

# Simulazione di Agenti BDI basati su Prolog in Alchemist

Tesi in: Sistemi Autonomi

*Relatore:*

Chiar.mo Prof.  
Andrea Omicini

*Presentata da:*

Filippo Nicolini

*Correlatori:*

Dott. Ing. Danilo Pianini  
Dott. Giovanni Ciatto

ALMA MATER STUDIORUM – Università di Bologna  
Campus di Cesena

12 Dicembre 2019

# Contesto

## Oggi

Nel mondo ad agenti sono presenti ambienti con focus sulla programmazione oppure con orientamento alle prestazioni delle simulazioni.

## Esempio Jason

Interprete di una versione estesa di AgentSpeak che fornisce una piattaforma per sistemi multi-agente.

## Esempio MARS

Piattaforma per simulazioni distribuite di agenti programmati ad alto livello.

# Obiettivo della tesi

## Obiettivo

Unificare piattaforme orientate alla programmazione di agenti con ambienti di simulazione di agenti.

## Come

Per unificare programmazione e simulazione si è voluto portare il modello di agenti BDI all'interno di Alchemist.

## Scelte tecnologiche

Per portare il modello BDI in Alchemist si è scelto di utilizzare tuProlog per creare un interprete multi-paradigma.

# Scelte tecnologiche

## Motivazione

Alchemist e tuProlog sono progetti solidi e sviluppati indipendentemente, il cui impiego semplifica la manutenibilità del progetto.

## Alchemist

Strutturato su un meta-modello flessibile per implementare modelli diversi ed ha grande potenza di calcolo per eseguire simulazioni.

## tuProlog

Libreria Java con core minimale che include un motore Prolog utilizzabile per applicazioni e infrastrutture distribuite: è multi-paradigma, ovvero permette di integrare Prolog con piattaforme e linguaggi OO.

# Modello BDI, AgentSpeak

## Modello BDI

Modello BDI (Beliefs, Desires, Intentions) implementa gli aspetti principali del ragionamento umano per programmare agenti intelligenti.

## AgentSpeak

Linguaggio orientato agli agenti basato su modello BDI e programmazione logica per programmare agenti autonomi.

Struttura agente:

- **stato**: informazioni su se stesso e sull'ambiente che lo circonda
- **piani**: comportamento dell'agente per reagire ad eventi
- **intenzioni**: istanze di piani per perseguire attivamente un goal

# Interprete tuProlog di AgentSpeak

## Formalizzazione

Estensione di AgentSpeak realizzando un'interprete attraverso definizione di alcune sintassi.

## API – agente

- inizializzazione agente
- invocazioni verso linguaggio OO
- gestione 'belief base'
- gestione eventi e posizionamento nell'ambiente virtuale

## API – interprete

- verifica contesto e recupero corpo del piano
- esecuzione intenzione

# Alchemist

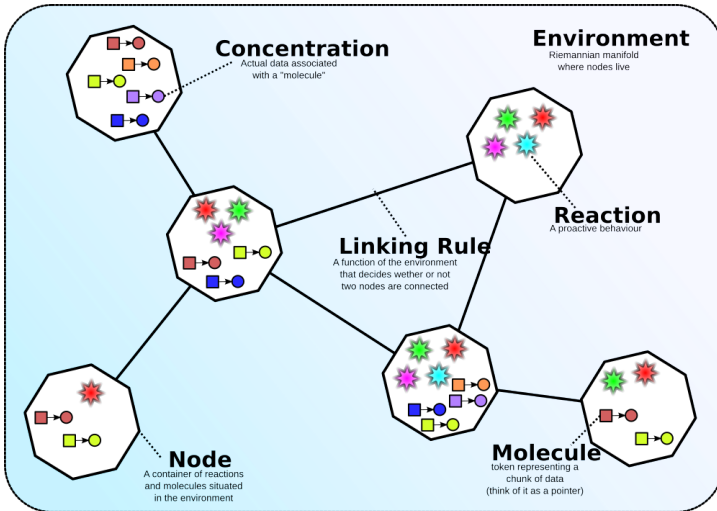
## Alchemist

Simulatore che si basa su un meta-modello flessibile e che permette implementazioni di modelli diversi tra loro.

