAX + HOME_XMIN) / 2;
And y position: (HOME_YMAX + HOME_YMIN) / 2.

After invocation of setObjectLocation method, Invoke method scheduleRepeating.
Then an agent is added to schedule:
Time of scheduling: Schedule.EPOCH + x;
Order of scheduling: 0;
Agent to schedule: ant;
Interval of scheduling: 1.

After invocation of scheduleRepeating method, End for. Invoke method scheduleRepeating.
Then an agent is added to schedule:
Time of scheduling: Schedule.EPOCH;
Order of scheduling: 1;

```
Agent to schedule: new Steppable(){
  public void step(  SimState state){
    toFoodGrid.multiply(evaporationConstant);
    toHomeGrid.multiply(evaporationConstant);
  }
}
;
```

Interval of scheduling: 1.

Invoke method multiply.
Multiplies 'toFoodGrid' against the scalar 'evaporationConstant'.
After invocation of multiply method, Invoke method multiply.
Multiplies 'toHomeGrid' against the scalar 'evaporationConstant'.
After invocation of multiply method, After invocation of scheduleRepeating method, method terminates.

- *Step method of Ant.java*

Invoke method depositPheromone.
(Submodels section for more info).
After invocation of depositPheromone method, Invoke method act.
(Submodels section for more info).
After invocation of act method, method terminates.

Design concepts

- *Basic principles*

Basic principles

- *Emergence*

Emergence

- *Adaption*

Adaption

- *Objectives*

Objectives

- *Learning*

Learning

- *Prediction*

No Prediction defined.

- *Sensing*

No Sensing defined.

- *Interaction*

No Interaction defined.

- *Stochasticity*

No Stochasticity defined.

- *Collectives*

No Collectives defined.

- *Observation*

No Observation defined.

Initialization

At time t=0:

Entity AntsForage.java
variable 'randomActionProbability' has initial value = 0.1;
variable 'buggrid' has initial value = new SparseGrid2D(GRID_WIDTH,GRID_HEIGHT);
variable 'toHomeGrid' has initial value = new DoubleGrid2D(GRID_WIDTH,GRID_HEIGHT,0);
variable 'FOOD_XMAX' has initial value = 25;
variable 'momentumProbability' has initial value = 0.8;
variable 'diagonalCutDown' has initial value = computeDiagonalCutDown();
variable 'FOOD_XMIN' has initial value = 25;
variable 'ONE_OBSTACLE' has initial value = 1;
variable 'ALGORITHM_VALUE_ITERATION' has initial value = 1;
variable 'evaporationConstant' has initial value = 0.999;
variable 'FOOD_YMAX' has initial value = 25;
variable 'ONE_LONG_OBSTACLE' has initial value = 3;
variable 'TWO_OBSTACLES' has initial value = 2;
variable 'LIKELY_MAX_PHEROMONE' has initial value = 3;
variable 'NO_OBSTACLES' has initial value = 0;
variable 'serialVersionUID' has initial value = 1;
variable 'OBSTACLES' has initial value = TWO_OBSTACLES;
variable 'toFoodGrid' has initial value = new DoubleGrid2D(GRID_WIDTH,GRID_HEIGHT,0);
variable 'FOOD_YMIN' has initial value = 25;
variable 'sites' has initial value = new IntGrid2D(GRID_WIDTH,GRID_HEIGHT,0);
variable 'reward' has initial value = 1.0;
variable 'FOOD' has initial value = 2;
variable 'obstacles' has initial value = new IntGrid2D(GRID_WIDTH,GRID_HEIGHT,0);
variable 'ALGORITHM' has initial value = ALGORITHM_VALUE_ITERATION;
variable 'updateCutDown' has initial value = 0.9;
variable 'HOME_XMIN' has initial value = 75;
variable 'HOME' has initial value = 1;
variable 'IMPOSSIBLY_BAD_PHEROMONE' has initial value = -1;
variable 'ALGORITHM_TEMPORAL_DIFERENCE' has initial value = 2;
variable 'HOME_XMAX' has initial value = 75;
variable 'HOME_YMIN' has initial value = 75;
variable 'HOME_YMAX' has initial value = 75;
variable 'numAnts' has initial value = 1000;

Entity Ant.java
variable 'foodColor' has initial value = Color.red;
variable 'reward' has initial value = 0;
variable 'last' has initial value = ;

variable 'noFoodColor' has initial value = Color.black;
variable 'serialVersionUID' has initial value = 1;
variable 'hasFoodItem' has initial value = false;

User initialization comment.

