Sistemi Ad Agenti Per Smart Grid

Mario Di Bacco
mario.dibacco2@studio.unibo.it

Maggio 2010
Relazione di Sistemi Multi-Agente LS

Abstract

Le aree urbane sono organismi complessi e in costante evoluzione, il cui sviluppo ne condiziona il metabolismo: i consumi energetici, seppure entro un certo limite statistico prevedibili, risultano necessità comunque casuali, sia nel tempo che nelle quantità. Parallelamente, una caratteristica importante delle reti di fornitura elettrica è la quasi assenza di dispositivi di accumulazione, che sarebbero molto costosi e poco efficienti, allo stato delle attuali tecnologie. Per questo motivo è evidente come l'energia elettrica prodotta dalle centrali debba essere sempre circa pari a quella consumata: su scala nazionale, i produttori di elettricità devono modulare costantemente la produzione energetica, prevedere con complessi modelli statistici i consumi e attivare una parte di impianti di riserva all'occorrenza. Il tutto con costi enormi: solo negli USA, escludendo i costi per acquistare petrolio per produrre energia e l'energia acquistata, nel 2005 sono stati spesi 40 miliardi di dollari per gestire e mantenere il sistema energetico. Considerando inoltre il trend tecnologico nel campo della produzione energetica, si evidenzia come passare a fonti rinnovabili porterà, oltre a noti benefici, anche grandi problematiche: le fonti di energia rinnovabile soffrono della incostanza e non permanenza delle sorgenti generanti (sole, vento, maree, ecc); inoltre gli impianti di conversione possono essere ubicati solamente in determinati luoghi, ove sono presenti tali sorgenti.

Questi problemi rendono necessaria la progettazione accurata e l'attenta pianificazione sia della gestione di tali sistemi, sia soprattutto delle reti di distribuzione. I problemi appena illustrati rendono evidente come sia necessario sviluppare dei sistemi di rete elettrica capaci di una gestione intelligente ed evoluta delle risorse. In questo ambito sono stati concepiti due concetti, che analizziamo in questo documento: le smart grid (Capitolo 1) e le tecnologie loro connesse, necessarie per la gestione, ovvero i sistemi di controllo della rete distribuiti e basati su agenti (Capitoli 2 e seguenti).

Capitolo 1

Smart Grid

Una Smart Grid è una rete cosiddetta "intelligente" per la distribuzione di energia elettrica. Essa possiede strumenti di monitoraggio intelligenti, per tenere traccia di tutto il flusso elettrico del sistema, come pure strumenti per integrare energia proveniente da fonti rinnovabili e altre fonti intermittenti nella rete. La smart grid aumenta la connettività, l'automazione e la coordinazione tra i fornitori, i consumatori e la rete, per compiere al meglio i lavori di trasmissione e distribuzione dell'energia. Sarebbe in realtà corretto parlare di *reti* invece che di rete elettrica, dato che il complesso sistema che connette impianti, dispositivi di trasformazione e di consumo, sulla grande scala, non è identificabbile come una entità unica e en limitata, ma è composto da una interconnessione di sottosistemi. Le smart grid sono rivolte anche a rendere armoniosa l'integrazione delle numerose sottoreti.

Una rete intelligente può effettuare il routing dell'energia automaticamente, per rispondere alle esigenze di mercato, nonché adeguare in modo ottimale il bilanciamento topologico di domanda e offerta di energia. Per dare un'idea delle possibili applicazioni di una smart grid, si immagini che quando il costo economico dell'energia diventa minore, essa può decidere ad esempio di attivare dei processi industriali oppure elettrodomestici casalinghi, aumentando l'efficienza energetica grazie alla stabilizzazione della richiesta di energia totale alla grande rete. Una grande forza dei sistemi di reti elettriche intelligenti risiede nella possibilità di prelevare energia da una grande varietà di fonti. Tale diversità permette agli operatori della linea elettrica diverse possibilità di scelte da effettuare in casi di emergenza. Questi sono solo alcuni degli esempi che una concezione di questo tipo consente di ottenere. Per chi costruisce un sistema elettrico oggi, è certamente necessario, visto le variegate fonti energetiche, costruire nuovi siti di produzione, organizzare nuove reti elettriche, strutturare nuovi sistemi decisionali in grado di governare il flusso di energia, provvedendo nel caso ad intervenire durante eventuali cali di tensione, mancanze di energia o maggiori richieste di energia.

La smart grid è un'idea emersa nei settori di ricerca internazionali da molti anni, che porta con sé sfide tecnologiche e soprattutto investimenti notevolissimi, ma sempre più necessaria da realizzare, poiché eventi catastrofici naturali, sempre più frequenti, come pure attacchi volontari, mostrano come le interruzioni di servizio (*outage*) della rete elettrica abbiano ripercussioni socio-economiche

che vanno ben oltre quelle immediatamente prevedibili. È in questo contesto che giocheranno un ruolo fondamentale le Smart Grid.

1.1 Funzioni

Il *DOE*, Dipartimento dell'Energia degli Stati Uniti, ha di recente promosso l'iniziativa *Modern Grid Initiative*[1] proprio per promuovere la ricerca verso tale direzione, specificando anche una *Smart Grid Implementation Strategy*. Tra le funzioni necessarie ad una rete, riconducibili a questa iniziativa, sottolineiamo le seguenti:

Autoriparazione Usando sensori embedded real-time e sistemi di controllo automatici per anticipare, individuare e rispondere a problemi del sistema, una smart grid evita o riduce i problemi di sovraccarico o interruzione dell'energia. Si deve prevedere di utilizzare un sistema automatico capace di apprendimento per riuscire a individuare cause di guasto e piani di risoluzione quando vengono risolti problemi della rete con strategie efficaci.