## Meta-modello

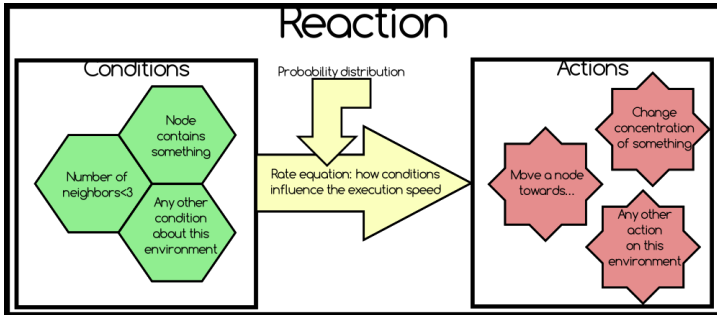
- Environment
- Node
- Linking Rule
- Molecola
- Concentrazione
- Reaction: composta da Distribuzione temporale, Condizioni, Azioni

# Meta-modello





# Meta-modello



# Spazi di tuple

## LINDA

Modello di coordinazione e comunicazione tra processi paralleli con memoria associativa, virtuale, condivisa.

## Spatial Tuples

Estensione del modello base di tuple per i sistemi distribuiti multi-agente.

- tuple posizionate nel mondo fisico
- comportamento delle primitive può dipendere dalle proprietà spaziali
- livello virtuale che aumenta la realtà fisica

# Ciclo di ragionamento

## Percezioni

Informazioni ricevute tramite un apparato con le quali l'agente percepisce i cambiamenti dell'ambiente.

## Eventi

Sono relativi a percezioni che l'agente ha ricevuto e possono essere catturati dall'agente.

## Piani

Definiscono come l'agente agisce per raggiungere goal.

## Intenzioni

Operazioni che l'agente vuole eseguire per portare a termine un certo goal.

# Unione modelli

## Mapping

Environment → Spazio agenti

Nodo → Contenitore agenti

Reazione → Agente

## Caratteristiche mapping

Ricercata la massima flessibilità lavorando su più strati: nodo può essere inteso come device in cui operano più agenti.

# Scenario

## Goldminers

Un gruppo di minatori deve recuperare pepite d'oro da miniere sparse nell'ambiente e riportarle in un deposito.

## Entità → ruoli

All'interno del problema si individuano le seguenti entità:

- minatori → agenti
- pepite → tuple
- miniere → spazi di tuple
- deposito → agente

# Realizzazione

## Minatore

Comportamento diviso in 4 stati:

- **ricerca**: spostamento casuale emettendo richieste di tuple;
- **ricezione tupla**: salva posizione della miniera e si dirige al deposito;
- **arrivo deposito**: invio pepita e si dirige alla posizione della miniera;
- **arrivo miniera**: torna stato ricerca.

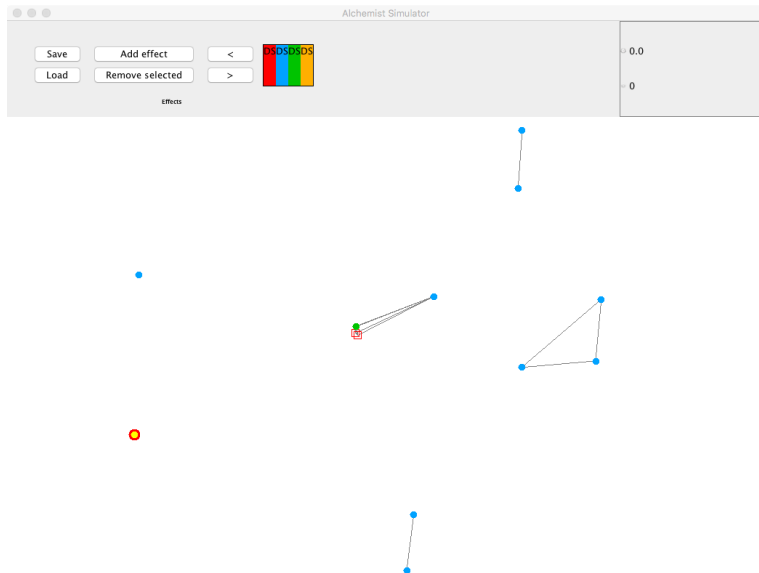
## Miniera

Istanza N tuple all'inizializzazione e risponde alle richieste dei minatori.

