Specifica, Implementazione ed Esecuzione di un Prototipo di Sistema Multi-Agente in D-CaseLP

Riccardo Albertoni* Maurizio Martelli† Viviana Mascardi†
Stefano Miglia†

* IMATI – Istituto di Matematica Applicata e Tecnologie Informatiche, Sezione di Genova
Consiglio Nazionale delle Ricerche Via De Marini 6, 16149 Genova, Italy. Email: albertoni@ima.ge.cnr.it

† DISI – Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova,
Via Dodecaneso 35, 16146, Genova, Italy.
Email: {martelli,mascardi}@disi.unige.it, stefano@actarus.vislink.it

***Abstract*— Una tecnologia giovane come quella ad agenti viene tipicamente impiegata in contesti con un alto contenuto innovativo dove l'informazione di cui gli agenti necessitano per svolgere i propri compiti è distribuita tra i vari componenti del sistema e richiede la attuazione di sofisticati protocolli di coordinazione e cooperazione per essere condivisa. Per sviluppare sistemi con queste caratteristiche, i cui requisiti possono essere all'inizio instabili o poco chiari, strumenti e metodologie che supportino la prototipazione rapida si rivelano estremamente utili.**

D-CaseLP (*Distributed Complex Application Specification Environment based on Logic Programming*) è un ambiente di sviluppo che fornisce un supporto alla specifica, implementazione, esecuzione e debugging di prototipi di sistemi multi-agente. Esso offre la possibilità di specificare, usando UML ed altri linguaggi, le architetture degli agenti (ovvero, la loro struttura interna ed il meccanismo che ne regola il funzionamento), i ruoli che essi devono ricoprire ed i protocolli di interazione tra ruoli, le ontologie che essi comprendono ed infine la logica del loro comportamento. D-CaseLP consente di tradurre in maniera automatica le specifiche UML in regole del sistema esperto JESS ed utilizza gli strumenti messi a disposizione da JADE, una piattaforma FIPA-compliant, per eseguire e testare il prototipo risultante.

I. Introduzione

Negli ultimi anni abbiamo assistito ad una significativa crescita dell'interesse verso il paradigma e la tecnologia ad agenti. Tale interesse non è circoscritto agli ambienti di ricerca, ma vede sempre più spesso la partecipazione di soggetti provenienti dal mondo dell'industria. Nonostante ciò, non si può ancora considerare la tecnologia ad agenti come una tecnologia affermata nei contesti applicativi: come analizzato in [23], l'eterogeneità della concettualizzazione del paradigma ad agenti, la mancanza di strumenti di sviluppo adeguati e la mancata individuazione di un campo applicativo in cui il paradigma ad agenti sia decisamente più appropriato degli altri rappresentano un freno alla sua affermazione. Recentemente sono stati effettuati alcuni significativi passi nella risoluzione dei problemi sopra citati. In particolare, il processo di standardizzazione di cui la Foundation for Intelligent Physical Agents (FIPA [12]) si è fatta carico ha posto le basi per una concettualizzazione degli agenti largamente condivisa che viene supportata da diverse piattaforme denominate "FIPA-compliant". Per quanto riguarda il dominio di applicazione in cui gli agenti possono dimostrare le loro potenzialità, vi è un grande interesse da parte delle aziende che lavorano nel campo delle comunicazioni. In particolare, l'attenzione posta alle problematiche dell'interoperabilità fra sistemi aperti ha candidato la tecnologia ad agenti come un possibile strumento per il superamento del web inteso come entità esclusivamente passiva e al fine di ottenere una "rete intelligente". I due aspetti cruciali in questo campo applicativo sono quindi la *eterogeneità* degli agenti appartenenti a diversi sistemi aperti, caratterizzati quindi da architetture diverse ed implementati usando linguaggi differenti, e la loro *capacità di comunicare* seguendo protocolli di interazione sofisticati.

D-CaseLP [1], [20], [19] è uno strumento di prototipazione rapida progettato e sviluppato dal gruppo di Programmazione Logica del Dipartimento di Informatica e Scienze dell'Informazione dell'Università di Genova in una versione funzionante anche se non ancora completa di tutte le funzionalità previste. D-CaseLP è una evoluzione di CaseLP [17], [5], [18], [16] ed ha tra i suoi obiettivi il supporto alla specifica ed alla prototipazione rapida di *agenti eterogenei* e dei *protocolli di interazione* che gli agenti devono seguire per raggiungere i propri scopi. La eterogeneità degli agenti si manifesta a tre livelli diversi:

- Eterogeneità del linguaggio con cui l'agente è specificato: D-CaseLP offre un insieme di linguaggi diversi per la specifica degli agenti, o di particolari aspetti del loro funzionamento. Lo sviluppatore può scegliere il più adatto tra essi in base al proprio scopo ed alle proprie preferenze.
- Eterogeneità della architettura dell'agente: D-CaseLP consente di specificare agenti con architetture diverse (si veda la Sezione II).
- Eterogeneità del linguaggio di implementazione dell'agente: D-CaseLP si basa su JADE [3], una piattaforma FIPA-compliant per lo sviluppo di sistemi ad agenti implementata in Java. Grazie alla possibilità di integrare Java con numerosi linguaggi di programmazione mediante la "Java Native Interface" (JNI), ed alle interfacce già implementate tra Java e vari linguaggi (si veda ad esem-

pio `http://www.cs.huji.ac.il/~sdbi/1999/alicser/projects.html` per sistemi che integrano Java e Prolog), agenti implementati in linguaggi diversi (purché integrabili con Java, e quindi con JADE) possono coesistere in un unico prototipo D-CaseLP.

Riguardo ai protocolli di interazione, D-CaseLP offre una serie di strumenti per tradurre in maniera automatica i diagrammi UML che definiscono i protocolli tra i ruoli, l'associazione tra le classi di agenti ed i ruoli che esse ricoprono e le istanze di agenti che appartengono alle varie classi. Il risultato della traduzione è un insieme di regole del sistema esperto JESS per ogni classe di agente. Le regole devono essere completate manualmente con i dettagli non contenuti nelle specifiche ma necessari all'esecuzione. Dopo questo completamento viene definito, sotto forma di fatti JESS, lo stato iniziale di ogni istanza di agente. Grazie ad una interfaccia tra JESS e JADE, i programmi JESS così ottenuti possono essere eseguiti in JADE sfruttando tutti gli strumenti di monitoraggio e debugging che JADE offre.