Partecipazione dei consumatori I consumatori possono beneficiare di una riduzione dei prezzi, poiché l'efficienza energetica derivante dalla corretta gestione di una smart grid consente una notevole riduzione dei costi di gestione sostenuti dalle società operanti nel campo. Inoltre, con una rete intelligente, è molto facile per un consumatore scegliere il proprio operatore e in questo modo attivare la concorrenza sul mercato. Inoltre è più semplice immettere energia in rete per quei consumatori che posseggono piccoli impianti casalinghi: la possibilità di partecipare alla gestione dell'energia elettrica porta a parlare di *democratizzazione energetica*.

Resistenza agli attacchi Le tecnologie della rete intelligente consente anche di riconoscere e rispondere a interruzioni manuali della fornitura elettrica. Il sistema dovrebbe consentire di isolare le aree coinvolte e re-direzionare il flusso energetico verso altre aree.

Potenza di alta qualità La stabilità della tensione elettrica che giunge al consumatore è un fattore di qualità di servizio importante. Costi economici altissimi vengono affrontati per garantire questa caratteristica. Le tecnologie smart grid permettono di tenere stabile questa caratteristica molto più facilmente.

Abilitazione del marcato energetico elettrico Aumentare le possibilità d trasmettere l'energia fa in modo che siano più facili gli approvvigionamenti da soggetti diversi. Quando vendere e produrre elettricità diventa più semplice, si attivano dei meccanismi di mercato vantaggiosi per i consumatori su diverse scale.

Ottimizzazione dei guadagni Le smart grid consentono di ridurre moltissime voci dei costi di gestione delle infrastrutture.

[1] http://www.netl.doe.gov/smartgrid/

Rendere possibile l'alta penetrazione delle fonti di energia intermittenti È ormai chiaro per diversi motivi come sia fondamentale introdurre grosse quantità di energia proveniente da fonti rinnovabili. Esse però, per la maggior parte, sono intermittenti. Permettendo l'adattamento e la riorganizzazione dinamica della rete, una smart grid consentirà un uso massiccio di queste fonti.

1.2 ICT per Smart Grid

Le reti intelligenti devono essere ovviamente regolate da software di gestione adeguati, sensori e controlli intelligenti, strumenti avanzati per la pianificazione delle operazioni e per la reazione a eventi imprevisti. Tali sistemi dovranno usare una comunicazione bidirezionale, in grado di rendere le smart grid un'infrastruttura dinamica ed interattiva allo stesso tempo, capace di sfruttare un'architettura plug-and-play aperta, tale da creare un ambiente sicuro per consentire il passaggio di risorse tra le varie reti, e da consentire agli operatori sia reali che virtuali di dialogare ed interagire tra loro. Le analogie tra reti distribuite di energia e reti di telecomunicazioni peer-to-peer suggeriscono di ereditare i protocolli già studiati nel settore delle ICT e progettarli in modo da supplire alle nuove problematiche della generazione distribuita dell'energia, quali disponibilità, bassi tempi di risposta domanda/offerta, gestione delle anomalie (blackout, sovratensioni, ecc.).

Per fornire caratteristiche di intelligenza a una smart grid sono state individuate tre categorie di tecnologie: l'uso di **agenti intelligenti** distribuiti, l'uso di **strumenti analitici** (algoritmi specifici e super-computer) e l'uso di **sistemi SCADA** (Supervisory Control And Data Acquisition). I primi hanno lo scopo di agire sul sistema svolgendo dei compiti assegnati, per perseguire uno scopo comune, che come visto è quello di mantenere la rete sempre operativa e a livelli di qualità di servizio il più possibile elevati. I sistemi SCADA si occupano invece di mettere in campo gli hardware di monitoraggio e di controllo e anche i protocolli e i segnali necessari per scambiare informazioni di stato e controllo degli apparati. Si noti come oggi vengano utilizzate quasi esclusivamente queste ultime tecnologie per gestire il sistema elettrico, che tuttavia hanno bisogno dell'intervento umano per effettuare moltissime operazioni, al contrario dei sistemi multi agente.

1.3 Micro-Griglie autonome: IDAPS

Sono stati portati avanti numerosi progetti nel campo dei sistemi elettrici con sistemi ad agenti, spinti anche dalla volontà di diversi organismi politici di portare avanti ricerche su un campo strategico. Alcuni di questi progetti si sono occupati di diagnosi dei disturbi, ripristino dell'operatività, controllo del voltaggio e monitoraggio della potenza. Uno di questi progetti si è occupato della tecnologia ad agenti nel contesto delle IDAPS, *Intelligent Distributed Autonomous Power Systems*. Rispetto ad una smart grid, il contesto è più ristretto: si tratta di *microgrid*, ovvero di piccoli sottoinsiemi della rete elettrica generale. Una IDAPS è una microgrid specializzata che ha lo scopo di gestire in modo intelligente le risorse energetiche distribuite possedute dagli utenti e i carichi elettrici,

in modo che tali risorse possano essere condivise con altri utenti presenti nella microgrid sia in condizioni normali (quando la microgrid è integrata la rete generale) che di interruzione della rete elettrica (quando la microgrid è isolata dalla rete generale).

Fu l'*Advanced Research Institute of Virginia Tech [10]* a proporre il concetto di IDAPS, poiché è facilmente modellabile concettualmente e nelle simulazioni, ma soprattutto è riconducibile, dietro alcune modifiche, alla sottorete elettrica che compone il sistema di distribuzione generale.

Una IDAPS, secondo il modello citato, deve certamente essere:

Intelligente: la gestione della domanda è basata sul prezzo dell'energia, e deve assicurare il mantenimento dei carichi elettrici strategici.

Distribuita: come le risorse naturali sono distribuite, anche nella rete i dispositivi comunicano via rete (ad esempio tramite protocollo Internet Protocol).

Autonoma: la microgrid deve restare attiva in autonomia dal sistema elettrico generale, in caso di outage.