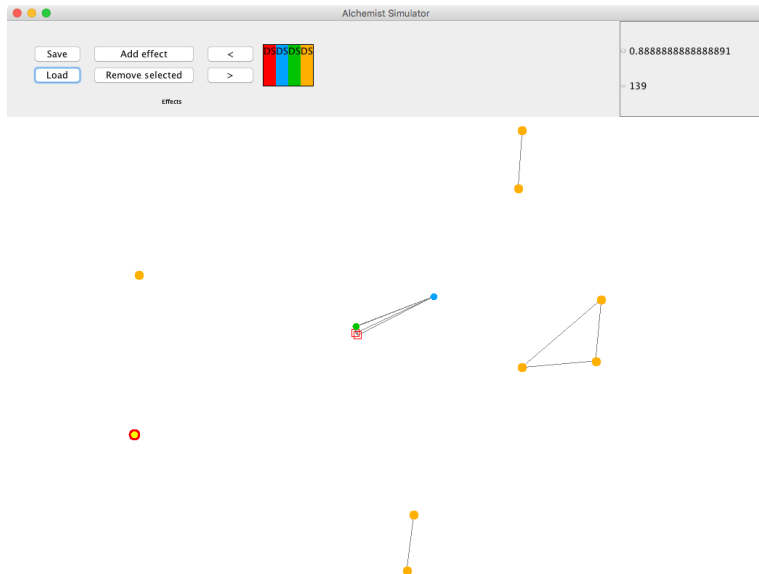
## Deposito

Statico nell'ambiente, riceve la pepita tramite un messaggio.

# Simulazione - Linking Rule

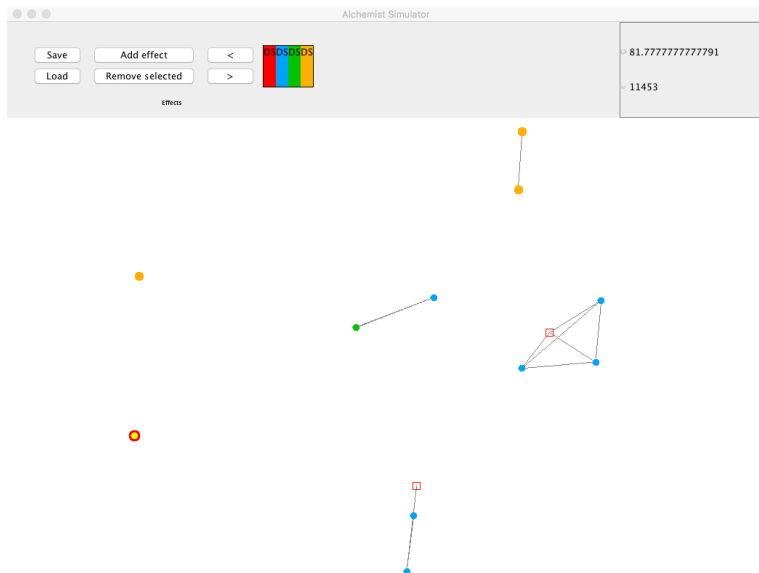


# Simulazione - Inizializzazione





# Simulazione - Esecuzione



# Conclusioni e lavori futuri

## Conclusioni

- Realizzazione di interpreti per programmare e simulare agenti
- Grande flessibilità modello e interprete (Prolog e OO)
- Estensione di spazi di tuple

## Lavori futuri

- Implementazione interprete OO in piattaforme per ambiente reale
- Ricerca per miglorie sia nell'interprete Prolog che nell'interprete OO

# Simulazione di Agenti BDI basati su Prolog in Alchemist

Tesi in: Sistemi Autonomi

*Relatore:*

Chiar.mo Prof.  
Andrea Omicini

*Presentata da:*

Filippo Nicolini

*Correlatori:*

Dott. Ing. Danilo Pianini  
Dott. Giovanni Ciatto

ALMA MATER STUDIORUM – Università di Bologna  
Campus di Cesena

12 Dicembre 2019