D-CaseLP nasce come estensione di CaseLP e ne eredita ed estende la metodologia di sviluppo, la concettualizzazione degli agenti ed i vari livelli di astrazione che caratterizzano un sistema ad agenti. Rispetto a CaseLP, che utilizza Prolog come linguaggio "target" per la implementazione del prototipo, D-CaseLP utilizza il linguaggio a regole del sistema esperto JESS (`http://herzberg.ca.sandia.gov/jess/`). La scelta di JESS, che condivide con Prolog la dichiaratività e la struttura a regole, è stata motivata dalla sua integrabilità in JADE; l'utilizzo di JESS e di JADE ha consentito di superare i due limiti maggiori di CaseLP: la centralizzazione e la mancanza di supporto alla esecuzione concorrente degli agenti. Stiamo attualmente lavorando alla integrazione in JADE di un interprete Prolog. Il principale obiettivo di tale integrazione è poter usufruire in D-CaseLP degli strumenti che CaseLP fornisce per la traduzione in Prolog dei linguaggi di specifica ad alto livello $\mathcal{E}_{hhf}$ [7], [8] ed AgentRules [19]. Rispetto a JADE, D-CaseLP offre la parte teorica di concettualizzazione degli agenti e del sistema multi-agente, la metodologia di sviluppo consolidata in anni di esperienza con CaseLP e la possibilità di tradurre in JESS ed eseguire in JADE agenti che rispettano protocolli di interazione ed i diagrammi per la definizione delle classi e delle istanze specificati in UML.

In questo articolo introdurremo le definizioni dei principali concetti che caratterizzano gli agenti D-CaseLP (Sezione II) e la nostra metodologia di sviluppo (Sezione III). Ci concentreremo in particolare sugli aspetti metodologici relativi alla modellazione di diagrammi per la specifica dei ruoli e protocolli di interazione tra ruoli (Sezione IV) ed alla modellazione delle associazioni tra le classi di agenti, i ruoli che interpretano, la architettura interna che le caratterizza e le istanze di agenti appartenenti alla classe (Sezione V). Nella sezione VI illustreremo come avviene la traduzione dai diagrammi UML in codice JESS e come gli agenti ottenuti possono essere eseguiti in JADE. Un breve esempio (Sezione VII) mostrerà in pratica come adottare la metodologia di sviluppo e gli strumenti offerti da D-CaseLP. Proporremo infine un confronto tra CaseLP e strumenti analoghi esistenti e le possibili direzioni di sviluppo futuro (Sezione VIII).

II. Concettualizzazione degli agenti in D-CaseLP

Dal punto di vista computazionale, gli agenti in D-CaseLP sono caratterizzati da

- uno *stato*,
- un *programma*,
- un *"engine"*.

Le componenti contenenti lo stato ed il programma dell'agente, insieme all'engine operante su di esse, determinano l'architettura dell'agente. Lo stato contiene le informazioni che, durante l'esecuzione dell'agente, possono subire modifiche mentre il programma contiene le informazioni che non cambiano durante l'esecuzione. L'engine, infine, governa l'esecuzione dell'agente fornendo un meta-interprete per il programma e per i dati (stato) su cui il programma opera.

Da un punto di vista descrittivo, un agente è caratterizzato da tre caratteristiche ortogonali:

- la sua *architettura*,
- i *ruoli* che esso ricopre,
- le *ontologie* che esso conosce.

Mentre l'architettura determina il modo in cui l'agente ragiona, pianifica il proprio comportamento e mette in atto le proprie decisioni, ed è quindi strettamente correlata all'autonomia, reattività e proattività dell'agente, i ruoli sono legati alla sua capacità sociale. Un ruolo, infatti, determina quali messaggi l'agente è in grado di riconoscere, con quali messaggi deve rispondere ad essi ed in quale ordine deve avvenire lo scambio di messaggi con agenti che ricoprono altri ruoli. In pratica, i ruoli che un agente interpreta determinano i protocolli d'interazione a cui esso è in grado di conformarsi. Infine, le ontologie servono a strutturare e rappresentare la conoscenza dell'agente. Esse sono determinanti sia per il funzionamento interno del singolo agente sia per lo scambio di informazioni in un contesto multi-agente, in quanto agenti che conoscono le stesse ontologie condividono un vocabolario comune che ne rende più facile la interazione.

Agenti che ricoprono gli stessi ruoli, hanno la stessa architettura e condividono le stesse ontologie possono essere raggruppati in una unica *classe di agenti*[1]. Dal punto di vista computazionale gli agenti appartenenti ad una stessa classe hanno lo stesso engine (poiché la loro architettura è la stessa), lo stesso programma (poiché la logica del loro comportamento è la stessa, in quanto interpretano gli stessi ruoli e condividono la stessa conoscenza), e differiscono solo nel contenuto dello stato. La organizzazione degli agenti in termini di *classi di agenti* è estremamente utile per una corretta ingegnerizzazione del sistema. Essa infatti costringe lo sviluppatore a individuare tipologie di

[1] Anche se incluse nella caratterizzazione descrittiva di un agente ed integrate in essa da un punto di vista concettuale, le ontologie non sono integrate in D-CaseLP; al momento esso offre strumenti per associare ad una classe di agenti i ruoli interpretati e l'architettura, ma non le ontologie conosciute, che rimangono quindi implicite nel codice degli agenti.