1.4 Agenti per Smart Grid

In un sistema IDAPS è molto comodo pensare di utilizzare la tecnologia dei sistemi multi agente, in modo da rispettare le caratteristiche appena viste in modo diretto: un agente ha il compito di collaborare con gli altri per **individuare blackout** sulla rete generale, e in questo caso **intervenire per rendere la microgrid autonoma** e funzionante. La tecnologia ad agenti può rendere il sistema molto più flessibile, potendo ad esempio **ridefinire i confini del sistema autonomo dinamicamente**. Le decisioni degli agenti considerano una lista di carichi elettrici da mantenere alimentati in base ad un ordine di priorità e per mantenerli utilizzano risorse interne (pannelli fotovoltaici, turbine eoliche, celle a combustibile o batterie).

Durante la *Power Systems Conference and Exposition* (Seattle 2009), un gruppo di lavoro della IEEE PES, *Power and Energy Society* ha illustrato una architettura MAS per le microgrid IDAPS. Verrà analizzato questo progetto per comprendere l'efficacia dell'applicazione dei MAS alle reti appena specificate, e verrà poi illustrato lo strumento di cui il gruppo si è servito per realizzarlo.

Per l'implementazione dell'architettura scelta, il gruppo di lavoro ha studiato molte delle numerose piattaforme open source ad agenti esistenti. Un criterio fondamentale per la scelta risiede, come vedremo, nella **compatibilità con lo standard FIPA**, *Foundation for Intelligent Physical Agents*, per garantire interoperabilità tra sistemi ad agenti eterogenei. Due sono i sistemi che il gruppo ha ritenuto validi: *Zeus* e *JADE*. Tra questi due la scelta è ricaduta su Zeus, che come spiegato più avanti è un framework Java based che usa la comunicazione ACL, ha alcune caratteristiche orientate alla sicurezza ed una interfaccia GUI abbastanza avanzata.

Non possiamo analizzare in questa sezione i dettagli del progetto, prima che il framework Zeus venga approfondito. Un riepilogo di come è stata portata a termine ogni fase del progetto del sistema appena citato verrà descritta quindi nel Capitolo 3.

Capitolo 2

Zeus Agent Toolkit

Zeus Agent Building Toolkit è un ambiente integrato per MAS sviluppato dai British Telecommunications Labs[1]. Come detto, è implementato in Java ed è open source[2]. Ha diverse funzioni user friendly che permettono la comunicazione e consentono la negoziazione per lo scambio di risorse (nel caso del progetto visto, la potenza elettrica) molto facili e veloci da istanziare. Complessivamente, i progettisti di Zeus hanno cercato di nascondere la complessità che c'è dietro i molti tool per MAS, fornendo un framework per quanto possibile general purpose e personalizzabile. L'obiettivo dei progettisti era quindi permettere di utilizzare Zeus ad ingegneri del software con una conoscenza dei sistemi ad agenti rudimentale.

La filosofia di Zeus rispecchia la necessità dei ricercatori di avere a disposizione metodologie e tool-kit per risolvere problemi, invece che soluzioni mirate a problemi specifici. Per far ciò, esso fornisce una libreria di componenti basati su agenti e un ambiente che li supporta. Si può considerare Zeus come una sintesi di tecnologie per agenti già affermate, unite ad alcune soluzioni innovative per sviluppare applicazioni in modo rapido e integrato.

2.1 Componenti di Zeus

I progettisti di Zeus considerano che Java sia un ottimo linguaggio per sviluppare una piattaforma, grazie al paradigma ad oggetti, al multithreading, alle librerie per la comunicazione fornite e per la portabilità. Essi hanno strutturato Zeus in tre gruppi funzionali di componenti:

[1] Sito ufficiale del progetto: www.labs.bt.com/projects/agents/zeus
[2] Source code di Zeus disponibile qui: http://sourceforge.net/projects/zeusagent/

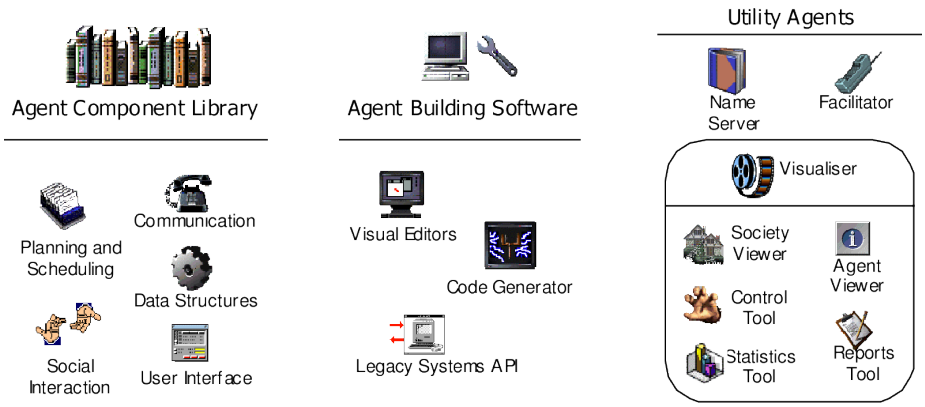

Figura 2.1: Struttura del framework Zeus

Libreria di componenti agente Comprende i mattoni con cui costruire gli agenti, ossia gli oggetti che implementano le funzioni di livello agente. Vi è un linguaggio di comunicazione inter-agente compatibile con ACL, un sistema di rappresentazione della conoscenza e di storage, un sistema a scambio di messaggi asincrono. Il toolkit fornisce anche un sistema di scheduling e si possono utilizzare eventi. Usando le API evento è possibile personalizzare il comportamento dell'agente, monitorandone lo stato interno e intervenendo dall'esterno.

La cooperazione tra agenti è conseguita tramite un motore sensibile al contesto, ai ruoli, alle risorse e alle strategie di cooperazione. Come detto, si è puntato molto sulla forza delle librerie: in questo ambito sono fornite due famiglie di protocolli già pronti, una per i protocolli di coordinazione e una per le relazioni organizzative tra gli agenti.