Input Data

testtesttesttest

Submodels

- *act*

Invoke method getObjectLocation.
After invocation of getObjectLocation method, If hasFoodItem There is a for loop that while dx < 2 There is a for loop that while dy < 2 If (dx == 0 && dy == 0) || _x < 0 || _y < 0 || _x >= AntsForage.GRID_WIDTH || _y >= AntsForage.GRID_HEIGHT || af.obstacles.field[_x][_y] == 1 End if. If m > max There is an assignment for count to value 2.
After this assignment, End if. If m > max || (m == max && state.random.nextBoolean(1.0 / count++)) Invoke method nextBoolean.
After invocation of nextBoolean method, There is an assignment for max to value m.
After this assignment, There is an assignment for max_x to value _x.
After this assignment, There is an assignment for max_y to value _y.
After this assignment, End if. End for. End for. If max == 0 && last != null If state.random.nextBoolean(af.momentumProbability) Invoke method nextBoolean.
After invocation of nextBoolean method, If xm >= 0 && xm < AntsForage.GRID_WIDTH && ym >= 0 && ym < AntsForage.GRID_HEIGHT && af.obstacles.field[xm][ym] == 0 There is an assignment for max_x to value xm.
After this assignment, There is an assignment for max_y to value ym.
After this assignment, End if. End if. If state.random.nextBoolean(af.randomActionProbability) Invoke method nextBoolean.
After invocation of nextBoolean method, Invoke method nextInt.
After invocation of nextInt method, Invoke method nextInt.
After invocation of nextInt method, If !(xd == 0 && yd == 0) && xm >= 0 && xm < AntsForage.GRID_WIDTH && ym >= 0 && ym < AntsForage.GRID_HEIGHT && af.obstacles.field[xm][ym] == 0 There is an assignment for max_x to value xm.
After this assignment, There is an assignment for max_y to value ym.
After this assignment, End if. End if. End if. Invoke method setObjectLocation.
After invocation of setObjectLocation method, If af.sites.field[max_x][max_y] == AntsForage.HOME There is an assignment for reward to value af.reward.
After this assignment, There is an assignment for hasFoodItem to value !hasFoodItem.
After this assignment, End if. There is a for loop that while dx < 2 There is a for loop that while dy < 2 If (dx == 0 && dy == 0) || _x < 0 || _y < 0 || _x >= AntsForage.GRID_WIDTH || _y >= AntsForage.GRID_HEIGHT || af.obstacles.field[_x][_y] == 1 End if. If m > max There is an assignment for count to value 2.
After this assignment, End if. If m > max || (m == max && state.random.nextBoolean(1.0 / count++)) Invoke method nextBoolean.
After invocation of nextBoolean method, There is an assignment for max to value m.
After this assignment, There is an assignment for max_x to value _x.
After this assignment, There is an assignment for max_y to value _y.
After this assignment, End if. End for. End for. If max == 0 && last != null If state.random.nextBoolean(af.momentumProbability) Invoke method nextBoolean.
After invocation of nextBoolean method, If xm >= 0 && xm < AntsForage.GRID_WIDTH && ym >= 0 && ym < AntsForage.GRID_HEIGHT && af.obstacles.field[xm][ym] == 0 There is an assignment for max_x to value xm.
After this assignment, There is an assignment for max_y to value ym.
After this assignment, End if. End if. If state.random.nextBoolean(af.randomActionProbability) Invoke method nextBoolean.
After invocation of nextBoolean method, Invoke method nextInt.

After invocation of nextInt method, Invoke method nextInt.
After invocation of nextInt method, If !(xd == 0 && yd == 0) && xm >= 0 && xm < AntsForage.GRID_WIDTH && ym >= 0 && ym < AntsForage.GRID_HEIGHT && af.obstacles.field[xm][ym] == 0 There is an assignment for max_x to value xm.
After this assignment, There is an assignment for max_y to value ym.
After this assignment, End if. End if. End if. Invoke method setObjectLocation.
After invocation of setObjectLocation method, If af.sites.field[max_x][max_y] == AntsForage.FOOD There is an assignment for reward to value af.reward.
After this assignment, There is an assignment for hasFoodItem to value !hasFoodItem.
After this assignment, End if. End if. There is an assignment for last to value location.
After this assignment, method terminates.

- *depositPheromone*

Invoke method getObjectLocation.
After invocation of getObjectLocation method, If AntsForage.ALGORITHM == AntsForage.ALGORITHM_VALUE_ITERATION If hasFoodItem There is a for loop that while dx < 2 There is a for loop that while dy < 2 If _x < 0 || _y < 0 || _x >= AntsForage.GRID_WIDTH || _y >= AntsForage.GRID_HEIGHT End if. If m > max There is an assignment for max to value m.
After this assignment, End if. End for. End for. There is an assignment for af.toFoodGrid.field[x][y] to value max.
After this assignment, There is a for loop that while dx < 2 There is a for loop that while dy < 2 If _x < 0 || _y < 0 || _x >= AntsForage.GRID_WIDTH || _y >= AntsForage.GRID_HEIGHT End if. If m > max There is an assignment for max to value m.
After this assignment, End if. End for. End for. There is an assignment for af.toHomeGrid.field[x][y] to value max.
After this assignment, End if. End if. There is an assignment for reward to value 0.0.
After this assignment, method terminates.