1) **Modellazione**
   - Modellazione delle componenti indipendenti dal dominio: architetture, ruoli (per "ruolo" intendiamo sempre un "modello dei ruoli", in cui ogni ruolo è caratterizzato da un nome e da un insieme di protocolli di interazione con altri ruoli a cui è in grado di conformarsi), ontologie indipendenti dal dominio.
   - Modellazione della struttura statica del sistema ad agenti:
     – scelta dei ruoli necessari alla applicazione;
     – definizione delle ontologie dipendenti dal dominio e scelta di quelle indipendenti dal dominio;
     – raggruppamento dei ruoli in classi di agenti;
     – per ogni classe, scelta della architettura e delle ontologie più adatte;
     – scelta delle istanze di agenti necessarie per ogni classe.
   - Modellazione della dinamica del sistema ad agenti: definizione del programma relativo ad ogni classe di agenti e dello stato iniziale di ogni istanza.
2) **Verifica e validazione**
3) **Implementazione**
4) **Esecuzione**

TABELLA I
LE QUATTRO FASI DELLA METODOLOGIA D-CASELP.

agenti con caratteristiche comuni, rendendo la struttura del sistema multi-agente più chiara e comprensibile, e consente di definire il programma relativo ad una classe di agenti una volta sola e riutilizzarlo per ogni istanza di agente appartenente alla classe, rendendo il lavoro dello sviluppatore del sistema ad agenti più rapido e meno soggetto ad errori.

III. METODOLOGIA DI SVILUPPO

Lo sviluppo di sistemi ad agenti seguendo la metodologia D-CaseLP è caratterizzato dalle quattro fasi indicate nella tabella I. Nella fase di modellazione abbiamo separato la definizione delle componenti indipendenti dal dominio da quelle dipendenti da esso; le prime, infatti, una volta specificate o implementate possono essere riutilizzate in altri contesti, mentre le seconde, specificate durante la definizione della struttura statica del sistema, sono utilizzabili solo per la applicazione in via di sviluppo. La modellazione della struttura statica del sistema riguarda la determinazione di quali tra i ruoli e protocolli definiti nella fase precedente sono necessari alla applicazione, il loro raggruppamento in classi di agenti, la scelta delle architetture agente e delle ontologie adatte ad ogni classe e la individuazione delle istanze di agenti necessarie alla applicazione. Durante la modellazione della dinamica del sistema la specifica viene completata con le informazioni necessarie ad eseguire il prototipo: si definirà il programma che descrive il comportamento di ogni classe di agenti e lo stato iniziale delle istanze di agenti presenti nella applicazione.

La metodologia prevede una fase di verifica durante la quale le specifiche espresse in linguaggi per i quali esiste un interprete o un meccanismo di animazione vengono eseguite. Numerosi esperimenti sono stati condotti in CaseLP usando il linguaggio $\mathcal{E}_{hhf}$ che risulta essere particolarmente adatto alla modellazione di protocolli di comunicazione ad alto livello e la cui esecuzione consente di verificare determinate condizioni sugli stati finali degli agenti e sulla esecuzione del sistema.

Per quanto riguarda l'implementazione del prototipo, abbiamo già anticipato che gli agenti modellati usando gli strumenti offerti da D-CaseLP vengono tradotti in JESS, il quale viene eseguito integrandolo con la piattaforma JADE sviluppata in Java. Al momento gli unici agenti integrabili in D-CaseLP sono pertanto quelli implementati in JESS e provenienti da specifiche UML; stiamo però valutando la possibilità di integrare anche agenti implementati in Java e provenienti dalla traduzione semi-automatica di specifiche scritte in HEMASL [16], un linguaggio imperativo fornito da CaseLP per specificare architetture astratte e concrete, classi di agenti e loro istanze. In [15] si descrive una possibile traduzione da HEMASL a Java. Poiché JADE è implementato in Java riteniamo che gli agenti risultanti da una compilazione in Java di specifiche HEMASL possano essere facilmente integrati con agenti compilati in JESS ottenendo quindi un sistema multi-agente eterogeneo sia a livello di linguaggi di specifica (UML, HEMASL, $\mathcal{E}_{hhf}$, AgentRules), sia a livello di architetture (HEMASL è stato progettato appositamente per modellare architetture eterogenee) sia a livello di linguaggi di implementazione (Java, JESS ed in futuro Prolog).

L'esecuzione ed il debugging del prototipo sfruttano le funzionalità offerte da JADE quali la attivazione ed il monitoraggio di agenti remoti, la visualizzazione delle comunicazioni che intercorrono tra essi e la visualizzazione del loro stato interno.

IV. I PROTOCOL DIAGRAM D-CASELP

Il modello dei ruoli adottato in D-CaseLP è basato sui protocol diagram proposti da AUML [24], [25]. Rispetto ai protocol diagram AUML vengono poste diverse restrizioni che hanno lo scopo di eliminare ogni ambiguità e di consentire la completa automatizzazione del processo di traduzione da protocol diagram a regole JESS. Viene affrontato in maniera diversa anche il problema della parametrizzazione dei ruoli: in AUML i protocolli parametrici sono usati solo durante la fase di analisi, e vengono man mano raffinati fino ad ottenere, in fase di progettazione, un protocollo completamente istanziato in cui i soggetti sono singoli agenti e i messaggi sono interamente specificati. In D-CaseLP i soggetti dei protocolli sono ruoli, e non istanze di agenti, dalla specifica fino all'implementazione del prototipo; la associazione tra il ruolo e la classe di agenti che lo ricopre, e tra la singola istanza di agente e la classe a cui appartiene viene specificata a parte tramite gli architecture e gli agent diagram descritti nella sezione seguente. La ragione

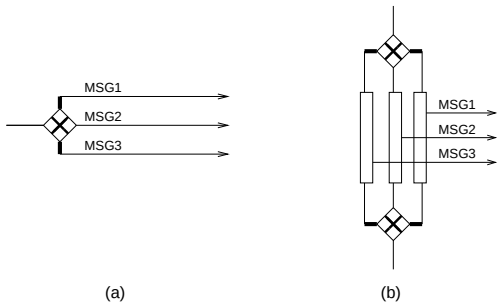

Fig. 1. Notazione implicita (a) ed esplicita (b) per l'utilizzo di connettori booleani: in D-CaseLP va usata la notazione (b).

della nostra scelta si spiega facilmente ipotizzando di avere un sistema ad agenti con due sole classi di agenti, C1 e C2, che incarnano ciascuna un solo ruolo, R1 ed R2 rispettivamente. In fase di progettazione, con D-CaseLP dobbiamo descrivere solo il protocollo di interazione tra R1 ed R2, indipendentemente da quante siano le istanze di C1 e C2. Seguendo l'approccio AUML, in fase di progettazione dovremmo invece replicare la descrizione del protocollo per ogni coppia di istanze di agenti che interagiscono, ovvero c1-inst $\times$ c2-inst volte, con c1-inst e c2-inst numero di istanze di C1 e C2.