Tool di visualizzazione Sono degli strumenti che permettono di catturare lo stato del sistema in esecuzione in modo globale. È questa una necessità fondamentale per testare il risultato dell'implementazione di un sistema complesso distribuito in più entità. Come visibile dalla Figura 2.1, il *visualiser* offre una famiglia di tool di visualizzazione, che interrogano continuamente lo stato e i processi di ogni agente e ne rappresentano lo stato globale in maniera differenziata: il *society viewer* ne mostra la forma organizzativa con le relazioni, lo *strumento report* mostra i task attivi, decomponendoli per ogni agente nei sottotask, l'*agent viewer* mostra lo stato di un agente, il *tool di controllo* permette di modificare runtime lo stato di un agente, ovvero i task correnti, le risorse che possiede, le relazioni con gli altri agenti e anche le strategie di coordinazione; infine lo *strumento statistiche* mostra in vari formati le informazioni collezionate, sia sulla società completa, sie per singolo agente.

Software per costruire agenti La filosofia che emerge dal processo di sviluppo Zeus è quella di offrire al progettista implementazioni già funzionanti per la comunicazione, il ragionamento e la cooperazione, per lasciarlo concentrare sulla definizione degli agenti necessari all'applicazione, che si realizza scrivendo il codice necessario a dotare l'agente di capacità specifiche del dominio applicativo.

Questa filosofia necessità però che il progettista sappia cosa aspettarsi dal framework. Per soddisfare tale aspetto, i progettisti di Zeus hanno definito delle

metodologie di sviluppo, parallelamente al software, con l'intenzione di guidare gli utenti alla creazione degli agenti, come vediamo di seguito.

2.2 Metodologia di sviluppo di Zeus

La metodologia di Zeus da adottare nella progettazione ha lo scopo di nascondere ai progettisti molti aspetti complessi dei sistemi MAS. Tale metodologia è suddivisa in sei fasi:

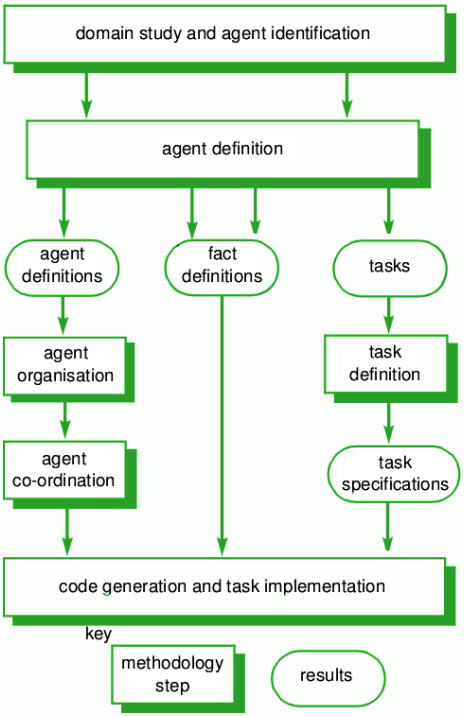

Figura 2.2: Fasi di sviluppo secondo la metodologia Zeus

Studio del dominio e identificazione degli agenti In questa fase lo sviluppatore analizza il problema per identificare i potenziali agenti. È difficile in questa fase catturare la granularità dell'agente, dato che un agente è qui semplicemente una generica entità autonoma, che agisce indipendentemente dall'intervento umano.

Definizione degli agenti Individuati gli agenti candidati, si passa a definirne gli attributi significativi. In questa fase si pensa ad un agente che ha un numero di risorse iniziali limitate (i *facts*) e che può eseguire un certo numero di task simultanei finché ha risorse. Bisognerà quindi definire quali risorse iniziali vi sono e che task è possibile svolgere.

Organizzazione degli agenti La fase di organizzazione può avere luogo quando tutte le definizioni sono state portate a termine. Qui lo sviluppatore

delinea i conoscenti di un agente: un certo agente A si avvicina al conoscente B quando crede che B ha delle risorse che gli occorrono. Le conoscenze dipendono anche dalla community a cui gli agenti appartengono, e al grado di subordinazione che possono avere tra loro.

Definizione dei task I task, che sono stati identificati durante la fase di definizione degli agenti, vengono ora definiti in termini di costo, durata, precondizioni, prodotti e vincoli. I task possono essere sia primitivi che composti.

Coordinamento degli agenti Il processo prosegue con la fase di coordinamento, in cui si scelgono dei protocolli di cui gli agenti hanno bisogno per interagire socialmente e svolgere i propri compiti (protocolli master/slave, protocolli di vendita di risorse, ecc).

Generazione del codice e implementazione dei task Il termine dello sviluppo è raggiunto quando si è pronti per generare il codice: giunti a questo punto il tool di generazione del codice ha tutti gli elementi per generare un codice sorgente per l'agente completo. Oltre a questo, vengono generati anche degli stub di codice per poter agganciare l'applicazione a strumenti software esterni (un esempio rilevante è il database).

Si noti come le fasi appena descritte costituiscono solo un outline del processo di sviluppo reale, dato che ogni fase è costituita da sotto-fasi, come si può facilmente intuire facendo una analogia con le metodologie di sviluppo dei software object oriented.

2.3 Agenti in Zeus

Ogni agente Zeus può essere logicamente diviso in tre livelli: quello di definizione, quello organizzativo e quello di coordinamento.

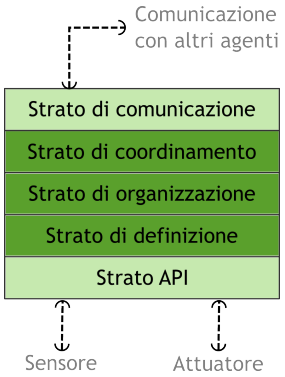

Figura 2.3: Macro struttura di un agente Zeus

Il livello di *definizione* di un agente serve a specificare ad esempio quali capacità di ragionamento ha l'agente, i suoi goal, le risorse, le abilità, le convinzioni, le preferenze.

Il layer *organizzativo* specifica le relazioni dell'agente con i suoi simili, come ad esempio l'agenzia sotto cui agisce, le funzioni che attribuisce ai suoi vicini o quali relazioni di autorità vi siano tra esso e gli altri.

Lo strato di *coordinamento* modella l'agente come una entità sociale, che negozia risorse con gli altri e si coordina con essi. Come visto, esistono librerie di protocolli già pronti per raggiungere questi scopi.

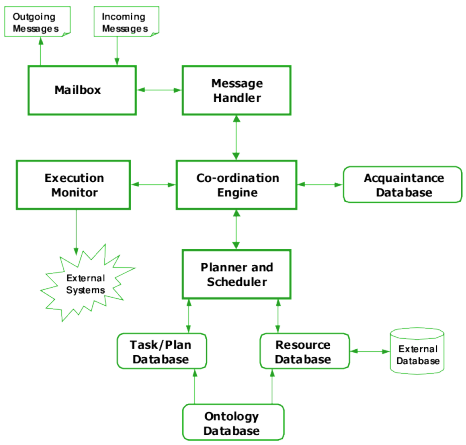

Figura 2.4: Struttura logica dei componenti di un agente Zeus

Zeus prevede che alcune delle funzioni, che normalmente sono necessarie in un sistema MAS, come ad esempio il name server, siano svolte da agenti specializzati: questi agenti sono inclusi nelle librerie di componenti per agenti e vengono chiamati *utility agents* e sono una importante porzione del sistema Zeus, come evidente dalla Figura 2.1. Le funzioni offerte dagli utility agent spaziano dal *name server*, che mappa nomi agente a indirizzi di rete, al *facilitator*, che fornisce i nomi di possibili agenti capaci di una certa capacità richiesta, come delle "Pagine Gialle".

2.4 Processo di sviluppo nel dettaglio

La suite di sviluppo di Zeus è stata elaborata per portare a termine il lavoro di implementazione in modo il più possibile interattivo. Per raggiungere questo scopo è stato incluso nell'IDE un editor per ogni aspetto della metodologia visto sopra, durante l'analisi della stessa. Il progettista deve elaborare tutte le fasi previste dalla metodologia.

Creazione dell'ontologia L'editor dell'**ontologia** (Figura 2.5) è usato per specificare quali oggetti ontologici (concetti, attributi e valori) sono validi nel dominio scelto. Esso consiste praticamente in un insieme di **facts/variabili**, elaborati nel dettaglio in un editor apposito, che ne specifica *nome, tipo, vincoli* e *valori di default*.

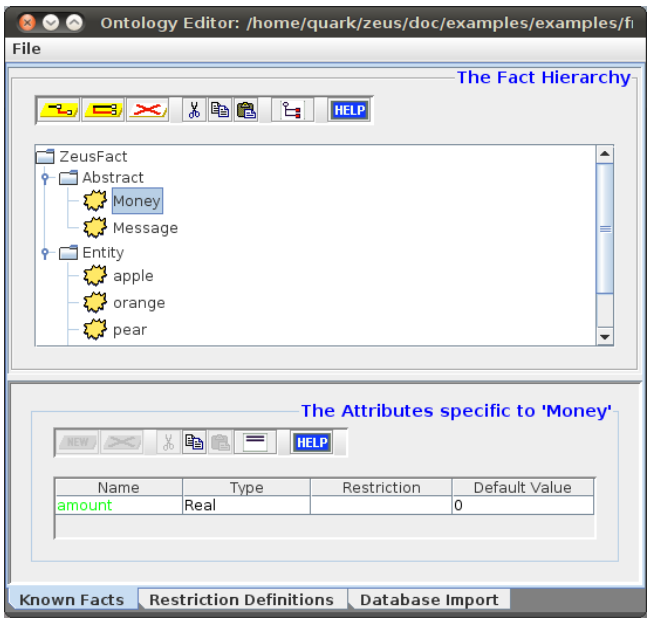

Figura 2.5: Editor dell'ontologia

Creazione di un agente La creazione di un agente è divisa in cinque sotto-fasi.

Inizialmente, un editor (Figura 2.6) provvede alla **definizione degli agenti** per associarvi *task* (che possono essere composti) e *risorse iniziali*.

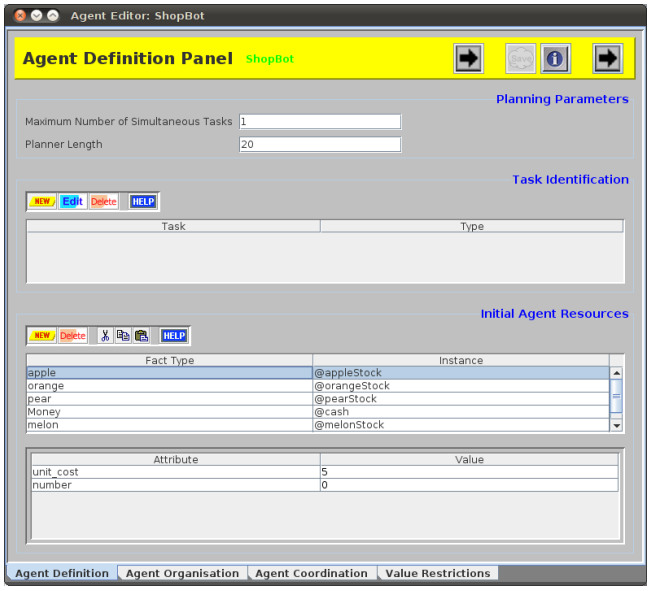

Figura 2.6: Editor dell'agente

Due strumenti di editing elaborano la descrizione dei **task** dell'agente: *precondizioni* (risorse necessarie per completare il task), *effetti, costo, durata* e *vincoli generali* (Figura 2.7).