Le restrizioni che abbiamo posto ai protocol diagram AUML per renderli traducibili automaticamente portano alla definizione di un "protocol diagram D-CaseLP" che è un protocol diagram AUML che rispetta il seguente insieme di vincoli:

1) In un protocollo non può esserci più di un ruolo iniziatore.
2) Se un protocollo coinvolge più di due ruoli, esso deve essere suddivisibile in sotto-protocolli, ciascuno tra due soli ruoli.
3) Un protocollo in cui compaia un connettore booleano per l'invio concorrente di messaggi deve essere rappresentato con la notazione esplicita in cui il thread con il connettore viene diviso in più thread, collegati dallo stesso connettore, da ognuno dei quali parte un messaggio. La figura 1 esemplifica questa rappresentazione.

Ad esempio, le figure 2 e 3 rappresentano due protocol diagram D-CaseLP corretti; il primo, B2C, corrisponde al FIPA Request Interaction Protocol [13] ed il secondo, B2B, corrisponde ad un semplice protocollo tra un negoziante ed un grossista. Nel protocollo B2C il ruolo B2C_Customer chiede al ruolo B2C_Trader di vendergli un certo quantitativo di merce (REQUEST). B2C_Trader può decidere se accordare la vendita (AGREE seguita dal messaggio INFORM per confermare la transazione avvenuta) o se rifiutare (REFUSE). Nel caso in cui non comprenda il contenuto del messaggio risponderà con NOT-UNDERSTOOD. Il protocollo B2B è ancora più semplice in quanto non prevede che il ruolo B2B_WholeSaler possa rifiutare la vendita o non comprendere il messaggio.

V. L'ARCHITECTURE E L'AGENT DIAGRAM D-CASELP

L'architecture e l'agent diagram D-CaseLP sono basati sulla proposta contenuta in [4]. L'architecture diagram ha lo scopo

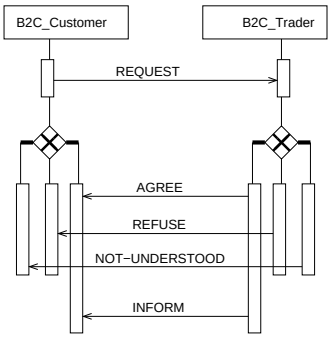

Fig. 2. Protocollo B2C.

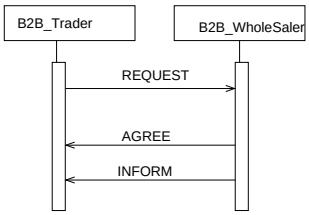

Fig. 3. Protocollo B2B.

di modellare le associazioni tra ogni classe di agente, la sua architettura (mediante la relazione has_architecture) ed uno o più ruoli che le istanze della classe possono interpretare (mediante la relazione plays_role). In questa prima versione di D-CaseLP tutte le classi condividono la stessa architettura JESS, una architettura deliberativa le cui strutture interne sono una lista di regole JESS, corrispondenti al programma dell'agente, ed una lista di fatti JESS, corrispondenti allo stato. L'engine per questa architettura è l'interprete del sistema esperto JESS.

La figura 4 rappresenta l'architecture diagram delle classi Apple_Customer, Apple_Trader ed Apple_WholeSaler che interpretano i ruoli illustrati nella sezione precedente.

L'agent diagram D-CaseLP viene usato per associare ad ogni istanza di agente la propria classe e lo stato iniziale.

La figura 5 rappresenta la definizione degli agenti Agent1 ed

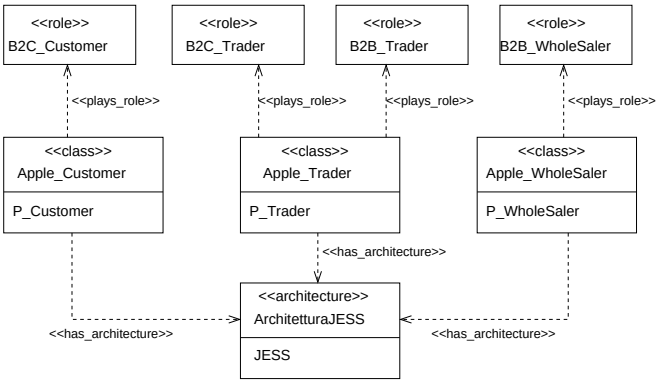

Fig. 4. Architecture diagram.

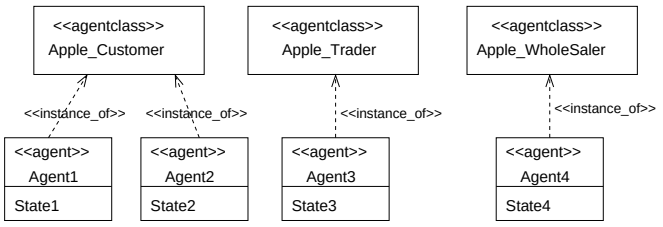

Fig. 5. Agent diagram.

- **Passo 1**: scelta di uno strumento grafico per la definizione di specifiche UML. Tale strumento deve consentire l'esportazione in formato XMI. In alternativa, è possibile utilizzare direttamente il formato testuale D-CaseLP-XML e saltare al passo 5.
- **Passo 2**: definizione in UML dei protocol, architecture ed agent diagram D-CaseLP descritti nelle sezioni IV e V.
- **Passo 3**: esportazione dei diagrammi D-CaseLP in formato XMI.
- **Passo 4**: traduzione della specifica XMI ottenuta al passo 3 in una specifica nel formato intermedio D-CaseLP-XML. Questa traduzione viene effettuata *automaticamente* utilizzando un file di trasformazione XSL.
- **Passo 5**: traduzione della specifica nel formato D-CaseLP-XML in gusci JESS che rispettano i protocolli e le associazioni definiti al passo 2 e nelle rispettive classi Java necessarie per il loro interfacciamento verso la piattaforma JADE. Questa traduzione viene effettuata *automaticamente* utilizzando un file di trasformazione XSL.
- **Passo 6**: riempimento dei gusci JESS con le condizioni e azioni appropriate.
- **Passo 7**: Attivazione di JADE, esecuzione del prototipo.