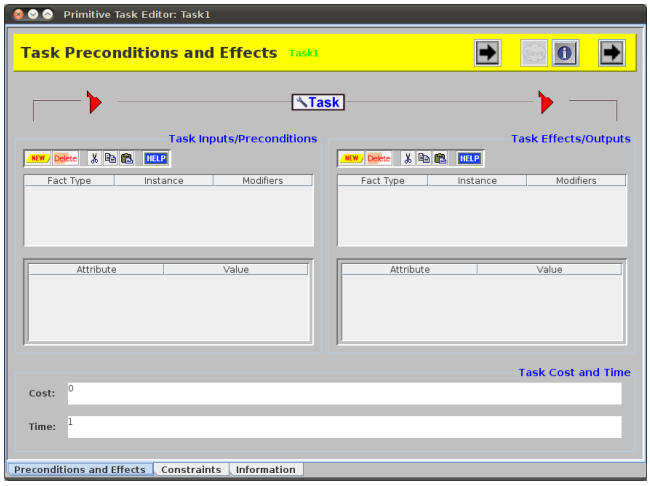

Figura 2.7: Editor del task

Anche l'**organizzazione** (per le relazioni tra gli agenti) è costruita con un editor (Figura 2.8): si specificano i conoscenti di ogni agente, dandone la *relazione* (superiore, subordinato, pari o collaboratore) e le *risorse* che l'agente suppone essi abbiano, in termini di facts.

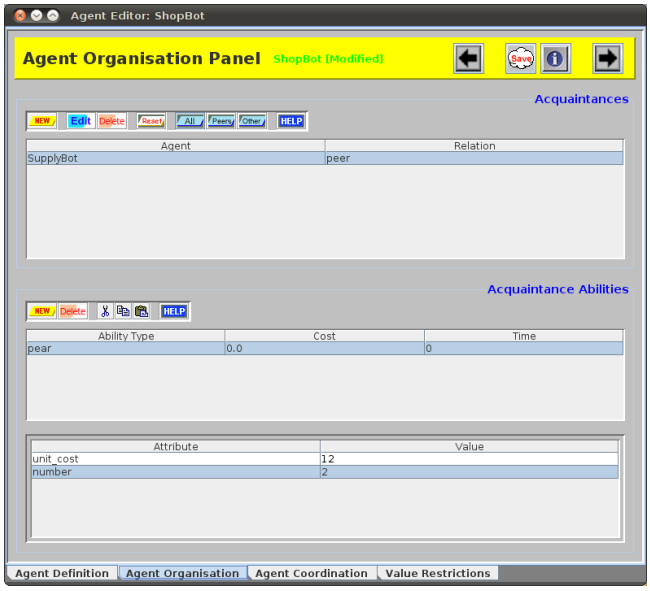

Figura 2.8: Editor dell'organizzazione agenti

Con l'editor del **coordinamento** (Figura 2.9) si selezionano i *protocolli di*

coordinamento di cui viene dotato ogni agente. Per ogni protocollo si possono elaborare delle *strategie di interazione*, sia scegliendole da una libreria sia implementandole.

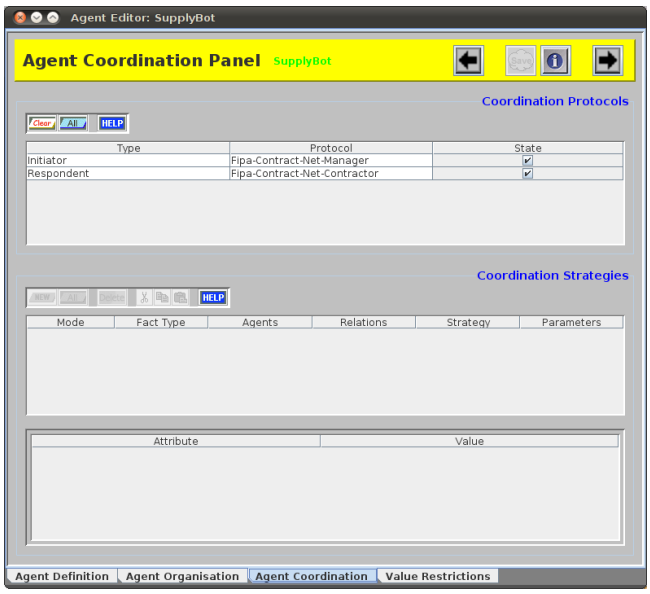

Figura 2.9: Editor del coordinamento agenti

Infine sono definite le *precondizioni* (sempre sei facts) e le *azioni* (modificare un fact, eseguire un task o impostare un goal) in una **rule base.**

Configurazione degli utility agent Gli *utility agent* vengono editati per fornire supporto alla società degli agenti attraverso servizi spesso fondamentali, quali un *name server*, un *facilitator* o un *visualizer* dei log, già descritti in precedenza.

Configurazione dei task agent Alcuni compiti operativi specifici dell'applicazione devono essere definiti in agenti appositi, i *task agent*: si tratta ad esempio di operazioni di collegamento a software esterni (front end grafici) oppure a sorgenti di dati esterne, che chiaramente non dovrebbero essere affidate ad un agente normale. È previsto che per ottenere un task agent venga ridefinito il comportamento di un agente generico. Si noti che esso non è un utility agent.

Implementazione Completate le elaborazioni di ogni aspetto con il proprio editor, lo **strumento di generazione del codice** genera i sorgenti Java degli agenti, includendo tutte le classi implicate, ed eventualmente anche sistemi legacy, quando necessario. La **compilazione** del codice è integrata nell'IDE. A questa segue naturalmente l'**esecuzione del sistema**: rientra anch'essa nel processo di sviluppo, poiché la fase di test implica che siano monitorate le dinamiche del sistema. Anche per questa necessità il framework Zeus offre i **tool di visualizzazione** visti precedentemente.

Capitolo 3

Implementazione di un sistema ad agenti per microgrid

Riassumiamo qui le caratteristiche del multi agent system previsto dal progetto del gruppo di lavoro della IEEE PES, nell'ambito delle microgrid IDAPS. Il progetto software è stato portato avanti seguendo cinque fasi di progetto:

Specifica degli agenti Il sistema MAS è composto da quattro tipologie di agenti, di cui riassumiamo le specifiche.

Agenti Controllori: si occupano del monitoraggio del voltaggio e della frequenza, mandano segnali agli interruttori generali per isolarsi dalla rete e inoltrano il prezzo dell'energia che ricevono dalla rete agli altri agenti.

Agenti Risorse: monitorano le sorgenti di energia locali, analizzandone il livello di potenza, lo stato e la disponibilità.

Agenti Utente: si occupano di fornire all'utente finale le informazioni sulla rete e di analizzare i livelli di assorbimento di corrente da parte degli utenti. Consentono anche all'utente di specificare le priorità di funzionamento dei propri dispositivi alimentati.

Agenti Database: memorizzano le informazioni in un database, accessibile sia dagli agenti che dagli utenti.

Ogni agente avrà ovviamente un proprio **goal specifico** e le relative responsabilità, come visto poco sopra. Tuttavia, lavorando in cooperazione, i quattro tipi di agenti perseguono un **goal generale** che corrisponde a mantenere in vita i dispositivi critici della rete durante le interruzioni di corrente.

Analisi dell'applicazione Le specifiche logiche precedentemente viste vengono formalizzate in ruoli e responsabilità usando delle tecniche di modellazione dei ruoli. Ci si è serviti di un diagramma di collaborazione, che definisce le interazioni tra gli agenti e quelle tra agenti e ambiente. Il diagramma è illustrato in Figura 3.1: si vedono i tre tipi di agenti dialogare tra loro, mentre il database agent è una composizione di facilitator e name server, che come visto sono due utility offerte già dal framework Zeus. In figura di vede anche il visualizer, anch'essa una utility Zeus. Per la comunicazione si è usato il protocollo TCP/IP.

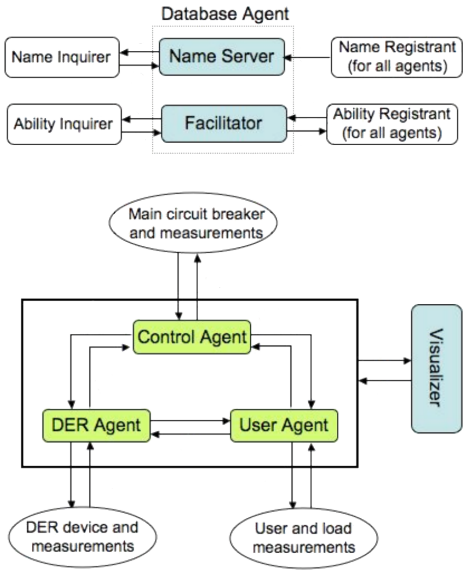

Figura 3.1: Diagramma di collaborazione

Progetto dell'applicazione La fase di progettazione prevede la modellazione della conoscenza di ogni agente, che come visto è chiamata in Zeus *fact*. Nel progetto sono stati elencati i seguenti facts: island mode (boolean), energia (kWh), interruttori dei carichi (aperti/chiusi), costo per i generatori distribuiti (cent/KWh), prezzo dell'elettricità (cent/kWh), richiesta di potenza per un carico (kW), potenza prodotta da un generatore (kW), stato dei generatori (on/off).

Anche gli agenti sono stati progettati nel dettaglio, specificando per ognuno le responsabilità sociali approfondite per il dominio e con il relativo progetto applicativo. Dato che le responsabilità nel dettaglio sono un approfondimento e una estensione di quelle illustrate durante la fase di specifica degli agenti, si evita in questo contesto di appesantire la trattazione elencandole tutte.

Realizzazione dell'applicazione Questa fase consiste nel creare concretamente, attraverso l'interfaccia grafica del framework, gli agenti e nel modellarne l'ontologia e i task, in accordo con il progetto. L'*ontologia* è una sorta di vocabolario che l'agente usa per comunicare, e consiste di facts, vincoli, tipi, attributi e valori default. Nel progetto sono state realizzate le entità fact elencate nella fase di progettazione.

La *creazione degli agenti* è la fase successiva, che consta dapprima di una generazione di agenti generici (uno per ognuna delle tre tipologie, oltre all'agente database che è già incluso nel framework), e poi dell'attribuzione a ognuno di essi di task concreti, ruoli, protocolli di coordinazione e stato. L'IDE permette di concretizzare questi aspetti in modo integrato, tutto attraverso l'interfaccia grafica. Si noti che nel progetto, gli agenti sono considerati tutti pari e sono reattivi, cioè agiscono sulla base dei facts che percepiscono. La reazione avviene in base alle regole impostate.

La *configurazione degli utility agent* serve a mettere a punto gli agenti già pronti, previsti nella libreria del framework: il database agent (che come visto funge da name server e facilitator) e il visualizer agent, che scrive a video tutti i messaggi che circolano nel sistema. La *configurazione dei task* è la fase in cui il programmatore può collegare il codice interno del task agente con software sterno o scrivendone le classi di collegamento.

Il processo realizzativo termina con la *generazione del codice*, un passo immediato e automatico. Dopo questa il MAS può essere già compilato e messo in esecuzione.

Implementazione del test Il gruppo di lavoro ha realizzato un circuito in Matlab/Simulink, per simulare la rete di distribuzione elettrica. Nel circuito sono stati inclusi generatori, interfacce di griglia, carichi e interruttori. Il team ha poi interfacciato questo modello con un server TCP/IP, in modo da rendere possibile la comunicazione e il controllo dei parametri di simulazione in modo standard. Su un'altra workstation è stato messo in esecuzione il sistema MAS, opportunamente configurato in modo che gli agenti potessero agire da sensori e attuatori sui giusti componenti del circuito simulato, via rete. La simulazione, il cui risultato in termini di andamento in forma d'onda di tensione e corrente è visibile in figura, mostra come un outage sulla rete esterna non metta in crisi la microgrid: in circa 0.008 secondi gli agenti riescono a individuare l'evento, decidere quali carichi elettrici della microgrid devono essere disconnessi, decidere quanta potenza elettrica deve essere prodotta dai generatori e riconfigurare il circuito ponendolo in *island-mode*, cioè con la microgrid isolata.