TABELLA II
DA UML A JADE: I PASSI DA SEGUIRE.

Agent2, istanze della classe Apple_Customer, Agent3, istanza di Apple_Trader ed Agent4, istanza di Apple_WholeSaler.

VI. DALLA SPECIFICA IN UML ALLA ESECUZIONE IN JADE

In questa sezione illustriamo i passi che uno sviluppatore deve seguire e gli strumenti che deve adottare per ottenere automaticamente "gusci" di regole JESS che rispettano il modello dei ruoli specificato. Con il termine "guscio" intendiamo regole parzialmente istanziate che, una volta completate dallo sviluppatore, possono essere eseguite. La tabella II e la figura 6 sintetizzano il processo di specifica e sviluppo da UML a JADE.

Il primo passo da seguire è individuare un ambiente grafico per la modellazione in UML che consenta di esportare le

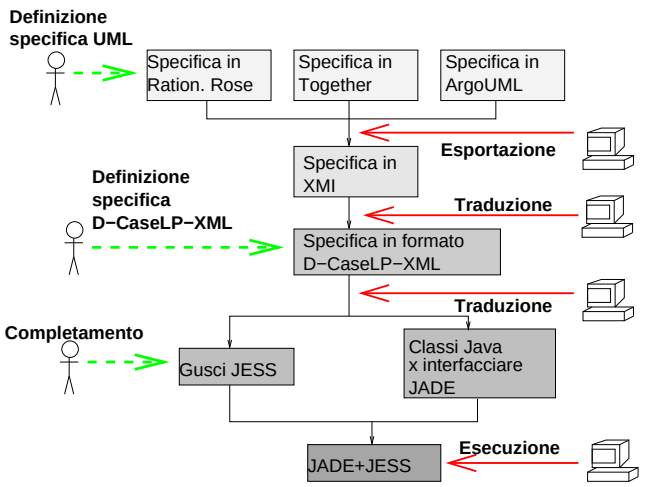

Fig. 6. Da UML a JADE: contributi manuali e traduzione automatica.

specifiche nel formato XMI. XMI (XML Metadata Interchange Format) è uno standard di fatto per lo scambio di metadati tra strumenti di modellazione basati su UML e strumenti ed archivi di metadati basati sulla Meta Object Facility OMG (MOF). La specifica può essere scaricata dal sito `http://www.omg.org/technology/documents/formal/xmi.htm`. Le nostre prove sono state effettuate utilizzando Rational Rose (`http://www.rational.com`), Together (`http://www.togethersoft.com/`) ed ArgoUML (`http://argouml.tigris.org`) e con tutti tre gli ambienti abbiamo ottenuto i risultati attesi. Se lo desidera, lo sviluppatore può utilizzare direttamente XMI o, preferibilmente, D-CaseLP-XML, un formato testuale basato su XML progettato da uno degli autori. Tale formato presenta un vantaggio rispetto all'uso diretto di XMI: esso offre infatti una maggiore leggibilità che ne rende più facile la comprensione e l'uso.

Se si decide di utilizzare un ambiente di sviluppo grafico, si disegnano i diagrammi che specificano il sistema (uno o più protocol diagram, un'architecture diagram ed un agent diagram), si esportano in formato XMI ed infine si trasformano in file di specifica nel formato D-CaseLP-XML, che serve quindi da formato intermedio. La trasformazione tra XMI e D-CaseLP-XML, entrambi basati su XML, avviene mediante un processo di traduzione automatico basato sulla tecnologia xslt che usa il Processor "Xalan" dell'Apache Foundation Group (`http://xml.apache.org/xalan-j/index.html`). Se lo sviluppatore vuole specificare i propri diagrammi direttamente nel formato testuale D-CaseLP-XML salterà le fasi precedenti.

A questo punto lo sviluppatore può trasformare, utilizzando un file XSL apposito, i file D-CaseLP-XML in file JESS costituenti il programma interno degli agenti specificati. Questo programma conterrà, per ogni classe di agenti, un insieme di regole parzialmente istanziate che rispecchiano i protocolli associati ai ruoli che la classe interpreta. Il codice JESS deve es-

sere completato per poter essere eseguito. La necessità del completamento deriva dal fatto che il protocol diagram D-CaseLP specifica il flusso dei messaggi, ma non sotto quali condizioni un messaggio deve, non deve o può essere spedito. Ad esempio, nel protocollo B2C è specificato che B2C_Trader può rispondere ad un messaggio REQUEST proveniente dal ruolo B2C_Customer con AGREE seguito da INFORM oppure (xor logico) con REFUSE oppure con NOT-UNDERSTOOD. La condizione per cui si può accettare o rifiutare la proposta non viene specificata nel protocol diagram: lo sviluppatore dovrà inserirla direttamente nel guscio di regola JESS. Tale guscio viene generato con numerosi commenti che indicano dove inserire le condizioni ed eventuali altre azioni da effettuare oltre all'invio di messaggi, per cui lo sviluppatore ha un valido supporto nel suo compito. Il processo di traduzione genera anche le classi Java necessarie per interfacciare le istanze di agenti JESS con la piattaforma JADE. Il processo di traduzione, sia nella sua fase da XMI al formato D-CaseLP-XML che in quella successiva da D-CaseLP-XML a JESS è guidato da un file di configurazione, anch'esso in formato XML, in cui vanno inseriti i path dei vari file di specifica del sistema. L'ultimo passo da compiere per poter eseguire il prototipo così ottenuto è quello di compilare i file Java generati ed integrarli con la piattaforma JADE. D-CaseLP fornisce due script, compile ed exec, che aiutano lo sviluppatore in questa fase.