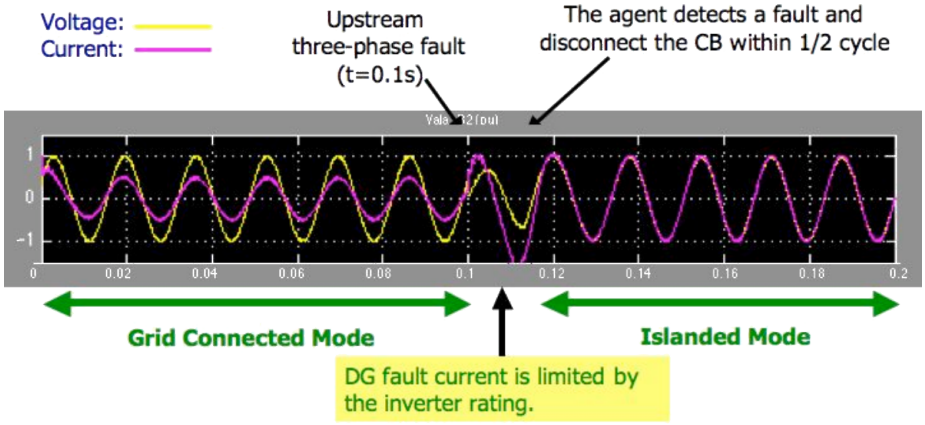

Figura 3.2: Simulazione realtime in Matlab/Simulink di un outage

Per i dettagli sull'implementazione e sulle informazioni di simulazione è possibile consultare l'articolo [6].

Capitolo 4

Valutazioni conclusive sulla piattaforma Zeus

Le metodologie per l'analisi e la progettazione dei sistemi multi-agente, proposte negli ultimi anni in diverse forme, mancano tutte di completezza nel processo di sviluppo. Dato che nello stesso tempo sono state progettati diversi framework per lo sviluppo di sistemi MAS, si capisce come sia evidente il gap concettuale esistente tra esse. Zeus è un ambiente di sviluppo che include un processo di sviluppo per ottenere un MAS funzionante. Tuttavia, per mantenere salde le caratteristiche di rapidità del processo, fortemente volute dai progettisti di Zeus, gli *aspetti concettuali* del processo di sviluppo sono celati dietro le *attività* che Zeus prevede per completare la fase di realizzazione del sistema. In questo modo, non è molto chiaro allo sviluppatore il mapping tra la metodologia e la procedura messa in pratica.

Lo *ZEUS Tehchnical Manual* [2] [1] dichiara che "*the agent creation methodology is vital to the use of the Zeus toolkit*". Lo sforzo di enfasi che sottolinea tale aspetto è evidente nell'abbondante documentazione di cui il progetto è fornito. Facendo un confronto con le altre piattaforme [14], Zeus risulta avere una completa integrazione tra le fasi di analisi, progettazione, sviluppo e deployment. Esso utilizza strumenti di sviluppo sia teorici che pratici dell'ingegneria del software, come i design pattern.

Nonostante ciò, il lavoro di Ricordel-Demazeau [14] riconosce a Zeus dei limiti: esso offre un solo modello di agente: l'agente astratto, di cui il programmatore deve re-implementarne alcuni aspetti. Questa caratteristica limita i confini progettuali di un sistema MAS complesso. La curva di apprendimento di Zeus inoltre non è molto ripida; tuttavia, una volta appresa la metodologia, secondo gli autori, esso si rivela una piattaforma produttiva che porta dei benefici notevoli.

Rispetto ad altre piattaforme, Zeus è riconducibile ad un editor per agenti BDI invece che un framework per progettare con un linguaggio di programmazione orientato ad agenti: se da un lato è efficiente (considerando la qualità del software rispetto al costo del processo) e lavora ad un alto livello di astrazione, dall'altro l'architettura dell'agente è statica, e ciò ne fa un framework

[1] Documento disponibile qui: http://www.upv.es/sma/plataformas/zeus/Zeus-TechManual.pdf

meno versatile di altri, in cui gli agenti possono essere programmati a più basso livello.

La valutazione finale del prodotto in esame, come pure confermato dallo studio citato, risulta essere complessivamente molto buona.

Bibliografia

[1] Smart Grid, Filippo Fruscalzo, Francesco Morassuti e Andrea Rossato, progetto FSE Tekne-Smart Grid Cavanis

[2] Smart Grid alla Sapienza, Livio de Santoli

[3] Smart grid, da Wikipedia, the free encyclopedia

[4] ZEUS: A Toolkit for Building Distributed Multi-Agent Systems, Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee & Jaron C. Collis

[5] Update: Agent Standardization Efforts, James Odell

[6] Multi-Agent Systems in a Distributed Smart Grid: Design and Implementation, Pipattanasomporn, Feroze, Rahman

[7] ZEUS: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems, Hyacinth S. Nwana, Divine T. Ndumu & Lyndon C. Lee

[8] The ZEUS agent building tool-kit, J C Collis, D T Ndumu, H S Nwana and L C Lee, BT Technol J Vol 16 No 3

[9] The Zeus Agent Building Toolkit, Jamie Stark

[10] Intelligent Distributed Autonomous Power Systems (IDAPS), Saifur Rahman, Manisa Pipattanasomporn, Yonael Teklu

[11] AgentAirPol System, an Agent-based System for Air Pollution Analysis, Elia Petre

[12] The Application Realisation Guide, Collis, Ndumu, Intelligent Systems Research Group, BT Labs

[13] Derivation of ZEUS Concepts from the GAIA Methodology to Develop Multi-Agent Systems, Villarreal, Martínez, Povolo, Stuchi, Chiotti, GIDSATD Research Group

[14] From Analysis to Deployment: a Multi-Agent Platform Survey, Pierre-Michel Ricordel, Yves Demazeau, LEIBNIZ laboratory

[15] ZEUS Technical Manual, Jaron Collis, Divine Ndumu, Intelligent Systems Research Group, BT Labs