VII. Esempio

Per esemplificare come applicare la metodologia D-CaseLP e come usare gli strumenti che esso offre, consideriamo un semplice sistema in cui avvengono interazioni di natura commerciale tra grossista ("WholeSaler"), negoziante ("Trader") e cliente ("Customer"). La merce che viene trattata è costituita da mele. Quando un agente cliente, che mangia mele ad intervalli regolari, si trova senza mele, ne acquista 10 da un agente negoziante; quando quest'ultimo si accorge di avere in negozio meno di 25 mele chiede ad un agente grossista di fornirgliene 100.

Supponiamo di disporre di una libreria di componenti indipendenti dal dominio che include in particolare i protocolli B2C e B2B illustrati nelle figure 2 e 3, la architettura JESS a cui si fa riferimento nella sezione V e due ontologie di termini relativi al commercio ed alla frutta[2]. Seguendo la metodologia esposta nella Sezione III, determiniamo innanzitutto i ruoli e i protocolli necessari all'applicazione. Poiché la applicazione coinvolge clienti, negozianti e grossisti avremo bisogno di quattro ruoli, B2B_WholeSaler, B2B_Trader, B2C_Trader e B2C_Customer (il negoziante commercia sia con i clienti, sia con i grossisti, e quindi gioca due ruoli distinti) che interagiscono adottando i protocolli B2C e B2B. Stabiliamo poi le classi di agenti del sistema associandole ai ruoli interpretati: avremo bisogno di una classe Apple_Customer che interpreta il ruolo B2C_Customer; una classe Apple_Trader che interpreta i ruoli B2C_Trader e B2B_Trader ed infine una classe Apple_WholeSaler che interpreta il ruolo B2B_WholeSaler. L'architecture diagram risultante è rappresentato in figura 4, in cui ad ogni classe è associata la architettura JESS. Infine, le istanze di agenti necessarie alla applicazione sono Agent1, Agent2, Agent3 ed Agent4 descritte in figura 5. Nell'esempio realmente condotto alcuni diagrammi sono stati definiti nel linguaggio D-CaseLP-XML per verificare che il processo di traduzione funzionasse sia partendo da UML, sia partendo da D-CaseLP-XML. In questo articolo, per ragioni di spazio, abbiamo descritto i diagrammi solo graficamente.

Poiché D-CaseLP non integra al momento stumenti di verifica, saltiamo il passo successivo della metodologia relativo alla verifica e validazione e passiamo direttamente alla implementazione del prototipo.

Utilizzando i meccanismi di traduzione offerti da D-CaseLP, otteniamo il codice JESS relativo alle tre classi Apple_Customer, Apple_WholeSaler ed Apple_Trader. Per esemplificare il risultato della traduzione ed il meccanismo del completamento manuale, ci soffermiamo sulla classe Apple_Customer. Nel programma associato a tale classe abbiamo inserito nella regola iniziale, che determina l'inizio di un nuovo protocollo B2C, la condizione (apples 0): ogni qualvolta un agente di classe Apple_Customer si trova senza mele, inizia un nuovo protocollo con un agente di classe Apple_Trader per acquistarne altre dieci; a questo scopo, il contenuto del messagio REQUEST è istanziato con la richiesta sell apples 10. Riportiamo qui sotto la regola iniziale completata (il codice aggiunto manualmente è scritto in **neretto**).

```
(defrule B2C_Customer_1
  (apples 0)
  =>
  (bind ?cid (newcid))
  (bind $?content (create$ sell apples 10))
  (send (assert (ACLMessage
  (communicative-act REQUEST)
    (role-sender B2C_Customer)
    (role-receiver B2C_Trader)
    (conversation-id ?cid)
    (content $?content))))
  (assert (state B2C_Customer_1 ?cid)))
```

Nella regola relativa alla ricezione del messaggio INFORM, che determina la conclusione positiva del protocollo, abbiamo aggiornato la quantità di mele posseduta dall'agente; in caso di REFUSE è stato introdotto un ritardo prima di effettuare una nuova richiesta. È stata infine aggiunta una regola JESS che simula il consumo di mele da parte dell'agente. Oltre al completamento dei programmi, nella fase di implementazione istanziamo anche gli stati dei singoli agenti; tali stati sono costituiti da fatti JESS inseriti nei file appositi. L'unico fatto

[2]La conoscenza che gli agenti hanno del dominio, inclusa in queste due ontologie, verrà "cablata" nel codice stesso degli agenti, per cui le ontologie risulteranno essere implicite.

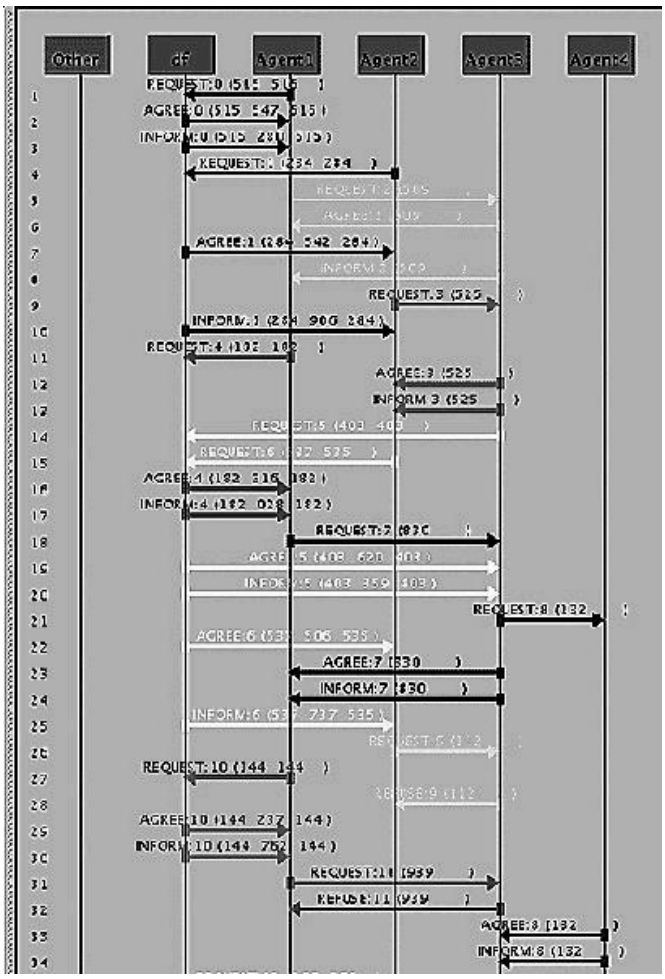

Fig. 7. Output dello strumento di monitoraggio Sniffer fornito da JADE.

inserito nello stato degli agenti è quello relativo alle mele in loro possesso: (apples N). Assegniamo ad Agent1 ed Agent2, agenti clienti, rispettivamente 10 e 15 mele mentre ad Agent3, di classe Apple_Trader, 40 mele; presupponiamo infine che l'agente Agent4 ne possegga infinite.

L'ultima fase prevista dalla metodologia è quella della esecuzione e debugging del prototipo. Per affrontarla, abbiamo compilato ed eseguito il MAS utilizzando gli script compile ed exec forniti da D-CaseLP. Abbiamo monitorato l'esecuzione utilizzando il tool grafico di JADE Sniffer che mostra l'evolvere dello scambio di messaggi tra gli agenti. L'output di Sniffer è mostrato in figura 7: oltre ai quattro agenti Agent1–Agent4 (gli ultimi quattro nella figura) abbiamo monitorato anche l'agente Directory Facilitator fornito da JADE (il secondo, identificato dal nome "df"); ogni qualvolta un agente vuole iniziare un protocollo chiede al Directory Facilitator se esiste un agente che interpreta il ruolo partecipante a cui deve essere inviato il messaggio.

VIII. Lavori collegati e sviluppi futuri

Esistono diversi ambienti di sviluppo di sistemi multi-agente che forniscono strumenti e facilitazioni per lo sviluppatore; per ragioni di spazio ne discutiamo solo alcuni, rimandando il lettore a [9], [27], [21] per approfondimenti sull'argomento.

AgentBuilder (`http://www.agentbuilder.com/`) è un prodotto commerciale sviluppato in Java da Reticular Systems, Inc, e consiste in due componenti principali: il Toolkit, che include strumenti per la gestione del processo di sviluppo del sistema multi-agente, ed il Run-Time System, che fornisce un ambiente per la esecuzione del software.

DESIRE [6] è un programma di ricerca condotto presso il Dipartimento di Intelligenza Artificiale della Vrije Universiteit di Amsterdam, Olanda. Gli obiettivi principali del progetto sono lo studio di sistemi multi-agente composizionali, lo sviluppo di componenti riutilizzabili e la definizione di metodologie di sviluppo.

IMPACT [11], [10], [2] è un progetto di ricerca internazionale condotto presso l'Università del Maryland, US. Lo scopo del progetto è lo sviluppo di una teoria e di un ambiente software per la creazione di agenti che accedono a fonti di dati eterogenee e distribuite e che interagiscono per condividere le informazioni accedute. Il sistema è implementato in Java.

Zeus [22] è un ambiente sviluppato da British Telecommunications per la specifica e l'implementazione di agenti collaborativi. Fornisce una metodologia di sviluppo e strumenti software per facilitare la progettazione rapida e lo sviluppo di sistemi multi-agente. È implementato in Java ed include una "Agent Component Library" che fornisce componenti riutilizzabili, gli "Agent Building Tools" per la specifica e la generazione del codice degli agenti, ed i "Visualization Tools" per il monitoraggio ed il debugging della applicazione.

Jack [14] è un ambiente di sviluppo commercializzato da Agent Oriented Software Pty. Ltd. ed include tutte le componenti dell'ambiente di sviluppo Java più estensioni specifiche per la implementazione di agenti con architettura BDI [26].

Le analogie tra D-CaseLP e gli ambienti citati sono principalmente legate alle funzionalità fornite dall'ambiente di sviluppo: tutti gli ambienti descritti consentono infatti l'esecuzione distribuita e concorrente degli agenti, implementano agenti predefiniti che forniscono servizi di pagine gialle e pagine bianche, ed offrono strumenti user-friendly ed interfacce grafiche per il monitoraggio e debugging dell'esecuzione del sistema. D-CaseLP eredita tutte queste funzionalità direttamente da JADE.

Rispetto a D-CaseLP, gli ambienti citati danno meno enfasi all'aspetto di eterogeneità degli agenti a livello di linguaggi di specifica ed architetture. Nella maggior parte di essi, infatti, l'architettura che gli agenti possono avere è unica ed è implicita nel linguaggio con cui gli agenti vengono specificati, anch'esso unico. DESIRE è l'unico ambiente a consentire la definizione di agenti con architetture diverse, offrendo modelli riusabili per la definizione di agenti con architetture reattive, proattive e BDI. Anche l'aspetto di validazione delle specifiche ad alto

livello, che in D-CaseLP è contemplato anche se non ancora integrato, è affrontato solo da DESIRE che pertanto sembra essere l'ambiente più affine a D-CaseLP.

Per quanto riguarda gli sviluppi futuri di D-CaseLP, abbiamo già osservato che D-CaseLP nasce dal progetto CaseLP e, pur mantenendone inalterato lo scopo, ne supera alcune limitazioni implementative dovute alla sua centralizzazione ed alla mancanze di una concorrenza reale tra gli agenti. Il prezzo di questo miglioramento è stata la perdita della possibilità di utilizzare i meccanismi traduzione semi-automatica da $\mathcal{E}_{hhf}$ ed AgentRules a codice eseguibile. Il linguaggio "target" di CaseLP è infatti Prolog ed i meccanismi di traduzione da $\mathcal{E}_{hhf}$ a Prolog e da AgentRules a Prolog non si possono banalmente adattare al linguaggio target di D-CaseLP, JESS, né tantomeno al linguaggio di implementazione di JADE, Java. La principale direzione di sviluppo prevede quindi la integrazione di Prolog in D-CaseLP ed il conseguente recupero delle funzionalità offerte da CaseLP. Un'altra direzione che intendiamo seguire consiste nel raffinare il meccanismo di traduzione da HEMASL a Java, abbozzato in [15], per integrare ed eseguire nella piattaforma JADE agenti specificati in HEMASL. Altri sviluppi riguardano infine il miglioramento dei linguaggi di modellazione e specifica: intendiamo estendere i protocol diagram D-CaseLP per includere la molteplicità nell'invio dei messaggi, supportare ed integrare esplicitamente le ontologie e dare la possibilità di definire agenti con architetture diverse da quella JESS.

RIFERIMENTI

[1] R. Albertoni. D-CaseLP: un ambiente distribuito per l'integrazione di agenti eterogenei. Master's thesis, DISI – Università di Genova, Genova, Italy, 2001. In Italian. Downloadable from `http://digilander.iol.it/richy74/tesi.pdf`.

[2] K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross, and V.S. Subrahmanian. IMPACT: A platform for collaborating agents. *IEEE Intelligent Systems*, 14(2):64–72, 1999.

[3] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with JADE. In *Intelligent Agents VII*, pages 89–103. Springer-Verlag, 2001. LNAI 1986.

[4] F. Bergenti and A. Poggi. Exploiting UML in the design of multi-agent systems. In *Engineering Societies in the Agents World*, pages 106–113. Springer-Verlag, 2000. LNCS 1972.

[5] M. Bozzano, G. Delzanno, M. Martelli, V. Mascardi, and F. Zini. Logic programming & multi-agent systems: A synergic combination for applications and semantics. In K.R. Apt, V.W. Marek, M. Truszczynski, and D.S. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 5–32. Springer Verlag, 1999.

[6] F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur. DESIRE: Modelling multi-agent systems in a compositional formal frameworks. *Journal of Cooperative Information Systems*, 1(6):67–94, 1997.

[7] G. Delzanno. *Logic & Object-Oriented Programming in Linear Logic*. PhD thesis, Università di Pisa, Dipartimento di Informatica, 1997. Technical Report TD 2/97.

[8] G. Delzanno and M. Martelli. Proofs as computations in linear logic. *Theoretical Computer Science*, 258(1–2):269–297, 2001.

[9] T. Eiter and V. Mascardi. Comparing environments for developing software agents. Technical Report INFSYS RR-1843-01-02, Wien Technical University, 2001. To appear in AI Communications.

[10] T. Eiter and V.S. Subrahmanian. Heterogeneous active agents, II: Algorithms and complexity. *Artificial Intelligence*, 108(1-2):257–307, 1999.

[11] T. Eiter, V.S. Subrahmanian, and G. Pick. Heterogeneous active agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255, 1999.

[12] Foundation for Intelligent Physical Agents. FIPA home page. `http://www.fipa.org/`.

[13] Foundation for Intelligent Physical Agents. FIPA Request Interaction Protocol specification. Experimental, 15-08-2001. Downloadable from `http://www.fipa.org/specs/fipa00026/`., 2001.

[14] N. Howden, R. Rönnquist, A. Hodgson, and A. Lucas. JACK intelligent agents - summary of an agent infrastructure. In *5th International Conference on Autonomous Agents*, Montreal, CA, 2001.

[15] S. Marini, M. Martelli, V. Mascardi, and F. Zini. HEMASL: A flexible language to specify heterogeneous agents. In A. Corradi, A. Omicini, and A. Poggi, editors, *Proceedings of WOA 2000. Dagli Oggetti Agli Agenti. Parma, Italy*. Pitagora editrice, Bologna, 2000.

[16] S. Marini, M. Martelli, V. Mascardi, and F. Zini. Specification of heterogeneous agent architectures. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII*, LNAI, Boston, MA, USA, 2000. Springer-Verlag.

[17] M. Martelli, V. Mascardi, and F. Zini. CaseLP: A complex application specification environment based on logic programming. In *Proc. of ICLP'97 Post Conference Workshop on Logic Programming and Multi-Agents*, pages 35–50, Leuven, Belgium, 1997.

[18] M. Martelli, V. Mascardi, and F. Zini. A logic programming framework for component-based software prototyping. In A. Brogi and P. Hill, editors, *Proc. of 2nd International Workshop on Component-based Software Development in Computational Logic (COCL'99)*, Paris, France, 1999.

[19] V. Mascardi. *Logic-Based Specification Environments for Multi-Agent Systems*. PhD thesis, Computer Science Department of Genova University, Genova, Italy, 2002. DISI-TH-2002-04. Downloadable from `ftp://ftp.disi.unige.it/person/MascardiV/Tesi/mythesis.ps.gz`.

[20] S. Miglia. Specifica ed implementazione di ruoli e protocolli d'interazione per agenti in D-CaseLP. Master's thesis, DISI – Università di Genova, Genova, Italy, 2002. In Italian. Downloadable from `ftp://gundam.vislink.it/stefano/thesis.pdf`.

[21] M. Nowostawski, G. Bush, M. Purvis, and S. Cranefield. Platforms for agent-oriented software engineering. In *Proceedings of the Seventh Asia Pacific Software Engineering Conference (APSEC'2000)*, pages 480–488. IEEE Computer Society Press, Los Alamitos, CA, 2000.

[22] H. Nwana, D. Ndumu, L. Lee, and J. Collis. ZEUS: A tool-kit for building distributed multi-agent systems. *Applied Artifical Intelligence Journal*, 13(1):129–186, 1999.

[23] H. S. Nwana and D. T. Ndumu. A perspective on software agents research. *The Knowledge Engineering Review*, 14(2):1–18, 1999.

[24] J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In *AOIS Workshop at AAAI 2000*, 2000.

[25] J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in UML. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering - First International Workshop, AOSE 2000*, pages 121–140, Limerick, Ireland, 2000. Springer-Verlag. LNCS 1957.

[26] A. S. Rao and M. Georgeff. BDI agents: from theory to practice. In *Proc. of International Conference on Multi Agent Systems (ICMAS'95)*, San Francisco, CA, USA, 1995.

[27] P-M. Ricordel and Y. Demazeau. From analysis to deployment: a multi-agent platform survey. In *ESAW'00 – First International Workshop on Engineering Societies in the Agents' World*, pages 93–105, Berlin, Germany, 2000. Springer-Verlag. LNAI 1972.